ATARI.RSC
The Atari Developer's Resource

Vol. IV, Issue 2
April-May 1991

The Bottom Line

Bill Rehbock, Director of Technical Services

CEPS, Networking, Video

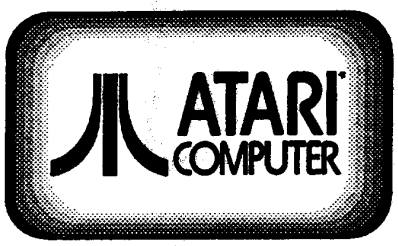
We are on a roll... There have been many exciting developments in the last two months. Networking is a reality, the MegaSTE has passed FCC Class B, and we have been recognized as a viable platform for mid to high-end digital typography.

In this issue, you will find details on the _FLK and _NET cookies. I encourage *everyone* to support file sharing and record locking as soon as possible. We designed the protocol to be as industry-standard as possible to promote the use of the supplied calls. As you read this, three separate Local Area Networks for the STe/TT are shipping.

All three support the same GEMDOS level locking calls, as will upcoming versions of TOS and other TOS networks that will ship in the future. I appreciate the cooperation that we received from Pascal Merle at Pam's Software, Chris Latham and Travis Guy at Application and Design Software, Charlie Seng at Versasoft, and Allan Pratt, Atari System Software Engineer, in getting the specification cast in stone.

There are a few unofficial file sharing and record locking protocols for TOS that are in scattered use about the globe. They can not and will not be supported. If you are currently shipping network software or a network application, please update your users to the correct standard as soon as possible.

The Matrix C32 VME 800x600 8-plane video card will be available to North American Atari dealers by late June. It will have a retail price of about $800 (U.S.). The card comes with 1 megabyte of RAM and an Intel graphics coprocessor and is completely user-programmable to support a wide variety of monitors, including the Atari PTC1426. There is one thing to beware of from an application standpoint: The card's



video memory is *not* arranged in interleaved planes like current ST and TT built-in video modes. Instead, it is PIXEL-PACKED. Writing directly to the card like you would with a built-in video mode produces interesting, but unwanted results. The card's drivers install as GDOS or FSMGDOS VDI screen drivers; there is very little Line-A support. Programs written with device independence in mind run perfectly on the cards. Please contact Gail Johnson if you would like to purchase a card for development purposes. ($575.00 U.S.)

At CEPS, the Corporate Electronic Publishing Show in Chicago, Atari's newly formed marketing and sales division, the Professional Systems Group, recently showed the Atari Direct To Press™ publishing solution to a gathering of thousands. The developers that demonstrated product at the show included ISD Marketing, Goldleaf Publishing, and Soft-Logik Publishing.

I would like to extend my thanks to Matthias Kurwig from ?K Computerbild and Dittmar Meynfeldt from DMC for flying staff and equipment to the show from Germany. The total cooperation from all those involved made the show a large success for us.

As many of you should know by now, Jerry Pournelle at Byte Magazine has received a TT. His initial impressions of the machine were quite favorable. He is lacking a supply of TOS application software to evaluate. If you are interested in the possibility of having your product reviewed in Byte magazine, I strongly urge you to forward an evaluation copy to him. Please drop me a note if you send a copy.

I wasn't kidding in the last issue when I said that time was going to move quickly! 1040STe's, MegaSTE's, and TT's are now moving smoothly into the dealer channel, with sales steadily increasing. The most common complaint from dealers now is, "Where is the new software?"

Inside This Issue

ATARI.RSC
The Resource File

CEO, PRESIDENT
ATARI CORPORATION
Sam Tramiel (408) 745-8824

GENERAL MANAGER
ATARI U.S.
Greg Pratt (408) 745-2146

TECHNICAL DIRECTOR
Bill Rehbock (408) 745-2082

DEVELOPER TECHNICAL SUPPORT
J. Patton (408) 745-2135
Mike Fulton (408) 745-8821
Fax: (408) 745-2094

DEVELOPER ADMINISTRATOR
Gail Johnson (408) 745-2022

SOFTSOURCE ADMINISTRATOR
Dan McNamee (408) 745-2024

Atari Corp.
1196 Borregas Ave.
Sunnyvale, CA 94088

Atari, the Atari logo, TT, and MEGA are trademarks of Atari Corporation.

ATARI.RSC edited by Mike Fulton

This newsletter was created using Pagestream on the TT030 computer with TTM 194 monochrome monitor, and printed using Ultrascript on the Atari SLM804 Laser Printer.

Developer Report

Gail Johnson

Hello ST/TT Developers!

Results from the January postcard mailing were sadly lacking. Of the ST/TT program developers who are on the newsletter mailing list, only 20% returned their cards. This does not include Canadians who have just joined us and are receiving registration materials with this newsletter. Since we are in the process of assigning developers to different categories in our new developer support program, it is important that I receive your cards even if you have nothing currently on the market. (Developers with nothing filled into database support fields may be automatically filtered out from mailings and other support features.)

If you did not receive a postcard, please make a photocopy of the form below, and please take a moment to fill it out and send it in (or just send a current sales brochure). If you have nothing on the market, please let me know of your development objectives.

Note: Noncommercial developers are now required to provide justification for all hardware orders. See the notice in this newsletter addressing that issue.

Greetings, Atari ST/TT Developers!

I am compiling a list of of the available products that you have developed for commercial use on the ST/TT line. Please take a minute to update us on your current titles and return the form to my attention. Thank you for your cooperation.

Company: _____

Primary Contact: _____

Developer ID #: _____

Address: _____

Phone: _____

Fax: _____

For each product your company produces, please list the following information: Product Title, Category (utilities, productivity, entertainment, education, etc.), Distributor(s), Retail Price

1) _____

2) _____

3) _____

If you have additional products, please list on separate sheet, and mail in with this form.

Send to: Atari Computer Corporation
Post Office Box 3427
Sunnyvale, Calif. USA 94088-3427
Attn: Gail Johnson

Running As Program Or Accessory

by Ken Baderfscher & Mike Fulton

Recently, several applications have been released which have the ability to be used as either a desk accessory or program. This has prompted other developers to ask, "how do I do that?".

There are two main things to accomplish. The first is to figure out which way the application was loaded, as program or accessory. Second is using that information and taking the proper steps in handling how the application works. For example, an application shouldn't install a desk accessory menu item, and an accessory shouldn't install a menu bar, so before doing these types of things, your application would have to check how it was loaded and act accordingly.

But besides such obvious things, there are many things which an application might commonly do which desk accessories should avoid. For more detailed information about this, see the article titled "***Desk Accessory Guidelines***" elsewhere in this newsletter.

Starting with the way a program is loaded, as defined in the GEMDOS manual, there are several differences that apply to accessories, and these differences are the key to determining how an application has been loaded.

(This article assumes you are familiar with the contents of the GEMDOS manual and the Pexec Cookbook. So if you haven't read these recently, now would be a real good time to refresh your memory.)

A program which can run as a program or desk accessory has to have a startup module which can determine which way the application was loaded and act accordingly. There are two things that need to be checked by such a startup module.

First of all, register A0 is always cleared at startup for programs run by the GEMDOS Pexec() function, but for accessories, A0 will contain a pointer to the basepage. Therefore, the first thing a customized startup module should do is test A0. If it is NULL, then you are running as a program. Otherwise, A0 will point to your accessory's basepage, and you should check the long value at 36(A0), which should be NULL for a desk accessory (for programs, this contains a pointer to the basepage of the parent process).

Immediately after the startup module decides if it is a program or accessory, it should set a flag of some sort that the rest of the program can check, and then continue to do the rest of the regular initialization, such as setting up the stack and so forth.

Desk accessories must setup their own stack before using it, because their stack pointer contains garbage when they are first started up. Programs need to move the stack pointer down from where it starts (usually the end of a large block of memory) and then shrink their TPA to the amount of memory they need (see the Pexec Cookbook). Desk Accessories don't need to shrink their TPA because they are loaded by GEM AES into just exactly the amount of memory they need.

You should study the regular startup modules your compiler uses for programs and accessories to insure that what you come up with will do everything required by your compiler as well as the system.

The Lattice C v5, Lattice C/TT, and Turbo C compilers come with auto-detecting startup modules that can be used as an option right out of the box. If you use one of these compilers, see your manual for more information.

One big thing to keep in mind is that there are a lot of programs that have no business being accessories, and a lot of accessories that would serve little purpose as programs. It is therefore recommended that you strongly consider all the various factors when deciding if your application should be able to work both ways. Don't make it work both ways just because you can.

As we said earlier, there are certain things which will have to be done differently depending on the current mode of operation. The earlier reference to standard GEM menu bars is probably the number one example. Some applications avoid this by simply not using menus regardless of the operating mode, but this may not be appropriate for everything.

The way other applications have handled this is to display the menu title bar within their own window, and then wait for a mouse click on a menu title. When one is received, then the application starts tracking the user's mouse movements through the menus in a manner similar to the regular AES menus. When a menu item is selected, then the program stops tracking the mouse through the menus and handles the user's choice for whatever menu item was selected (or not, if no item was selected).

Another thing to consider is that an application that goes both ways should be thoroughly tested both ways to shake out any bugs in general operation as well as being able to operate in either mode. Don't forget to test everything else because you've been concentrating so hard on getting it to work both ways.

Desk Accessory Guidelines

by Mike Fulton & Ken Badertscher

1) In general, all memory allocation should be done at initialization time, before the menu_register() call. Use static blocks of memory in your BSS space if your memory allocation requirements are known in advance. The only exceptions would be under circumstances where the accessory is certain it will be releasing the memory before anything else would have a chance to allocate memory or Pterm() out from under the accessory, including the operating system or other accessories.

With this in mind, accessories should open, read or write, and then immediately close disk files. Do not leave open files laying around. Also, accessories should never allocate memory on the fly to use as a buffer for reading from disk files.

When a process terminates, all memory allocated while the process was running is released, including memory allocated by accessories (which are not processes), and OS services that allocate memory on behalf of a program.

Keep in mind that task switching between desk accessories, the current application, and AES can occur during any AES call, so remember that whenever your accessory makes any AES call, something else could get control and do memory allocation of their own, causing memory fragmentation.

2) Desk accessories must always do screen output of any kind through the operating system.

As a general rule, GEM VDI should be used for screen output, but under certain circumstances GEMDOS or BIOS character-based text output may be acceptable, as long as the accessory is careful about sending redraw messages for the affected areas of the screen.

3) Desk accessories must not install themselves into interrupt vectors, because this will cause problems when the user does any resolution changes.

If a desk accessory needs to access something at interrupt level, then it should be accompanied by a TSR which goes into the AUTO folder, and which communicates with the desk accessory to provide the necessary information and/or functions.

The way to so this is for the TSR to install a cookie in the cookie jar that provides a vector for the desk accessory to jump through to get the desired information. The accessory would look for the cookie and then jump through the vector whenever it needed to update its information. This is preferable to installing a cookie that points at the TSR's data space, since in a memory protected system, that data space may not be directly accessible to the accessory. Also, this method doesn't rely on anything anything that's going to get zapped on a resolution change.

With older versions of TOS before the "new desktop", the memory used by accessories was not released on a resolution change, so accessories could get away with stealing interrupts because the memory continuing the interrupt handler normally wouldn't get zapped (unless a program wrote into memory it didn't own).

However, with the current versions of TOS with the "new desktop", memory belonging to accessories is released whenever a resolution change is made. Therefore, the interrupt vector would be left pointing at memory which is no longer owned by anything, and which would probably get overwritten by something fairly quickly. Most likely what will happen here is a mysterious crash because the interrupt handler will get zapped.

4) Whenever possible, desk accessories should operate in a movable window instead of a statically positioned dialog box. There are a number of desk accessories currently available which lock themselves into the middle of the screen and stay there until dismissed, and there is usually no good reason for it.

As resolution improves and screen sizes get larger, there is the increasing possibility of users wanting to have a number of non-overlapping windows spread out across their desktop. If your desk accessory is the only thing accessible from the time you select it from the menu until the time you make it go away, people are going to use something else instead of your desk accessory.

When you put your accessory into a movable window, there is no reason why you would not still be able to use GEM resources and objects for user interaction, although it may require the use of a customized form_do() function. Sample source code for a replacement form_do() function is available in the Atari Developer areas of GEnie and Compuserve. This can copied or modified as needed for your purposes.

5) Accessories should NEVER speak until spoken to. Wait until the user selects the accessory's menu choice before doing anything on screen. Accessories that put up sign-on messages can cause the system to hang or reboot, and are just generally annoying to many users.

As with any rule there are exceptions, but generally it is a bad idea for an accessory to make unnecessary noise at startup. It is much better to wait until the first time the user calls the accessory; any sign-on or error messages can be displayed at that time.

Remember that we are talking about accessories, not TSR's or applications.

Graphics on the Portfolio And The .PGC File Format

by Don Messerli, Software Vineyard

Programmers and hardware designers have been striving to advance the art of computer graphics for years. It is no surprise then, that serious graphics had to come to the Atari Portfolio sooner of later.

Portfolio Graphics

Despite the small size of the Portfolio's LCD screen, some rather impressive still pictures can be displayed. As you probably know, the Portfolio's screen has a text mode of 40 columns by 8 rows; made up of 6x8 pixel characters. There is also a graphics modes available: 240x64 pixels, which I will discuss.

I will not go into the details of how to put the Portfolio into graphics mode. Refer to the Atari Portfolio Technical Reference Guide for this information.

Pixels can be manipulated by using the usual display BIOS interrupts (int 10h services 0Ch and 0Dh). Using BIOS to set and clear pixels is usually considered very slow. However, the Portfolio's BIOS service is fairly quick compared to a desktop PC, because it doesn't have to worry about all the different possible graphics modes presented by the various display standards (CGA, MCGA, EGA, and VGA). Note that graphics are not preserved if one of the internal applications is "popped-up" over the top of it.

For dealing with larger portions of the screen or the entire screen, writing directly to display memory is the way to go. The Portfolio's display memory lies at B0000h. I have found that a mirror lies at address B8000h so writing to it has the same exact results. Unlike CGA, the Portfolio's display memory is laid out in a very logical fashion. Since we are dealing with monochrome graphics, we only need 1 pixel to represent a dot on the screen. Eight pixels per byte. 30 bytes per screen row of 240 pixels. If you multiply 30 bytes per row by 64 rows of dots, you will see that it takes 1920 bytes to represent a complete graphics screen. Below is a diagram showing this:

```
        column 0  column 1 ....  column 239
------------------------------------------
row 0   01101011  10010010 ..... 10011101
        byte 0    byte 1         byte 29
                                   . .
row 1   00111001  00001110          . .
.       byte 30   byte 31
.
.
row 63  00000000                  11011001
        byte 1890 .............   byte 1919
```

If we want to transfer an image from memory or disk to the screen, we just move 1920 bytes into the display's screen buffer. That simple, right? Well, not exactly. The LCD controller in the Portfolio keeps its own copy of the screen image which is not directly mapped into the memory space of the 8088 CPU. The bytes must be copied from our display buffer to the LCD controller's. This is accomplished by calling the Portfolio's BIOS routine to perform a "screen refresh" (int 61h service 12h). This is the same refresh whose behavior can be modified through the Portfolio's Setup application.

The screen refresh looks like multiple wipes (as in wipes and fades) taking place on different parts of the screen simultaneously.

The Need For A Standard File Format

After playing around and exploring the Portfolio's graphic capabilities, I decided to write a program that would give Portfolio owners a glimpse of those capabilities, and of things that lie ahead. I created some sample graphics screens. They were actually scanned images of the Atari Fuji and Portfolio logos from various advertisements and the box it came in. I wrote a program in assembler called PGSHOW. It displays a graphics screen and waits for a keypress or a specified time interval, then quits. I stored the graphic images on disk in files with a .PGF extension. Of course, they were all 1920 bytes in size. It was just a matter of saving memory to disk and then reading it back in from disk later.

As everybody who owns a Portfolio knows, RAM memory cards are expensive. Every bit of storage space savings is applauded by the user community. It wasn't long before I realized that some form of compression scheme for storing images on "disk" was necessary. There are many complex data compression schemes out there. However, I wanted one that would be quick, require little additional code, and be simple for others to implement. It ended up being the PGC (Portfolio Graphics Compressed) file format.

Compression can save quite a bit of space, typically from 20 to 80%, depending on the file. However, because of the compression algorithm I have chosen, it is possible for a file to become larger. This happens with files that have a lot of random dots on the screen with little repetition. Digitized photos are the culprits. Fortunately, because of the small size of the Portfolio screen, we probably won't be seeing too many of them.

The PGC file format has quickly become a standard for Atari Portfolio graphics. I urge all developers to use the PGC format. The user community will applaud your efforts and we will all benefit. Along with the handful of programs I have written (detailed below), other developers have already begun to support the PGC format. B.J. Gleason has incorporated support for PGC files in his public domain BASIC interpreter, PBASIC. PBASIC is the most significant public domain program, and arguably the most significant third-party application of any kind currently available for the Portfolio.

The .PGC File Format

The following is an excerpt from the PGC file specification which is documented in the file PGCSPEC.ZIP. See the end of this article for availability of any files I have discussed.

A PGC file consists of two parts; the header and the picture data.

Header - The first 3 bytes of the file are the PGC header. This is the only way to tell if the file is a PGC file or not. This should always be checked when reading PGC files. Your decoder routines could take a quick trip into never-never land if you try to decode a spreadsheet file.

The three bytes that make up the header are the ASCII codes for 'P' and 'G' followed by the revision number in hex. The current revision is 1. These three bytes should be:

"PG\001"

Data - The picture data is made up of repeating sets of single index bytes followed by data bytes. The index byte tells the decoder how to handle the data that follows. The key is whether the high bit is set or not. The maximum number of data bytes following an index byte is 127.

High Bit Set -- If the high bit is set, it indicates that the data is a string of identical bytes. The low 7 bits indicate how many times to repeat the following byte, from 0 to 127.

Example: 86 FF

High bit of index byte is set, low 7 bits = 6, so repeat data byte (FF) 6 times.

High Bit Clear -- If the high bit is NOT set, it indicates that the data following is a string of unique bytes and should be copied as is. The low 7 bits indicate how many bytes to copy, from 0 to 127

Example: 0A FF 01 10 09 1A BB CE D0 FF 1A

High bit of index byte is NOT set, low 7 bits = $0A, so copy next $0A (10 decimal) bytes as is.

Encoding and Decoding PGC Files

Writing a decoder is fairly straightforward. Listing 1 contains a C routine to read a PGC image from a file into a memory buffer. Of course, this buffer could be display memory. Writing an efficient compressor is a little more difficult. Listing 2 contains a C routine to write a PGC file to disk from an image in memory. PGLIB, discussed below, contains routines to both read and write PGC files.

Pushing the Standard

I have also written several other programs to propagate support for the PGC format and graphics on the Portfolio in general. All of these files are available online on the Atari BBS, GEnie, and CompuServe.

PGEDIT 1.1 - A program that runs on a desktop PC and requires 512K, EGA or VGA card, and a Microsoft compatible mouse. Allows you to create and edit PGC images on a PC. It also included support for importing MacPaint and Print Shop images.

PGCOMP 1.1 - A program that converts (compresses) PGF files to PGC files.

PGCHECK 1.0 - A program that checks a PGC file for proper format and optimum compression. This was designed as a tool for programmers trying to implement their own PGC routines.

PGLIB 1.0 - A library of canned routines you can add to your (Microsoft or Turbo) C programs to add graphics and PGC support.

PGC Grabber 1.0 - Created by popular demand. A program that runs on the Macintosh and allows you to grab the upper left-hand corner of a MacPaint picture and save it as a PGC image.

PGCSPEC.ZIP - The complete PGC file format specification.

About the Author:
Don Messerli is a registered Atari Portfolio Developer and President of Software Vineyard. He can be reached through the following online services:

Compuserve: 72500,1671
GEnie: DMESS

***Note:** Publication of this standard is not necessarily an endorsement by Atari Corporation. We encourage contributions to the newsletter by developers who have ideas they wish to share with the developer community.*

See Listing #1 below and Listing #2 on Page 8.

Listing #1

```
/*
    Routine to read a PGC image from disk into a
    memory buffer. Infile must have already been
    opened and the PGC header read and verified.
*/

ReadPGC(*infile, *p)
FILE*infile;                    /* file handle */
char*p;        /* pointer to destination buffer */
{
unsigned int c, idx;
int n = 0;

    do
    {
       c = fgetc(infile) & 0xff;   /* Index byte */

/* If high bit set, write data bytes index times */

       if (c & 0x80)          /* Is high bit set? */
       {
          idx = c - 128;      /* Figure out count */
          c = fgetc(infile);    /* Get data byte */
          while(idx--)
            p[n++] = c; /* Copy byte 'idx' times */
       }

/* Must be a string of bytes */

       else
       {
          index = c;                  /* Get # bytes */
          while(index--)  /* Read & Copy # bytes */
            p[n++] = fgetc(infile);
       }
    }
    while(n < 1920);
}
```

PGC File Format Listing #2

```
/*  Routine to write a PGC file.
    outfile must be opened for writing  */

#define LONG 127 /* Size of longest run of bytes */

char  header[] = "PG\x01";

/*  'p' - Pointer to 1920 byte picture data buffer
    'outfile' - file handle of picture file
*/

WritePGC(*outfile, *p)
FILE*outfile;
char*p;
{
int offset, run, unique;
char  buf[128];

/* Write the file header */

    fwrite(header, strlen(header), 1, outfile);
    offset = unique = 0;
    do
    {
       do
       {
          run = 0;  /* Check for a run (127 max) */

          while((p[offset+run] == p[offset+run+1])
                && (run < LONG-1)
                && (offset + run < 1919) )
             ++run;

          if(run > 0)          /* if there's a run */
          {                 /* check for a previous */
             if(unique)        /* unwritten string */
             {

/* If there's a string, write index byte */
/* and the string of bytes... */

                fputc(unique & 0x7f,outfile);
                fwrite(buf,1,unique,outfile);
                unique = 0;
             }

/* ...and then write the run. */
/* Run is 1 less than actual run length */

             fputc(run+129,outfile);
             fputc(p[offset+run],outfile);
             offset += (run + 1);
          }
          else
             buf[unique++] = p[offset++];
       }
       while(unique < LONG && offset < 1920);

       if(unique)           /* check for a string */
       {

/* If there's a string, write index byte */
/* and then the string of bytes */

          fputc(unique & 0x7f, outfile);
          fwrite(buf,1,unique,outfile);
          unique = 0;
       }
    }
    while(offset < 1920);

/* Is there still a string to be written ? */
/* If there's a string, write index byte */
/* and the string of bytes */

    if(unique)
    {
       fputc(unique & 0x7f, outfile);
       fwrite(buf,1,unique,outfile);
    }
}
```

Calendar of Upcoming Events

June 15-16
Pacific Northwest AtariFest
Steveston School
Richmond, B.C.
For details, contact Terry Schrieber at (604) 275-7944

June 29-30
Great Lakes Atari Computer Users Conference
Erie, PA
For details, contact Patty Marshall at (412) 225-8637

July 20
Blue Ridge AtariFest
Asheville, NC
For details, contact Sheldon Winick at (704) 251-0201

July 27
MIST AtariFest III
Indianapolis, IN
For details, contact Bill Loring at (812) 336-8103

August 23-25
Duesseldorf Atari Messe
Duesseldorf, Germany
For details contact Bill Rehbock

September 14-15
Southern Califnornia Atari Faire, Version 5.0
Glendale Civic Auditorium
Glendale, CA
For details, contact John King Tarpinian at
(818) 246-7286

October 12-13
WAACE Show
For details, leave GEnie Email to J.D.BARNES, or mail request to:
WACCE Vendor Coordinator
C/O John D. Barnes
7710 Chatham Rd.
Chevy Chase, MD 20815

October 21-25
Fall Comdex '91
Las Vegas Convention Center
& Sands Convention Center
Las Vegas, Nevada
For details, contact Bill Rehbock

November 23-24
Chicaco Atari Computer Show
Chicago, IL
For details, contact:
Larry Grauzas
P.O. Box 8788
Waukegan, IL 60079-8788
(708) 566-0671

Screen Grid Drawing Algorithm

Mike Fulton

Remember not too long ago when Atari announced that the Line-A interface would not be supported for new graphics modes in future machines? (See last issue for details.) One thing that's come to light since then is that some developers had been using Line-A for drawing the dots for their screen grids, and were reluctant to use GEM VDI instead because of speed considerations.

There's a solution which takes a new approach to this problem, and operates instantaneously even on a basic 8MHz 68000 machine. It is much faster than using either VDI or Line-A to draw dot by dot.

The basic idea is to stop drawing each and every dot of the grid separately. This is an awful lot of calls, and it gets even worse with high resolution monitors. With a little preparation, it's possible to draw an entire line of dots at the same time, which is much, much faster.

Let's say you have an 8" x 6" area represented on screen, with a grid spacing of 1/8". That's 64 columns by 48 rows, or a total of 3072 dots. Or to put it another way, 3072 Line-A or GEM VDI calls if you draw each dot individually.

But if you draw a line at a time, you can do it with just 48 VDI calls, one for each scanline that the grid appears on. With all but the tightest grid spacing, the grid appears on screen instantly from the user's viewpoint, even on a basic 8MHz 68000 machine. Plus, with this method, if the system has a blitter, it is automatically taken advantage of.

With a grid, you are drawing dots with a certain horizontal spacing. It is quite therefore quite easy to create a monochrome raster form that is basically a 1-scanline high grid template. This template would have blank pixels except where the grid dots are supposed to be. Once your template is set up, then to draw your grid, you simply blit this form onto the screen once for every scanline where your grid is required. If your grid is not exactly the same on each row it appears, then you could simply have multiple templates so that you could use whichever is appropriate at the moment.

This is a method that should run on any TOS machine, any monitor, and any resolution in any video mode. It uses 100% standard GEM VDI and AES calls and C bindings, and absolutely no Line-A calls at all.

The sample program shows one way to use this method, and for comparison purposes, it also shows off different dot-by-dot drawing methods using GEM VDI and Line-A, so you can see for yourself the speed differences all in one place. However, please note that although the VDI calls could be made faster by using customized bindings instead of the standard ones supplied with the C compiler, this was not done here in order to keep the example short.

This method is so fast, just to make sure you know when it starts and stops, the sample program has a pair of alert boxes surrounding all of the grid drawing calls. The sample program is shown in listing #1, and is also available in the ATARI.RSC Roundtable on GENIE. The filename is GRIDDEMO.ARC.

Some developers may look at the sample code and be concerned because their grid doesn't work out to exactly every 8th pixel. Sometimes the spacing might be 4 dots, sometimes 5, or something completely different, depending on how mapping inches or millimeters to pixels on screen rounds off.

Never fear, because that doesn't really matter. If your grid spacing is such that you sometimes have to have an extra blank column or row between dots, it's really not a problem. First of all, you can easily handle any horizontal spacing requirements when you are creating your grid template. If you only want the 5th, 9th, and 22nd pixels in your horizontal grid spacing, then set only those pixels in your template. If you need every 4th pixel, then set only those pixels. Whatever your spacing is, it shouldn't present any particular problem.

And for uneven vertical spacing, you could use a lookup table containing the scanlines where you need to draw each row of the grid. For example, you would set up a lookup table containing the y-axis values of the scanlines used by whatever vertical spacing your grid requires, such as the example below:

```
WORD *grid_scans, num_gridscans;

grid_scans =
   (WORD *)Malloc(screen.yres*sizeof(WORD));

grid_scans[0] = 4;
grid_scans[1] = 8;
grid_scans[2] = 13;
grid_scans[3] = 17;
num_gridscans = 4;
```

Then when you draw your grid, simply loop through and do each specified scanline, in loop like this.

```
 for( line = 0; line < num_gridscans; line++ )
{
   y = grid_scans[line];
   blit_gridline( y );
}
```

Please note that the above is pseudo-code for example purposes only, and is not part of the demonstration program nor directly workable as is. Also, the example program has the grid initialization immediately before the grid drawing, whereas in a real application you'd want to do this only when you changed or set your grid spacing, or when you changed screen magnification or did something else that affected the way the grid appears on screen.

Also, as was mentioned earlier, if you need to have different grid templates for different parts of the grid, then you can do that fairly easily.

In order to access Line-A in a way that is as portable as possible with different compilers, the use of inline assembly directives or other compiler specific functions was avoided. Therefore, a small amount of assembly

language is required to make the actual calls to Line-A. However, in order to keep the amount of assembly language to a minimum, a structure was used in the C listing to access the Line A variable table, and only the actual calls to Line-A are coded in assembly. Listing #2 contains the short assembly language routines used to actually call Line-A. The listing is designed for either DEVPAC or the assembler included with the Lattice C compiler, but should be easily transferable to whatever assembler your C compiler will work with.

If you have any questions regarding the use of this method, please feel free to call me at (408) 745-8821 or leave me EMAIL on GENIE to MIKE-FULTON.

Screen Grid Drawing Algorithm Listing #1

```
/*****************************/
/* GRIDTEST.C by Mike Fulton */
/*****************************/

#include <osbind.h>
#include <vdiwork.h>

#define LATTICE 1

#if LATTICE
#include <vdi.h>
#include <aes.h>                    /* Lattice C */
#define BUSY_BEE BUSYBEE
#else
#include <obdefs.h>
#include <gemdefs.h>
#define MFDB FDB
#endif

#define WORD short


/*****************************************/

typedef struct la_table {
  WORD  planes, width;
  WORD  *contrl, *intin, *ptsin, *intout, *ptsout;
  WORD  stuff[50];
} LINEA_TABLE;

VDI_Workstation screen;

WORD gl_apid,handle,clip[4],contrl[12],intin[256],
     ptsin[256],intout[256], ptsout[256];

char gridbuf[300];

MFDB grid, scrn = { 0L, 0, 0, 0, 0, 0, 0, 0, 0 };

/*****************************************/

void
ext_inquire( dev )
VDI_Workstation *dev;
{
WORD out[57];

  vq_extnd( dev->handle, 1, out );
  dev->screentype = out[0];
  dev->bgcolors = out[1];
  dev->textfx = out[2];
  dev->canscale = out[3];
  dev->planes = out[4];
  dev->lut = out[5];
  dev->rops = out[6];
  dev->cancontourfill = out[7];
  dev->textrot = out[8];
  dev->writemodes = out[9];
  dev->inputmodes = out[10];
  dev->textalign = out[11];
  dev->inking = out[12];
  dev->rubberbanding = out[13];
  dev->maxvertices = out[14];
  dev->maxintin = out[15];
  dev->mousebuttons = out[16];
  dev->widestyles = out[17];
  dev->widemodes = out[18];
}

/* Open a virtual workstation for the screen */

WORD
open_vwork( dev )
VDI_Workstation *dev;
{
WORD i, in[11];

  in[0] = Getrez() + 2; /* Device to be opened */
  dev->dev_id = in[0];
  for( i = 1; i < 10; in[i++] = 1 );
  in[10] = 2;
  i = graf_handle( &dev->wchar, &dev->hchar,
                   &dev->wbox, &dev->hbox );
  v_opnvwk( in, &i, &dev->xres );
  dev->handle = i;
  if( i )
    ext_inquire( dev );
  return(i);
}

void
clearscreen()
{
  graf_mouse( M_OFF, 0L );
  v_clrwk( screen.handle );
  graf_mouse( M_ON, 0L );
}

void
w_linea()
{
extern void lineadot();
extern LINEA_TABLE *linea_ptr();
register LINEA_TABLE *line_a_table;
WORD x, y;

  form_alert(1,"[0][Using line-a calls ][Ok]");
  graf_mouse( M_OFF, 0L );
  line_a_table = linea_ptr();
  line_a_table->intin[0] = 1;

  for( x = 0; x < screen.xres; x += 8 )
  {
    for( y = 0; y < screen.yres; y += 8 )
    {
      line_a_table->ptsin[0] = x;
      line_a_table->ptsin[1] = y;
      lineadot();
    }
  }
  graf_mouse( M_ON, 0L );
  form_alert( 1, "[0][Grid drawn ][ ok ]" );
}

void
w_pline()
{
short x, y, pxy[8];

  form_alert(1,"[0][Using v_pline() call ][Ok]");
  graf_mouse( M_OFF, 0L );
  for( x = 0; x < screen.xres; x += 8 )
  {
    pxy[0] = pxy[2] = x;
    for( y = 0; y < screen.yres; y += 8 )
    {
      pxy[1] = pxy[3] = y;
      v_pline( screen.handle, 2, pxy );
    }
  }
  graf_mouse( M_ON, 0L );
  form_alert( 1, "[0][Grid drawn ][ ok ]" );
}
```

continued on page 10

Screen Grid Drawing Algorithm Listing #1
Continued from Page 9

```
void
fast_grid()
{
short x, y, pxy[8];

   form_alert( 1, "[0][Using FAST method ][ ok ]" );
   grid.fd_addr = (void *)gridbuf;
   grid.fd_w = screen.xres + 1;
   grid.fd_h = 1;
   grid.fd_wdwidth = (grid.fd_w + 15) / 16;
   grid.fd_nplanes = screen.planes;
   grid.fd_stand = 0;
   grid.fd_r1 = grid.fd_r2 = grid.fd_r3 = 0;

   for( x = 0; x < 300; gridbuf[x++] = 0x08 );

   pxy[0] = pxy[1] = 0;
   pxy[2] = grid.fd_w;
   pxy[3] = pxy[4] = pxy[5] = pxy[6] = pxy[7] = 0;

   graf_mouse( M_OFF, 0L );
   for( y = 0; y < screen.xres; y += 8 )
     {
     pxy[5] = pxy[7] = y;
     vro_cpyfm(screen.handle,3,pxy,&grid,&scrn );
     }
   graf_mouse( M_ON, 0L );
   form_alert( 1, "[0][Grid drawn ][ ok ]" );
}

void
main()
{
   gl_apid = appl_init();
   graf_mouse( ARROW, 0L );
   if( open_vwork(&screen) )
     {
     clip[0] = clip[1] = 0;
     clip[2] = screen.xres;
     clip[3] = screen.yres;
     vs_clip( screen.handle, 1, clip );
     clearscreen();
     w_pline();
     clearscreen();
     w_linea();
     clearscreen();
     fast_grid();
     v_clsvwk( screen.handle );
     }
   appl_exit();
   Pterm0();
}
```

Screen Grid Drawing Algorithm Listing #2

```
;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
; ladot.s,   Written by Mike Fulton

   CSECT  TEXT
   XDEF lineadot,linea_ptr

linea_ptr:
   dc.w $a000
   rts

lineadot:
   dc.w $a001
   rts

   END
```

ATARI Developer News

ATARI.RSC Staff

Notice to ST/TT Developers:

In the best interests of Atari Computer Corporation and our dealer base, the developer section is limiting the hardware orders sold from our program. It is unfortunate that there are non-commercial developers ordering through the developer program and taking necessary business away from our dealerships. We are asking for your cooperation in supporting the Atari team by taking non-commercial orders to your local dealers.

We understand that there are developers in our program to whom hardware is necessary to a product that is currently under development. Special permission to purchase equipment may be obtained by submitting justification information for each hardware item as follows:

1. Reason for requiring new hardware.
2. Description of product under development that will be used with new hardware.
3. Description of programming language or computer systems that support your product.
4. Projected ship date and distributor for your product.

Please mail or fax your justification in to Bill Rehbock or Gail Johnson, along with your company name, contact person, address, and phone number. You will be notified upon approval of your order to send in the purchase total.

The exchange/replacement program for developers is still in effect. If you have malfunctioning merchandise in or out of warranty, please contact Gail Johnson for return information.

Hisoft Signs With Goldleaf

Goldleaf Publishing, of Larkspur, CA, have signed a distribution deal with U.K. software house HISOFT. U.S. programmers will now have much easier access to Hisoft's range of development tools, including Lattice C v5, Lattice C/TT, Devpac, and Devpac/TT.

For more information, contact Goldleaf at (415) 381-7717

Atari Announces Two New Portables at CeBIT

Atari announced two new portable computers at the recent CeBIT electronics show in Hanover, Germany.

continued on page 16

Portfolio Application Notes

J. Patton

Included with the Portfolio developer's newsletter is a copy of the APB. This booklet is a method of getting word out to Portfolio users about your Portfolio products as well as tips on using the Portfolio more effectively. We would like to invite any developers who have commercial applications for the Portfolio to immediately mail or fax sales literature to Don Thomas here at Atari so that we can tell our Portfolio customers about it and so it can be included in the next APB. If you send product for demonstration purposes, our sales people really and truly do use them when presenting the Portfolio.

Please take time to read the following notes, which have been released since the first version of the technical reference manual.

• 3.6.3 Executing a RUN file...

A RUN file can be executed from the Command processor by typing RUN <filename>, or by invoking Int 21h Fn 4Bh at the DOS level as for a normal program, but with AL set to 80h and CL set to CCh.

• APPENDIX C ROM Extensions...

Note: ROM extensions must never leave AL with a value of 0. This is reserved for use by Atari and DIP.

Q & A

Q: How can I disable the internal applications?

A: The internal applications can be disabled by setting the flag (app_inapp) that indicates that an application has been entered. This will prevent the hotkey from allowing another application to be started. This is useful when an application needs to lock out text screens from popping up over graphics screens.

```
app_psp     db 100h-2 dup(?)   ;application's PSP
            dw ?                    ..
app_int16   dw ?     ;original Int16
            dw ?
            dd ?
app_oldpsp  dw ?     ;original PSP on entry
app_newds   dw ?     ;application's data seg
app_inapp   db ?     ;counts invocations,0 or 1
```

To determine the application's PSP use:

```
Int 61h, Fn 22h -- Get Application PSP

Parameters:
    AH      22h
    AL      Subservice (0 - Get Mode, 1 - Set Mode)

  If AL - 1
  DS:00     pointer of structure in DS:00

Returns:
     If AL - 0
     DS:00      pointer to structure in DS:00
```

Q: I have a problem after initially writing a new file. When I make a second attempt to write to the file it produces an error, but it works as expected on a DOS machine.

A: This problem has been identified as the DOS function call Int 21h Fn 42h, or LSEEK. This is used to move the file pointer when a file is extended. The problem occurs when the new file size becomes a multiple of the card's sector size. For the 32K RAM card this is 128 byte sector size, 256 bytes for the 64K card, and 512bytes for the 128K card. One workaround is to produce a maximally sized file to start with and modify the records within it. A second method would be to have your application calculate the new file size so that it is not a multiple of the card's sector size.

Q: The numeric data stored in the PERMDATA.DAT file is very different from standard IEEE. Can you tell me more about it?

A: The structure of the calculator's sign, exponent, and mantissa stored in the PERMDATA.DAT file is as follows:

```
DIP - One byte    Only the 7th bit is used
                  to represent sign of the number.
      Two bytes   Exponent to base 10 stored in
                  two's complement.
      7 bytes     Mantissa.

IEEE- One bit     Represents the sign of the
                  number.
      11 bits     Exponent given with offset of
                  1023 base 10.
      52 bits     Mantissa.
```

New Products

Megabyte Computers has an internal RAM upgrade for the Portfolio which boosts it to 512K. The upgrade with a 6 month warranty is $350.00. They also offer new 512K models for $599.95 Contact Megabyte Computers at (817) 589-2950

BSE in California has released a Portfolio version of their portable hard disk drive, called the Flashdrive. The drive uses the Portfolio's parallel port, a device driver, and BSE's IDE port to connect drives from 20mb to 430mb. The 20mb Flashdrive sells for $499. Contact BSE at (714) 832-4316

New Online for the Portfolio

PFEDIT.ARC - C version of the I60 line editor.

Using the AES Scrap Library

Mike Fulton

The GEM AES Scrap Library provides a standard method of communication between applications for implementing a clipboard to allow data interchange. There are two GEM AES scrap library functions, scrp_read() and scrp_write().

```
WORD sc_wreturn -
scrp_write(sc_wpscrap);
```

The scrp_write() call establishes the directory and filename of the last item to be written to the clipboard folder. A normal sequence of events for an application to put something in the clipboard would be:

1) Do a scrp_read() call to determine the location of the system's clipboard folder. If no existing clipboard folder exists, then your application should use the root directory of the system's boot drive instead.**

2) Create and write your clipboard file, using one of the standard file formats given below, if at all possible,.

3) Do a scrp_write() call with the full pathname of the clipboard file so that other applications will be able to locate it. Clipboard files should be of the form "SCRAP.*" where the filename extension specifies the type of data contained within the file. For example, if your application wrote a raw text file to the clipboard, it would do something like:

```
scrp_write("C:\clip\scrap.txt");
```

If your application wanted to write out a bit image graphic instead of text, then it would do:

```
scrp_write("C:\clip\scrap.img");
```

The following filename extensions are reserved for the following file formats.

**.TXT* – ASCII only text, with a CR/LF at the end of each line

**.ASC* – ASCII only text, with a CR/LF at the end of each paragraph

**.RTF* – ASCII only text with formatting specified through the RICH TEXT FORMAT defined by Microsoft

**.GEM* – Standard GEM Metafile Graphics Image

**.IMG* – Standard GEM Bitmapped Graphics Image

**.DIF* – Data Interchange Format - Spreadsheet/Database data

**.EPS* – Encapsulated Postscript File

**.CVG* – Calamus Vector Graphic

```
WORD sc_rreturn -
scrp_read( sc_rpscrap );
```

The scrp_read() function returns the directory and filename of the last item written to the clipboard. A normal sequence of events for an application to read an item from the clipboard would be:

1) Do a scrp_read() call to get the pathname of the last clipboard file. If scrp_read() returns zero, then the clipboard folder has not been set since the computer was last reset. If the return value is non-zero, then the clipboard directory and filename of the last item placed in the clipboard is returned to you.

2) If scrp_read() returns an error code of zero, then the application should indicate to the user that nothing is available on the clipboard.

3) If the filename returned by scrp_read() ends with ".*" then the clipboard directory has been set, but nothing has been placed in it since the computer was last reset. See item #5 below for more info.

4) If the filename returned by scrp_read() indicates that the current clipboard file uses a file format that your application doesn't understand, then see item #5 below for more info.

5) If the current clipboard file has not been defined, or if it is a file format your application does not support, it is up to the application to optionally search the clipboard directory for a "SCRAP.*xxx*" file it understands. (If the clipboard directory is not defined, an application may optionally check the root directory of the boot drive, as noted earlier.)

(If the application cannot find or rejects the current clipboard file, and then searches for and finds a different clipboard file, it is recommended that it give the user a choice or at least a notification of what's going on.)

**The best way to determine the system's boot drive is to do a GEM AES shel_envrn() call searching for the PATH environment variable, like this:

```
found - shel_envrn( "PATH=" );
```

When you start up your system, GEM AES creates a PATH environment variable pointing to the boot drive. The AES's PATH environment variable should not be affected by commandline shells such as Micro C-Shell, the Mark Williams C Shell, or GULAM.

Please note however, that the PATH environment variable created by the AES is done incorrectly. Instead of "PATH=C:\" followed by a zero byte, it is set up as "PATH=", zero byte, "C:\", zero byte. So if shel_envrn() returns a pointer to a NULL, you should increment the pointer one byte ahead and look again for the actual path information.

Specification for GEMDOS File Sharing & Record Locking

Mike Fulton

The following information describes the methods that will be used when GEMDOS is extended to allow file sharing and record locking. Please note that at this time, despite the fact that this document discusses how certain aspects of these extensions relate to networking, this is not a full-fledged network specification. Developers are encouraged to write software with multiuser capabilities in mind when at all possible.

Detecting The Extensions

An application can determine if the extensions are available by checking the Cookie Jar for the presence of an "_FLK" cookie. See the STE TOS RELEASE NOTES for more information on the Cookie Jar.

_FLK cookie

The cookie "_FLK" indicates the presence of file and record locking extensions to GEMDOS. If this cookie is not found, then the application should assume that file and record locking extensions are not installed and make no GEMDOS calls that rely on them, or pass parameters to GEMDOS functions other than those given in the standard GEMDOS documentation. The value field of the "_FLK" cookie should contain a version number.

If the _FLK cookie is found, then the following extensions and changes to GEMDOS should be observed:

New GEMDOS call:

```
LONG
Flock( handle, mode,
       start, length );

WORD handle;
WORD mode;
LONG start;
LONG length;

GEMDOS Function code = $5C,
(92 decimal)
```

The Flock() function is designed to lock a specified portion of an open file to prevent other processes from accessing and/or modifying that part of the file.

The *handle* parameter is the GEMDOS file handle of the open file.

The *mode* parameter specifies if the portion of the file is being locked or unlocked. Valid values are:

0 = Create a lock, starting at *start* and extending for *length* bytes.

1 = Remove a previously set lock. Parameters *start* and *length* must match a previous lock.

The *start* parameter is the offset from the start of the file, in bytes, where the lock will begin.

The *length* parameter is the length of the locked area, in bytes. Once a lock has been set using a certain handle, other file handles will not be able to read or write the locked area of the file. If there are outstanding locks when a file handle is closed, the behavior is undefined.

If you duplicate a file handle, any existing locks are inherited by the duplicate file handle. If you use Fforce() to redirect a standard handle to a file with locks, the locks are inherited.

Flock() returns an error code in d0.L, see the section below on *GEMDOS Extensions Error Codes* for details.

NOTE: File locks should be used as though they are strictly advisory. Do not rely on getting back error codes from Fread() and/or Fwrite(). A program which is aware of record locking should set a lock on the file immediately before accessing it and clear the lock immediately afterwards.

Changed GEMDOS call:

```
WORD
Fopen( filename, mode );

CHAR *filename;
WORD mode;

GEMDOS Function $3D,
(61 decimal)
```

The Fopen() call is designed to open an existing file and return a file handle that can be used for accessing that file. Additional information relating to file sharing can now be included in the *mode* parameter.

The *filename* parameter is a pointer to a character array containing the filename of the file to be opened. If only the filename is given, without a drive and/or path specification, the file is assumed to reside on the current default drive in the current default path.

The *mode* parameter indicates how the file is to be used. Previously only bits 0-2 were used with this parameter. Bits 0-2 are still used to indicate the file access, which indicates what you want to do with the file. Bits 4-6 are now used to specify the file sharing modes. The sharing mode you use indicates what sort of access you want to allow other processes which may want to use the file.

```
Bit #
7 6 5 4 3 2 1 0
I . . . . . .  Inheritance Flag
. S S S . . .  Sharing Mode
. . . . R . . .  Reserved
. . . . . A A A  Access code
```

Bits 0-2 of *mode* are used for the file access mode. This is the same as the original version of GEMDOS without the extensions. The bits have the following meanings:

```
File Access Modes
Bit #
2 1 0
-----
0 0 0 - Read only access
0 0 1 - Write only access
0 1 0 - Read/Write access
```

Bit 3 of *mode* is reserved and should always be set to zero.

Bits 4-6 of *mode* are the sharing mode bits and define what will happen when more than one program attempts to open the same file. The sharing mode is set by the first process to open a file, and subsequent Fopen() calls to the same file will fail if they attempt to set a conflicting mode. You can open the file again in a more restrictive mode. There are five sharing modes, as shown in the table below.

When a second attempt is made to open a file which is already open, GEMDOS tests the sharing mode the file was originally opened with against the access requested in the second operation, and allows the second open to succeed only if the two modes are compatible. For example, if the file is originally opened with the Deny Writes flag set, then you can open the file again only for reading, not for write or read/write.

An open will fail if its share mode denies something which another handle already has. For example, an open for read access with Deny Writes will fail if the file is already open for Write access.

Compatibility mode is designed to allow existing programs to function, and it is assumed that programs using compatibility mode are not aware of file sharing and the related specialized error messages.

A compatibility mode open call for read access can be mutated into another mode if the file's read-only flag is set. In this case, the mode is mutated into the Deny Writes sharing mode. From that moment on, it isn't a compatibility mode open attempt any more, and you should refer to the rules for opens with Read Access and Deny Writes sharing mode instead.

Otherwise, the access and sharing modes work pretty much as expected. An attempt to open in a "deny" mode will fail if someone already has the file open with the access you want to deny. If it succeeds, then the file cannot be reopened with the access you deny.

If you attempt to open a file in compatibility mode and the mutation described above does not occur, it will succeed only if the file isn't open at all already, or if it is opened by the same process in compatibility mode. Once open in compatibility mode, the file can't be opened by any other process in any mode, and cannot be opened by the same process in anything but compatibility mode.

Bit 7 of *mode* indicates if a child process can inherit the use of this file. When a child process inherits a file handle, it inherits the file's access and sharing codes. If bit 7 = 0, a child process can use the same handle as the parent to access the file. If bit 7 = 1, the child process must open the file itself to obtain a new handle, and existing file sharing restrictions must be observed. However, regions of the file locked by the parent are not accessible to the child.

Standard GEMDOS error messages apply as before, as well as the new error codes described below.

```
File Sharing Modes
Bit 6 5 4 of mode parameter
    -----
    0 0 0 = Compatibility mode
    0 0 1 = Deny Read/Writes, file may not be re-opened
    0 1 0 = Deny Writes, file may be re-opened for read only
    0 1 1 = Deny Reads, file may be re-opened for write only
    1 0 0 = Deny None, file may be re-opened for read/write
```

Table 1, File Sharing Mode Bits

Changed GEMDOS call:

```
WORD
Fcreate( filename, attribs );

CHAR *filename;
WORD attribs;

GEMDOS Function $3C,
(60 decimal)
```

When a file is created using the Fcreate() call, the sharing mode is set as though the file already existed and you opened it for read/write access with Compatibility Mode specified.

If a file already exists and is already open when you try to create it, then you should get an access denied error (without the extensions the behavior is undefined).

Otherwise, Fcreate() functions as before.

GEMDOS Extensions Error Codes:

EOK (0) Operation was successful, no error.

ELOCKED (-58) Record is locked. Returned if requested portion of file is already locked, or if it overlaps a locked portion. This error will also be returned from Fread() and Fwrite() calls when accessing portions of the file which have been locked to different file handles.

ENSLOCK (-59) Matching Lock not found. Returned if the values specified for a lock removal operation do not match a lock set operation for the same handle.

All other standard GEMDOS error messages also apply.

Networking

Since one of the primary reasons for needing file sharing and record locking is to allow multiple nodes on a network to share files, we will discuss a few things about networks.

_NET cookie

The cookie "_NET" indicates the presence of networking software. It does not imply anything about file and record locking. It is the responsibility of the networking software to install the "_NET" cookie and initialize the pointer to the structure. If the "_FLK" cookie is not available, it is the responsibility of the networking software to install the GEMDOS extensions and the "_FLK" cookie as well.

The value field of the cookie is a pointer to a structure of the format shown in table #2. Additional network-specific information should be stored immediately following this structure.

Installing & Checking For The Cookies

An application should only check the cookie that covers the resources it needs. In other words, an application should not be checking the "_NET" cookie if all it is worried about is file locking. Accordingly, a program that sends EMAIL across a network probably only wants to know about the network and won't care about file and record locking.

It is also the responsibility of the networking software to create a cookie jar if no existing one is found. See the STE TOS RELEASE NOTES for more information on the cookie jar.

Comments and Suggestions, Anyone?

I am at this time soliciting comments and suggestions regarding certain aspects of this specification.

First, we're interested in standardizing at least part of the information following the structure pointed to by the "_NET" cookie. What sort of information should go here that an application should know before making calls to the network software? What calls, if any, can an application assume are available if it finds a generic _NET cookie? What calls should be common to all networks? We would like to specify as much of this as possible, as quickly as possible, so that products from different publishers of networking software can store this information in the same place and conform to the standard.

Secondly, as stated before, this specification currently covers only file sharing and record locking. Aspects of what the "_NET" cookie should indicate are still mostly undefined. However, we would also like to solicit opinions and ideas about other aspects of a network implementation. However, please keep in mind that the specification so far conforms very closely to MSDOS, and we would like to stick with that as much as possible.

Finally, there have been a couple of other networking extension outlines distributed by various developers throughout the world. However, compatibility with these protocols is not a primary concern, as these developers are willing to modify their software as needed to conform with an official Atari specification. This is mentioned because at least one of these outlines contains a much wider range of functions than this specification, and some people may feel that this specification gives up a lot of functionality. However, careful examination will reveal that there is a lot of redundancy to the functions detailed in that outline. For example, a separate file locking function is not required, because you can simply do an Fopen() call with the sharing mode set to Deny Read/Write, and the file will be untouchable by any other process. Also, separate functions for locking and unlocking are unnecessary, as the flag in the Flock() function detailed above allow it to both lock and unlock records.

Please address any questions or comments to:

Mike Fulton
Atari Developer Support
1196 Borregas Ave.
Sunnyvale, CA 94088
U.S.A.
(408) 745-8821 - voice
(408) 745-2094 - fax

GEnie = Mike-Fulton
CIS = 75300,1141

```
struct netinfo {
  long  publisher_id;  /* Special code for publisher, */
                       /* to be assigned by ATARI (USA) */
                    /* Usually a four-byte ASCII string */
                                  /* such as "ATRI" */

  long  version;     /* Version number of the network, */
                     /* to be assigned by the publisher */
}
```

Table #2, Network information Structure

Publisher ID's assigned as of May 29, 1991

The following Publisher ID codes have been assigned as of the above date. These are the values contained in the *publisher_id* field of the structure pointed to by the _NET cookie.

Application Design Software = "A&D\0"
Pams Software = "PAMS"
Itos Software = "ITOS"

Table #3, Publisher ID Assignments

Atari Developer News
Continued From Page 10

The first machine is known as the STylus, and is about the size of a magazine. It has a built-in LCD screen, but no keyboard. Instead of a keyboard, the machine comes with a special stylus which the user uses to write text directly onto the screen. Special software built into the machine's ROMs will transparently convert the user's writing into characters and pass them along to the application, just like they had been typed on a regular keyboard, and includes the ability to adapt to the users writing. The pen also acts to move the mouse pointer on screen, and will work with current applications like Degas and Easy-Draw.

If a keyboard is required, the machine will accept any Mega, MegaSTE, or TT keyboard through its built-in keyboard connector.

The STylus will be available with either 1 or 4 megabytes of memory, but the 1 megabyte version will not be upgradable due to the way the machines are designed for power considerations. Battery life is expected to be 10-20 hours.

A special expansion connector will allow the user to hook up floppy and hard disk drives. The machine will use small plug-in JAIDA RAM or ROM cards for STylus-specific applications. These are similar in appearance to the cards used by the Portfolio, but they are not compatible.

The STylus is aimed more at a VARs and vertical applications than the existing ST/TT market.

The second machine is the ST Book. This is a small notebook-sized computer which weighs about 4-1/2 pounds. It uses a built-in, non-backlit LCD screen. It will include a built-in IDE hard disk interface and drive with a transfer rate similar to that of ACSI devices. A built-in touch pad serves to operate the mouse pointer, and there is a connector for an external numeric keypad with external mouse and joystick ports.

The ST Book uses the same Nicad battery pack as the STylus, and battery life is expected to be 5-10 hours. However, if the batteries go dead, your data will be safe in the system's RAM for 50-60 hours.

A special SAVE AND RESUME function will save the system's entire state and shut the power off. Later when you turn it back on, you'll be right where you left off, even in the middle of a program. Closing the machine's lid will do this automatically.

There is no built-in floppy disk drive, but an external expansion connector will allow the user to hook up external floppy disk drives and/or ACSI hard disk drives.

Additional information on these machines will be forth coming in the next issue of ATARI.RSC. No information about availability of machines or documentation for developers is available yet, but such information will be announced in ATARI.RSC when it is available.

Atari Corporation
1196 Borregas Ave.
Sunnyvale, CA 94088