Introduction

The following pages are a work in progress. The Atari™ Compendium (working title) is designed to be a comprehensive reference manual for Atari software and hardware designers of all levels of expertise. At the very least, it will (hopefully) be the first book available that documents all operating system functions, including any modifications or bugs that were associated with them, from TOS 1.00 to whatever the final release version of Falcon TOS ends up being. GEMDOS, BIOS, XBIOS (including sound and DSP calls), VDI, GDOS, LINE-A, FSM, AES, MetaDOS, AHDI and MiNT will be documented. Hardware information to the extent that information is useful to a software programmer will also be covered. This volume will not include hardware specifications used in the creation of hardware add-ons, a programming introduction designed for beginners, or an application style guide. All of the aforementioned exclusions will be created separately as demand for them arise. In addition, I also plan to market a comprehensive spiral-bound mini-reference book to complement this volume.

By providing early copies of the text of this volume I hope to accomplish several goals:

1. Present a complete, error-free, professionally written and typeset document of reference.
2. Encourage compatible and endorsed programming practices.
3. Clear up any misunderstandings or erroneous information I may have regarding the information contained within.
4. Avoid any legal problems stemming from non-disclosure or copyright questions.

A comprehensive Bibliography will be a part of this volume. For now you should know that I have mainly relied on five major sources of information:

1. Atari Developer Documentation, including, but not limited to, original OS docs, release notes, newsletters, and technical support.
2. Compute’s AES/TOS/VDI series. This series seems to be the most complete English reference available, however, its usage is limited by the fact it is only current as of TOS 1.02
3. Lattice C Atari Library Manual and Addendum
4. Atari Profibuch - Excellent German text.
5. Developer Roundtable on GEnie and Compuserve.

How to Edit...
Below are some simple suggestions as to how to notate any changes you would like to see made. I understand you are probably just as busy as I usually am so if you can’t take the time to follow these steps, ragged handwriting in the corner would be just as appreciated.

Included in your package should be seven items:

1. This introduction letter.
2. A binder.
3. Revision notes
4. Looseleaf notebook paper.
5. Two highlighter pens
6. Dividers
7. The latest revision of the text

If you are missing any items, please contact me.

Each revision will be accompanied by a set of revision notes. These will highlight what to look for, what I already know is wrong and am planning to change, and what has changed since last time.

The looseleaf notebook paper should be used to make general suggestions as to content, style (writing/typesetting), and so on....

When proofing the text use the blue highlighter to circle spelling, grammar, or style errors (any typo). The green highlighter is for blatant errors or misunderstandings where an explanation is necessary. Please notate the error and correction in the margins. If it is a very large misunderstanding beyond simply writing it down, please call me or E-Mail me.

Also, as a part of the volume will be a listing of standard conventions. The following is a brief listing of conventions used in the book:

<table>
<thead>
<tr>
<th>Typestyle</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>The quick brown fox......</td>
<td>Normal Text</td>
</tr>
<tr>
<td>WORD appl_init(VOID)</td>
<td>Function Definitions</td>
</tr>
<tr>
<td>mode, flag, ap_id</td>
<td>program/system variables</td>
</tr>
<tr>
<td>WORD, TOS, WM_CLOSED</td>
<td>macros, typedef’s, define’s, OS components</td>
</tr>
<tr>
<td>typedef struct {</td>
<td>Program listings/bindings</td>
</tr>
<tr>
<td>A basic explanation is listed...</td>
<td>Text in tables</td>
</tr>
<tr>
<td>CTRL-G</td>
<td>Keyboard keys</td>
</tr>
<tr>
<td>OPCODE</td>
<td>Headings</td>
</tr>
</tbody>
</table>

Any questionable stray from the conventions should be notated as a possible error.

**Revision Schedule**

I would like to swap edited text with new revisions about every two weeks. The final revision should be approved by November 15th to try for a release date of December 15th. This schedule is not fixed and I will be in contact to find out what’s best for you.

**Thank You...**

Thank you for your time and effort. Your name will be credited if you desire and you should check for it in a final revision.
SCOTT D. SANDERS, OWNER
SOFTWARE DEVELOPMENT SYSTEMS
CONTENTS

Foreword vii

Chapter 1: Introduction to Atari Programming .......... 1.1
Atari Computer Hardware ................................................................. 1.3
Atari Computer Software ................................................................. 1.6
Atari GEM ......................................................................................... 1.7
Third-Party System Software.......................................................... 1.8
Programming Languages ................................................................. 1.9
Conventions ..................................................................................... 1.10

Chapter 2: GEMDOS ................................................................. 2.1
Overview ............................................................................................ 2.3
The TOS File System ......................................................................... 2.3
Memory Management ......................................................................... 2.8
GEMDOS Processes ........................................................................... 2.9
GEMDOS Vectors .............................................................................. 2.13
MiNT .................................................................................................... 2.14
MiNT Interprocess Communication ................................................... 2.27
MiNT Debugging .............................................................................. 2.31
The MINT.CNF File .......................................................................... 2.33
GEMDOS Character Functions .......................................................... 2.34
GEMDOS Time & Date Functions ....................................................... 2.35
GEMDOS Function Calling Procedure ............................................. 2.35
GEMDOS Function Reference ........................................................... 2.37

Chapter 3: BIOS ................................................................. 3.1
Overview ............................................................................................ 3.3
System Startup .................................................................................. 3.3
OS-Header ......................................................................................... 3.4
Cookie Jar ......................................................................................... 3.8
BIOS Devices ..................................................................................... 3.14
Media Change ................................................................................... 3.15
BIOS Vectors .................................................................................... 3.18
Chapter 4: XBIOS

Chapter 5: Hardware

Chapter 6: AES
Chapter 7: VDI ..............................................................7.1

Overview ........................................................................................................ 7.3
VDI Workstations........................................................................................... 7.3
Workstation Specifics .................................................................................... 7.5
Using Color ........................................................................................................ 7.8
VDI Raster Forms .......................................................................................... 7.9
Vector Handling ............................................................................................. 7.10
GDOS ........................................................................................................... 7.11
GDOS 1.x ..................................................................................................... 7.12
FONTGDOS ................................................................................................ 7.13
FSM-GDOS ................................................................................................. 7.13
SpeedoGDOS ............................................................................................... 7.14
Device Drivers ............................................................................................. 7.16
VDI Function Calling Procedure................................................................ 7.18
VDI/GDOS Function Reference .................................................................. 7.21

Chapter 8: Line-A .........................................................................................8.1

Overview ........................................................................................................ 8.3
The Line-A Variable Table ............................................................................ 8.3
Line-A Font Headers ..................................................................................... 8.7
Line-A Function Calling Procedure ............................................................. 8.8
Line-A Function Reference .......................................................................... 8.9

Chapter 9: The Desktop ..............................................................................9.1

Overview ........................................................................................................ 9.3
MultiTOS Considerations ............................................................................... 9.3
Desktop Files ................................................................................................. 9.4
Chapter 10: XCONTROL .........................................................10.1
  Overview .......................................................................................................10.3
  XCONTROL Structures ..................................................................................10.4
  CPX Flavors ....................................................................................................10.6
  CPX File Formats ............................................................................................10.12
  XCONTROL Function Calling Procedure .....................................................10.13
  XCONTROL Function Reference .................................................................10.15

Chapter 11: GEM User Interface Guidelines ..........11.1
  Overview .......................................................................................................11.3
  The Basics .....................................................................................................11.3
  Windows .........................................................................................................11.4
  Dialog Boxes ..................................................................................................11.8
  Alerts ...............................................................................................................11.10
  The File Selector ............................................................................................11.12
  Progress Indicators .........................................................................................11.12
  Toolboxes ........................................................................................................11.13
  Toolbars .........................................................................................................11.14
  Menus ..............................................................................................................11.15
  Keyboard Equivalents .....................................................................................11.20
  Device Independence ......................................................................................11.22
  Globalization ..................................................................................................11.23
  Colors ..............................................................................................................11.23
  Sound ...............................................................................................................11.24
  Application Software .....................................................................................11.24
  Installation Software ......................................................................................11.25
  Entertainment Software ..................................................................................11.25

Appendix A: Functions by Opcode ................. A.1
  GEMDOS Functions by Opcode ....................................................................... A.3
  BIOS Functions by Opcode ............................................................................. A.7
  XBIOS Functions by Opcode ........................................................................... A.9
  AES Functions by Opcode .............................................................................. A.13
  VDI Functions by Opcode .............................................................................. A.15

Appendix B: Memory Map ...................................... B.1
  Usage ............................................................................................................. B.3
  Memory Map .................................................................................................. B.4
Appendix C: Native File Formats ................................. C.1
  The .GEM File Format ........................................................................................................C.3
  The .IMG File Format ........................................................................................................C.5
  The .FNT File Format .........................................................................................................C.7
  The .RSC File Format ........................................................................................................C.9

Appendix D: Error Codes........................................... D.1

Appendix E: Atari ASCII Table ................................. E.1

Appendix F: IKBD Scan Codes ................................. F.1

Appendix G: Speedo Fonts .......................................... G.1
  The Speedo Font Header ..................................................................................................G.3
  The Bitstream International Character Set ........................................................................G.7

Appendix H: The Drag & Drop Protocol .............. H.1
  Overview ..........................................................................................................................H.3
  The Originator ...............................................................................................................H.3
  The Recipient .................................................................................................................H.5

Appendix I: The Programmable Sound Generator .. H.1

Bibliography

Index
FOREWORD

About eight months have passed since The Atari Compendium® was first released, and I must admit to being amazed with the amount of attention the book has received from Atari developers worldwide. When I started writing the first draft of the book I didn’t know enough about Atari computers to write half of the 860 pages it eventually became. The learning process that I went through to see the book to its completion was responsible for a great deal of personal growth and a greater understanding of computer science in general.

It was inevitable, of course, that there would be errors in a book this big. I didn’t want to revise the book simply to correct those errors, however. I was determined to add some missing topics as well. This first revision now adds about 60 pages to the original and led me back to the on-the-job learning process and several phone calls and E-mail letters to Sunnyvale.

The Compendium now covers almost every conceivable topic a software programmer needs to know about Atari computers. You still won’t find timing diagrams, pinouts, and hardware specifications simply because my level of competence in those matters is unfortunately minor. The only other topics you won’t find discussed are those covered completely in separate volumes (referenced in the Bibliography). These include hardware-direct ACSI/SCSI/IDE programming, SCC programming, DSP programming, and direct BLiTTER chip usage. In every case except for DSP programming, almost all functions of these devices may be accessed by the average programmer through the use of OS calls, which are, of course, documented. The basics of DSP programming, like assembly or ‘C’ is left to the reader to explore in other books dedicated to their teaching.

New to this revision you will find an enhanced style guide and memory map (the two most popular sections of the book, it seems), information on programming MiNT device drivers and file systems, and a section documenting the XBRA protocol. Most importantly, though, almost every conceivable parameter and return value has been listed with a corresponding definition name. These names may be used with any language that supports constant naming, and, when used, improve program readability dramatically. The TOS.H and TOSDEFS.H include files will be available from SDS upon the release of this revision. To find out how to obtain them, be sure to send in your registration card.

I owe thanks to Mike Fulton, Eric Smith, and Jay Patton were very helpful in ensuring that the new material was correct and old errors were eliminated. Many independent readers of the book also deserve thanks for taking the time to report errors and submit their comments.

In addition, my close friends Dennis, Mike, Keith, Cathryn, Shawn, Cathy, Shaun, and Kristýna provided moral support and dragged me away from work when I needed a break badly. Also, as always, my mom supported me tremendously and continues to proudly display a plastic-wrap’d copy of the book to friends and relatives even though to her its about as useful as a phone book for some remote city in Alaska.
Thanks to you, especially, the Atari developers and users who made this book a reality. Enjoy!

—Scott D. Sanders, April 1994
Atari Computer Hardware

The 260/520/1040 ST
The first Atari ST computers became available to the public in 1985. The new Atari models were the first 16-bit computers well-suited for use in the home. The availability of these computers signaled the end of the Atari 8-bit era of computers such as the 400, 800, 800XL, 1200XL, 1450XLD, 65XE, and 130XE computers.

The name ‘ST’ is derived from the capabilities of the Motorola 68000 processor upon which the original Atari line was based. The 68000 uses a Sixteen-bit data bus with a Thirty-two bit address bus.

16-bit computers introduced a new concept in computer technology called the operating system (OS). Atari’s operating system, The Operating System (TOS), was loaded from a boot disk originally, but is now almost always installed in ROM.

A primary subsystem of TOS is GEM (‘Graphics Environment Manager’), the graphical user interface used by Atari computers. GEM, which was developed by Digital Research, Inc., manages the graphic interface to applications and provides access to popular computing features with buzzwords such as windows, the mouse, menus, and the desktop.

GEM was originally designed for PC-compatible computers. PC-based GEM, however, is no longer completely compatible with Atari GEM. Only components of GEM relative to its use on the Atari will be covered in this guide. Some functions which were originally documented for Atari GEM yet never implemented have been included for completeness.

Other TOS subsystems include GEMDOS, the BIOS, and the XBIOS. These subsystems provide a hardware interface and management functions for the file system.

The original ST computers featured the following:

- Motorola 68000 32-bit processor running at 8MHz.
- Integrated GEM/TOS operating system.
- RAM memory storage of 256k, 512k, or 1 Mbyte (depending on model).
- Built-in MIDI, dual joystick, floppy drive, ACSI, serial, and parallel ports.
- Sophisticated DMA peripheral access.
- Yamaha 3-voice FM sound generator.
- External 128k cartridge port.
- Integrated video controller capable of generating (320x200x16), (640x200x4), and (640x400x2) video modes from as many as 512 colors.
Mega ST 2/4

Two years after the release of the original ST series Atari released the Mega ST series of computers. The Mega ST computers were shipped with TOS 1.02 and featured several new features as follows:

- BLiTTER chip (for faster graphics).
- Internal expansion bus.
- Separate keyboard and CPU.
- Either two or four megabytes of RAM.
- Peripheral co-processor slot (for 68881 coprocessor, etc.).

STacy

The STacy was released shortly after the Mega ST to provide a portable means of Atari computing. STacy computers were shipped with TOS 1.04. The STacy’s design supplemented the basic features of an ST with the following:

- Integrated CPU/keyboard/carrying case.
- Monochrome LCD screen.
- Track ball for mouse control.
- Optional hard drive.

1040 STe

The 1040 STe, released in 1990, was designed to expand upon the capabilities of the 1040 ST. Many of the features added were geared towards entertainment and multimedia applications. The 1040 STe was shipped originally with TOS 1.06. The following features were added to those of the basic ST:

- Extended color palette to support up to 4096 colors.
- Support for horizontal and vertical fine scrolling.
- Video GENLOCK capability.
- Stereo 8-bit PCM sound.
- Two extra joystick ports with support for paddles and light pens.
- 256k Operating System in ROM.
- SIMM memory slots to upgrade memory to 4 Mb

Mega STe

Released in 1990, the Mega STe was designed to provide for more computing power than the 1040 STe and add several new hardware features. The Mega STe shipped with TOS 2.02, 2.05, or 2.06. It adds features to that of a 1040 STe as follows:

- Motorola 680000 32-bit processor running at 8MHz or 16MHz.

THE ATARI COMPENDIUM
• Optional 68881 math coprocessor.
• One, two, or four megabytes of RAM memory.
• Optional internal hard drive.
• Modern case design with separate keyboard/CPU.
• Three serial ports.
• Localtalk compatible networking port.
• VME compatible expansion slot.

**TT030**

Also released in 1990, the TT030 computer was the first Atari computer workstation designed for high-end computer users. The TT030 workstation was shipped with TOS 3.01, 3.05, or 3.06. It adds the following features to that of the Mega STe:

• Motorola 68030 32-bit processor running at 32MHz with cache.
• Memory capacity of 32Mb with optional ‘fast’ RAM.
• Standard 68882 math coprocessor.
• Four serial ports.
• SCSI device port.
• Stereo RCA jacks for sound output.
• Extra video resolutions of (320x480x256), (640x480x16), and (1280x960x2).

**ST Book**

Designed to replace the STacy as the defacto portable ST computer, the ST Book brought the basic computing power of an ST to a lightweight notebook computer. This machine was only released in Europe and Atari only shipped a very small quantity. The ST Book was shipped with TOS 2.06. Minus the internal floppy drive, it supported features beyond that of a STacy as follows:

• Lightweight case design.
• Keyboard with integrated numeric keypad.
• Mouse ‘vector’ pad.
• Processor-direct expansion slot.
• External keypad port.
• Floppy drive connector.

**Falcon030**

The newest member of the Atari line, the Falcon030 is to become the new base model Atari system. The Falcon030 is currently shipping with TOS 4.04. While remaining backwardly-compatible, the Falcon030 adds many new features as follows:
1.6 – Introduction to Atari Programming

- Integrated case and keyboard design.
- Motorola 68030 processor running at 16MHz with cache.
- Motorola 56001 DSP with 96k RAM.
- Standard configurations with 1, 4, or 14Mb RAM.
- Internal 2 ½” IDE hard drive optional.
- Video resolutions from 320x200 to 640x480 with a palette from 2 to 256 colors and 16-bit true color.
- Adaptable to Atari monitors, standard VGA monitors, and composite video.
- GENLOCK-ready design.
- Ports include parallel, serial, external floppy, SCSI-2, LAN, 4 joystick, MIDI in/out, microphone, headphone, and ST compatible cartridge port.
- Interior processor expansion port.
- Sound system includes standard Yamaha FM chip, connection matrix, and 8-track, 16-bit stereo record/playback.

Atari ‘Clone’ Computers

Atari ‘clone’ computers first became available in early 1994. These computers, while mostly software compatible with Atari-produced computers, contain hardware enhancements and modifications that may cause incompatibilities in software that relies on hardware access rather than the recommended method of using standardized OS calls.

The recent availability of these computers as well as enhanced graphics and peripheral boards emphasizes the value of programming using the OS whenever possible to allow software to be run on the widest variety of machine configurations.

Atari Computer Software

GEMDOS

GEMDOS consists of file system management routines that provide access to all of the basic devices supported by Atari computers. It bears resemblance to MS-DOS in its functions and opcode numbering while still maintaining some differences and advantages.

MultiTOS

MultiTOS is the first truly multi-tasking extension to GEMDOS supported by Atari. Based on MiNT, developed by Eric Smith, MultiTOS adds true pre-emptive multitasking, memory protection, and process control. Its methods of job control and interprocess communication will be familiar to UNIX users. With the ability to support loadable device drivers and file systems, MultiTOS provides a complete range of functions to complement GEMDOS. In its current incarnation, MultiTOS is an option and thus disk-based as opposed to burned in ROM.
BIOS

The ST BIOS (‘Basic Input/Output System’) comprises the lowest-level of device communication. GEMDOS uses the BIOS to accomplish many of its file system operations.

XBIOS

The XBIOS (‘eXtended Basic Input/Output System’) controls other hardware-specific features such as the floppy drive, video controller, DSP, MFP, and sound system.

Atari GEM

AES

The AES is responsible for window and menu control, messaging services, and object rendering and manipulation.

VDI

The VDI consists of a series of drivers which provide device-independent access to the display screen and external output devices such as printers and plotters through GDOS. All graphic primitive operations are accomplished with the VDI. The AES, for instance, uses the VDI to render its objects on screen.

GDOS

GDOS is a disk-loadable subsystem of the VDI. The term GDOS can refer to original GDOS, FONTGDOS, or SpeedoGDOS. It controls loadable device drivers and fonts. The original GDOS was limited to bitmap fonts and did not have the bezier capabilities of FONTGDOS or SpeedoGDOS.

FONTGDOS

FONTGDOS is essentially a newer, faster GDOS with bezier rendering functions present. FONTGDOS is otherwise completely backwardly compatible with GDOS.

SpeedoGDOS

SpeedoGDOS, named for the Speedo™ font format created by Bitstream, Inc., adds outline font rendering capability to the basic features of GDOS. SpeedoGDOS also includes a sophisticated caching system to promote the fastest rendering possible.

Two versions of outline GDOS exist. The original version (referred to as Font Scaling Module (FSMGDOS)), based on QMS/Imagen fonts, was never officially released. Nonetheless, a small number of users still use FSMGDOS and differences between them are noted.

LINE-A

LINE-A is a special set of routines that provide an assembly language interface to routines and variables belonging to the VDI and XBIOS. It is so named because instruction opcodes beginning with the hexadecimal number $A utilize a special microprocessor exception which point to the LINE-A routines in ROM.
LINE-A is the only operating system component that has become out of date and incompatible. Atari recommends that software developers avoid using LINE-A as it will be supported less and less as hardware advancements make its use more incompatible. LINE-A is documented briefly in this reference for completeness.

**Desktop**

The ‘Desktop’ is a independent GEM application burned into ROM. It facilitates program launching and file manipulation as well as providing a graphical shell for user-interaction.

**XCONTROL**

XCONTROL (Extensible Control Panel) is a desk accessory application that provides access to multiple modules called CPX’s (Control Panel Extensions) which are used to control system configuration and other related functions. A special section in this reference discusses the creation of CPX’s and the utility functions provided by the XCONTROL shell accessory.

**Third-Party System Software**

**Geneva**

Geneva is an alternative, TOS-compatible operating system developed by Gribnif Software. It functions mostly as an AES replacement although it supplements other areas of the OS to provide cooperative multitasking (as opposed to MultiTOS’s pre-emptive multitasking).

Programming for Geneva 1.0 is identical to programming for GEM with AES version 4.0. Geneva does not currently support MiNT extensions though Gribnif has announced plans to eliminate this incompatibility in a future version. You can detect Geneva by searching for the cookie ‘Gnva’ in the system cookie jar. Likewise, the presence of MiNT extensions can be determined by the ‘MiNT’ cookie.

Programmers should not rely specifically on the presence of these cookies to determine if the current OS variety supports multitasking. The AES global array contains values to help determine the possible number of concurrent processes and the AES version number. In addition, the AES call appl_sysinfo(), available as of AES 4.0, can be used to determine the presence of special AES features.

Geneva offers several system extensions not available under MultiTOS. Information on programming the Geneva OS is available in the commercial package and direct from Gribnif Software.

**Programming Languages**

‘C’

‘C’ has become the default standard for Atari computer programming. Most reference books and materials illustrate OS functions using ‘C’ style bindings. This book is oriented towards ‘C’ without, hopefully, alienating developers who develop in other languages. Several different ‘C’
compilers exist in the Atari domain. All have their various features and quirks which make it necessary to be familiar enough with your implementation to modify the source contained in this reference appropriately.

All ‘C’ bindings in this book were created for use with Lattice ‘C’ by Hisoft, Inc.. They should be easily convertable to other major Atari ‘C’ compilers.

Luckily, most ‘C’ compilers agree with their function naming and in most cases you can simply call the function as listed. If you have an older version compiler you may need to add some bindings using the information provided in accordance with your compiler’s recommendations.

Assembly Language
For the convenience of assembly language programmers, all functions are listed with their opcode and related binding. In addition, a section provided in front of the function reference will explain the calling conventions for functions in that category.

All assembly listings in this book were created for use by the AS68 compiler included in the Atari developer’s kit.

BASIC
Depending on the type of BASIC you utilize, functions may be named identically or differently from what is listed in this book. It is recommended that you seek a BASIC compiler that gives you proper access to all of the functions of the machine or familiarize yourself with a more robust language.

Other Languages
Various other languages exist in the Atari domain. Pascal, Forth, ‘C++’, and others have implementations that are similar in design to ‘C’. You should refer to your language manual to properly utilize information found in this reference.

Conventions

Typesetting
The following table displays a list of typesetting conventions used in this book:

<table>
<thead>
<tr>
<th>Style</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>Normal Text</td>
<td>Standard body text.</td>
</tr>
<tr>
<td><strong>BOLD TEXT</strong></td>
<td>Bolded words include function names like <code>appl_init()</code> , ‘C’ macros,</td>
</tr>
<tr>
<td></td>
<td>‘#defined’ data types like <code>WORD</code>, and operating system components</td>
</tr>
<tr>
<td></td>
<td>such as <code>GEM</code> and <code>TOS</code>.</td>
</tr>
<tr>
<td><em>Italicized Text</em></td>
<td>Italicized text is used to represent</td>
</tr>
</tbody>
</table>
variable names like handle. In addition sections of this book like AES Reference Manual will be in capitalized italic text.

<table>
<thead>
<tr>
<th>Text between vertical bars</th>
<th>Vertical bars imply the absolute value of the variable or expression within. For instance:</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>-2</td>
</tr>
</tbody>
</table>

( Number 1, Number 2 ) Two numbers contained within parentheses and separated by a comma indicate a coordinate point X followed by Y. For instance, ( 100, 100 ).

Number1 ^ Number 2 2 ^ 8 is the same as 2^8 or 2 to the power of 8.

Fixed Width Text This style of text is used to present bindings and computer listings.

Table Text This smaller style of text is used in tables as body text.

Functions
The function references in this guide are designed in a compatible manner for ease of reading. Each function is illustrated as follows (headings not applicable for a particular function will be omitted):

objc_draw()

WORD objc_draw( tree, obj, depth, bx, by, bw, bh )
OBJECT *tree;
WORD obj, depth, bx, by, bw, bh;

Immediately following the definition, a brief summary of the function will follow.

Opcode The opcode related to the function will be listed in decimal and hexadecimal where appropriate.

Availability This section will indicate any special conditions that must exist for this function to be present (i.e.: OS version, presence of GDOS, etc.).

Parameters The meaning of each parameter to the function will be explained here. If any data
pointed to by parameters is modified it will be noted here as well.

**BINDING**

This section will list a binding for the function in either ‘C’ format or assembly format, whichever is more appropriate. Please note bindings were written with ease of reading, not necessarily optimized code, in mind.

**RETURN VALUE**

This section explains the return value of the function. This covers only that value returned on the left side of the function expression.

**VERSION NOTES**

Under this heading, any features of a function which are only present under certain conditions are discussed.

**CAVEATS**

Known bugs or abnormalities of a function are listed next to this heading.

**COMMENTS**

Other useful information or hints are listed here.

**SEE ALSO**

Functions which bear a relation to the current function or which are codependent on one another are listed here.

---

**Data Types**

Within function definitions, several data types are referenced that vary from compiler to compiler. The following provides a key to the data type used and their actual definition. Other data types will contain a structure definition or ‘typedef’ within the binding. Be aware that some compilers default to 16-bit integers while others use 32-bit integers.

<table>
<thead>
<tr>
<th>Usage</th>
<th>Synonyms</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>WORD</td>
<td>short, int, short int</td>
<td>16-bit signed integer</td>
</tr>
<tr>
<td>UWORLD</td>
<td>unsigned int, unsigned short, unsigned short int</td>
<td>16-bit unsigned integer</td>
</tr>
<tr>
<td>LONG</td>
<td>long, int, long int</td>
<td>32-bit signed integer</td>
</tr>
<tr>
<td>ULONG</td>
<td>unsigned long, unsigned int, unsigned long int</td>
<td>32-bit unsigned integer</td>
</tr>
<tr>
<td>VOID</td>
<td>void</td>
<td>This naming is used to denote a function with no parameters or return value.</td>
</tr>
<tr>
<td>BOOLEAN</td>
<td>bool, boolean, short, short int, int</td>
<td>16-bit signed integer valid only as TRUE (non-zero) or FALSE (0)</td>
</tr>
<tr>
<td>WORD *</td>
<td>short *, int *, short int *</td>
<td>This is a pointer to a 16-bit signed integer.</td>
</tr>
<tr>
<td>UWORD *</td>
<td>unsigned short *, unsigned int *, unsigned short int *</td>
<td>This is a pointer to a 16-bit unsigned integer.</td>
</tr>
<tr>
<td>LONG *</td>
<td>long *, int *, long int *</td>
<td>This is a pointer to a 32-bit signed integer.</td>
</tr>
<tr>
<td>ULONG *</td>
<td>unsigned long *, unsigned int *, unsigned long int *</td>
<td>This is a pointer to a 32-bit unsigned integer.</td>
</tr>
<tr>
<td>VOIDP</td>
<td>void *, char *</td>
<td>This represents a pointer to an</td>
</tr>
</tbody>
</table>
1.12 – Introduction to Atari Programming

Numeric Values

Because different computer languages use different nomenclature to specify numbers in different bases, you will come across numbers presented in a variety of different ways within this book as follows:

<table>
<thead>
<tr>
<th>Prefix</th>
<th>Decimal 23 as an Example</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>None</td>
<td>22</td>
<td>This number is shown in decimal (base 10) format. The majority of numbers shown will be in this format for simplicity.</td>
</tr>
<tr>
<td>0x</td>
<td>0x16</td>
<td>This number is shown in hexadecimal (base 16) format. Function opcodes in assembly language and numbers used as mask values will appear mostly in this format.</td>
</tr>
<tr>
<td>$</td>
<td>$16</td>
<td>Same as above.</td>
</tr>
<tr>
<td>0</td>
<td>026</td>
<td>This number is shown in octal (base 8) format. Only in extremely specialized cases will numbers be represented in this manner.</td>
</tr>
<tr>
<td>%</td>
<td>%00010110</td>
<td>This number is shown in binary (base 2) format. Only when dealing with hardware registers and in a few other circumstances will numbers be represented in this manner.</td>
</tr>
</tbody>
</table>

Constant Definitions

Modern programming practices dictate the use of named constants wherever possible in place of ‘raw’ values. Take for example the following call to Devconnect():

In ‘C’:

```
Devconnect( 3, 9, 0, 0, 1 );
```

In assembly language:

```
move.w #1,-(sp)
move.w #0,-(sp)
move.w #0,-(sp)
move.w #9,-(sp)
move.w #3,-(sp)
move.w #$8B,-(sp)
```
Calling the function in this format makes debugging and program maintenance more difficult because the parameters’ meanings are concealed by the numeric assignments. The following code illustrates the preferred method of coding:

**In ‘C’:**

```c
/* Extracted from TOSDEFS.H, included by TOS.H */
#define ADC 3
#define DMAREC 0x01
#define DAC 0x08
#define CLK_25M 0
#define CLK_COMPAT 0
#define NO_SHAKE 1

/* Program segment */
#include <TOS.H>
Devconnect( ADC, DMAREC|DAC, CLK_25M, CLK_COMPAT, NO_SHAKE );
```

**In assembly language:**

```assembly
; Extracted from TOSDEFS.I
ADC EQU 3
DMAREC EQU $01
DAC EQU $08
CLK_25M EQU 0
CLK_COMPAT EQU 0
NO_SHAKE EQU 1

Devconnect EQU $8B

; Program Segment
INCLUDE "TOSDEFS.I"
move.w #NO_SHAKE, -(sp)
movw. w #CLK_COMPAT, -(sp)
movw. w #CLK_25M, -(sp)
movw. w #DMAREC!DAC, -(sp)
movw. w #ADC, -(sp)
movw. w #Devconnect, -(sp)
trap #14
lea 12(sp),sp
```

Unfortunately, because many function call parameters do not have standard definitions associated with them, programmers have had to create their own, which in turn makes their programs less portable, or use the ‘raw’ constants. In addition, some compilers do not use standardized definitions at all.

To help alleviate these difficulties, this revision of the Compendium contains named definitions for almost every possible function parameter. These definitions come from the ‘C’ header files TOS.H and TOSDEFS.H or the assembly include file TOSDEFS.I, both available on disk from
SDS. Every attempt has been made to ensure that these files compile with development tools in the Lattice ‘C’, Pure ‘C’, and Alcyon ‘C’ packages. Some modifications to these files may be necessary, however, due to the peculiarities of some compilers.

The ‘C’ header files consist of two parts to improve portability between compiles. The TOS.H file is a compiler dependent file used to bind the operating system calls to definitions. This file, in turn, includes the file TOSDEFS.H which should remain portable between compilers.

When choosing definitions for inclusion in the TOSDEFS files, names given by Atari were given highest precedence followed by those assigned (and kept consistent) by compiler manufacturers. Other definitions were created with simplicity and consistency in mind.

Use of the given constants will increase program code readability and provide for a higher level of portability between compilers.
Overview

GEMDOS contains functions which comprise the highest level of TOS. In many cases, GEMDOS devolves into BIOS calls which handle lower level device access. GEMDOS is responsible for file, device, process, and high-level input/output management. The current revision number of GEMDOS is obtained by calling $\text{Sversion}()$. You should note that the GEMDOS version number is independent of the TOS version number and you should not count on any particular version of GEMDOS being present based on the TOS version present.

Much of GEMDOS closely resembles its CPM 68k and MS-DOS heritage. In fact, the file system and function calls are mostly compatible with MS-DOS. MS-DOS format floppy disks are readable by an Atari computer and vice-versa.

For the creation of MultiTOS, GEMDOS was merged with the MiNT operating environment which derives many of its calls from the UNIX operating system.

The TOS File System

GEMDOS is responsible for interaction between applications and file-based devices. Floppy and hard disk drives as well as CD-ROM, WORM, and Magneto-Optical drives are all accessed using GEMDOS calls.

Prior to the advent of MultiTOS, Atari programmers were limited to the TOS file system for file storage and manipulation. With the introduction of MultiTOS, it is now possible for developers to create custom file systems so that almost any conceivable disk format becomes accessible.

As a default, MultiTOS will manage files between the TOS file system and alternative file systems to maintain backward compatibility. Applications which wish to support extra file system features may do so. The $\text{Pdomain}()$ call may be used to instruct MultiTOS to stop performing translations on filenames, etc. Other calls such as $\text{Dpathconf}()$ can be used to determine the requirements of a particular file system.

The explanation of the file system contained herein will limit itself to the TOS file system.

Drive Identifiers

Each drive connected to an Atari system is given a unique alphabetic identifier which is used to identify it. Drive ‘A’ is reserved for the first available floppy disk drive (usually internal) and drive ‘B’ for the second floppy disk drive. If only one floppy drive exists, two letters will still be reserved and GEMDOS will treat drive ‘B’ as a pseudo-drive and request disk swaps as necessary. This feature is automatically handled by GEMDOS and is transparent to the application.
Drives ‘C’ through ‘P’ are available for use by hard disk drives. One letter is assigned per hard drive partition so a multiple-partition drive will be assigned multiple letters. MultiTOS extends drive letter assignments to ‘Z’ drive. Drive ‘U’ is a special drive reserved for MultiTOS and is unavailable for assignment.

The amount of free storage space remaining on a drive along with a drive’s basic configuration can be determined using the `Dfree()` call.

**GEMDOS Filenames**

Under GEMDOS, each file located on a device is given a filename upon its creation which serves to provide identification for the file. The filename has two parts consisting of a name from one to eight characters long and an optional file extension of up to three characters long. If a file extension exists, the two components are separated by a period. The extension should serve to identify the format of the data whereas the name itself should identify the data itself.

Filenames may be changed after creation with the function `Frename()`; however, under no circumstances may two files with the same filename reside in the same directory.

All GEMDOS functions ignore the alphabetic case of file and pathnames. The following characters are legal filename characters:

<table>
<thead>
<tr>
<th>Legal GEMDOS Filename Characters</th>
</tr>
</thead>
<tbody>
<tr>
<td>A-Z, a-z, 0-9</td>
</tr>
<tr>
<td>! @ # $ % ^ &amp; ( )</td>
</tr>
<tr>
<td>+ - = ~ ` ; ' &quot; ,</td>
</tr>
<tr>
<td>&lt; &gt;</td>
</tr>
</tbody>
</table>

**GEMDOS Directories**

To further organize data, GEMDOS provides file directories (or folders). Each drive may contain any number of directories which, in turn, may contain files and additional directories. This organization creates a tree-like structure of files and folders. A file’s location in this tree is called the path.

Directory names follow the same format as GEMDOS filenames with a maximum filename length of 8 characters and an optional 3 character extension. The first directory of a disk which contains all subdirectories and files is called the root directory.

The `Dcreate()` and `Ddelete()` system calls are used to create and delete subdirectories.

Two special, system-created subdirectories are present in some directories. A subdirectory with the name ‘..’ (two periods) refers to the parent of the current directory. The ‘..’ subdirectory is present in every subdirectory.

A subdirectory with the name ‘.’ refers to the current directory. There is a ‘.’ subdirectory in every directory.
GEMDOS Path Specifications

To access a file, a complete path specification must be composed of the drive letter, directory name(s), and filename. A file named ‘TEST.PRG’ located in the ‘SYSTEM’ directory on drive ‘C’ would have a path specification like the following:

\C:\SYSTEM\TEST.PRG

The drive letter is the first character followed by a colon. Each directory and subdirectory is surrounded by backslashes. If ‘TEST.PRG’ were located in the root directory of ‘C’ the path specification would be:

\C:\TEST.PRG

The drive letter and colon may be omitted causing GEMDOS to reference the default drive as follows:

\TEST.PRG

A filename by itself will be treated as the file in the default directory and drive. The current GEMDOS directory and drive may be found with the functions Dgetpath() and Dgetdrv() respectively. They may be changed with the functions Dsetpath() and Dsetdrv().

Wildcards

The GEMDOS functions Fsfirst() and Fsnext() are used together to enumerate files of a given path specification. These two functions allow the use of wildcard characters to expand their search parameters.

The ‘?’ character is used to represent exactly one unknown character. The ‘*’ character is used to represent any number of unknown characters. The following table gives some examples of the uses of these characters.

<table>
<thead>
<tr>
<th>Filename</th>
<th>Found</th>
<th>Not Found</th>
</tr>
</thead>
<tbody>
<tr>
<td>**</td>
<td>All files</td>
<td>None</td>
</tr>
<tr>
<td>*.GEM</td>
<td>TEST.GEM</td>
<td>TEST.G</td>
</tr>
<tr>
<td></td>
<td>ATARI.GEM</td>
<td>ATARI.IMG</td>
</tr>
<tr>
<td>A?ARI.?</td>
<td>ATARI.O</td>
<td>ADARI.IMG</td>
</tr>
<tr>
<td></td>
<td>ADARI.C</td>
<td>ATARI.GEM</td>
</tr>
<tr>
<td>ATARI.??</td>
<td>ATARI.GEM</td>
<td>ATARI.O</td>
</tr>
<tr>
<td></td>
<td>ATARI.IMG</td>
<td>ATARI.C</td>
</tr>
</tbody>
</table>

Disk Transfer Address (DTA)

When using Fsfirst() and Fsnext() to build a list of files, TOS uses the Disk Transfer Address (DTA) to store information about each file found. The format for the DTA structure is as follows:
typedef struct
{
    BYTE d_reserved[21]; /* Reserved - Do Not Change */
    BYTE d_attrib; /* GEMDOS File Attributes */
    UWORD d_time; /* GEMDOS Time */
    UWORD d_date; /* GEMDOS Date */
    LONG d_length; /* File Length */
    char d_fname[14]; /* Filename */
} DTA;

When a process is started, its DTA is located at a point where it could overlay potentially important system structures. To avoid overwriting memory a process wishing to use Fsfirst() and Fsnext() should allocate space for a new DTA and use Fsetdta() to instruct the OS to use it. The original location of the DTA should be saved first, however. Its location can be found with the call Fgetdta(). At the completion of the operation the old address should be replaced with Fsetdta().

File Attributes

Every TOS file contains several attributes which define it more specifically. File attributes are specified when a file is created with Fcreate() and can be altered later with Fattrib().

The ‘read-only’ attribute bit is set to prevent modification of a file. This bit should be set at the user’s discretion and not cleared unless the user explicitly requests it.

If the ‘hidden’ attribute is set, the file will not be listed by the desktop or file selector. These files may still be accessed in a normal manner but will not be present in an Fsfirst() or Fsnext() search unless the correct Fsfirst() bits are present.

The ‘system’ attribute is unused by TOS but remains for MS-DOS compatibility.

The ‘volume label’ attribute should be present on a maximum of one file per drive. The file which has it set should be in the root directory and have a length of 0. The filename indicates the volume name of the drive.

The ‘archive’ attribute is a special bit managed by TOS which indicates whether a file has been written to since it was last backed up. Any time a Fcreate() call creates a file or Fwrite() is used on a file, the Archive bit is set. This enables file backup applications to know which files have been modified since the last backup. They are responsible for clearing this bit when backing up the file.

File Time/Date Stamp

When a file is first created a special field in its directory entry is updated to contain the date and time of creation. Fdatetime() can be used to access or modify this information as necessary.

File Maintenance

New files should be created with Fcreate(). When a file is successfully created a positive file handle is returned by the call. That handle is what is used to identify the file for all future operations until the file is closed. After a file is closed its handle is invalidated.
Files which are already in existence should be opened with `Fopen()`. As with `Fcreate()`, this call returns a positive file handle upon success which is used in all subsequent `GEMDOS` calls to reference the file.

Each process is allocated an OS dependent number of file handles. If an application attempts to open more files than this limit allows, the open or create call will fail with an appropriate error code. File handles may be returned to the system by closing the open file with `Fclose()`.

`Fopen()` may be used in read, write, or read/write mode. In read mode, `Fread()` may be used to access existing file contents. In write mode, any original information in the file is not cleared but the data may be overwritten with `Fwrite()`. In read/write mode, either call may be used interchangeably.

Every file has an associated file position pointer. This pointer is used to determine the location for the next read or write operation. This pointer is expressed as a positive offset from the beginning of the file (position 0) which is set upon first creating or opening a file. The pointer may be read or modified with the function `Fseek()`.

Existing files may be deleted with the `GEMDOS` call `Fdelete()`.

### File/Record Locking

File and record locking allow portions or all of a file to be locked against access from another computer over a network or another process in the same system.

All versions of TOS have the ability to support file and record locking but not all have the feature installed. If the `_FLK` cookie is present in the system cookie jar then the `Flock()` call is present. This call is used to create locks on individual sections (usually records) in a file.

Locking a file in use, when possible, is recommended to prevent other processes from modifying the file at the same time.

### Special File Handles

Several special file handles are available for access through the standard `Fopen()/Fread()/Fwrite()` calls. They are as follows:

<table>
<thead>
<tr>
<th>Name</th>
<th>Handle</th>
<th>Filename</th>
<th>Device</th>
</tr>
</thead>
<tbody>
<tr>
<td>GSH_BIOSCON</td>
<td>0xFFFFF</td>
<td>CON:</td>
<td>Console (screen). Special characters such as the carriage return, etc. are interpreted.</td>
</tr>
<tr>
<td>GSH_BIOSAUX</td>
<td>0xFFFE</td>
<td>AUX:</td>
<td>Modem (serial port). This is the ST-compatible port for machines with more than one.</td>
</tr>
<tr>
<td>GSH_BIOSPRN</td>
<td>0xFFFD</td>
<td>PRN:</td>
<td>Printer (attached to the Centronics Parallel port).</td>
</tr>
<tr>
<td>GSH_BIOSMIDIIN</td>
<td>0xFFF4</td>
<td></td>
<td>Midi In</td>
</tr>
<tr>
<td>GSH_BIOSMIDIOU</td>
<td>0xFFF5</td>
<td></td>
<td>Midi Out</td>
</tr>
</tbody>
</table>
These files may be treated like any other GEMDOS files for input/output and locking. Access to these devices is also provided with GEMDOS character calls (see later in this chapter).

**File Redirection**

Input and output to a file may be redirected to an alternate file handle. For instance you may redirect the console output of a TOS process to the printer.

File redirection is handled by the use of the `Fforce()` call. Generally you will want to make a copy of the file handle with `Fdup()` prior to redirecting the file so that it may be restored to normal operation when complete.

**Memory Management**

Atari systems support two kinds of memory. Standard RAM (sometimes referred to as ‘ST RAM’) is general purpose RAM that can be used for any purpose including video and DMA. Current Atari architecture limits the amount of standard RAM a system may have to 14MB.

Alternative RAM (sometimes referred to as ‘TT RAM’) can be accessed faster than standard RAM but is not suitable for video memory or DMA transfers.

The `Malloc()` and `Mxalloc()` calls allocate memory blocks from the system heap. `Malloc()` chooses the type of memory it allocates based on fields in the program header (see later in this chapter). `Mxalloc()` allows the application to choose the memory type at run-time.

MultiTOS uses memory protection to prevent an errant process from damaging another. It is possible with `Mxalloc()` to dynamically set the protection level of an allocated block.

Memory allocated with either `Malloc()` or `Mxalloc()` may be returned to the system with `Mfree()`. Memory allocated by a process is automatically freed when the process calls `Pterm()`.
have more than one child process. Depending on the mode used with \texttt{Pexec()}, the child may share data and address space and/or run concurrently (under \texttt{MultiTOS}) with the parent. \textbf{GEMDOS} executable files (\texttt{GEM} and \texttt{TOS} applications or desk accessories) contain the following file header:

<table>
<thead>
<tr>
<th>Name</th>
<th>Offset</th>
<th>Contents</th>
</tr>
</thead>
<tbody>
<tr>
<td>\texttt{PRG_magic}</td>
<td>0x00</td>
<td>This \texttt{WORD} contains the magic value (0x601A).</td>
</tr>
<tr>
<td>\texttt{PRG_tsize}</td>
<td>0x02</td>
<td>This \texttt{LONG} contains the size of the TEXT segment in bytes.</td>
</tr>
<tr>
<td>\texttt{PRG_dsizet}</td>
<td>0x06</td>
<td>This \texttt{LONG} contains the size of the DATA segment in bytes.</td>
</tr>
<tr>
<td>\texttt{PRG_bsize}</td>
<td>0x0A</td>
<td>This \texttt{LONG} contains the size of the BSS segment in bytes.</td>
</tr>
<tr>
<td>\texttt{PRG_ssizet}</td>
<td>0x0E</td>
<td>This \texttt{LONG} contains the size of the symbol table in bytes.</td>
</tr>
<tr>
<td>\texttt{PRG_res1}</td>
<td>0x12</td>
<td>This \texttt{LONG} is unused and is currently reserved.</td>
</tr>
<tr>
<td>\texttt{PRGFLAGS}</td>
<td>0x16</td>
<td>This \texttt{LONG} contains flags which define certain process characteristics (as defined below).</td>
</tr>
<tr>
<td>\texttt{ABSFLAG}</td>
<td>0x1A</td>
<td>This \texttt{WORD} flag should be non-zero to indicate that the program has no fixups or 0 to indicate it does. Since some versions of \texttt{TOS} handle files with this value being non-zero incorrectly, it is better to represent a program having no fixups with 0 here and placing a 0 longword as the fixup offset.</td>
</tr>
</tbody>
</table>

**Text Segment**

0x1C  This area contains the program’s TEXT segment. A process is started by \texttt{JMP’ing} to \texttt{BYTE 0} of this segment with the address of your processes basepage at 4(sp).

**Data Segment**

\texttt{PRG\_tsizet + 0x1C}  This area contains the program’s DATA segment (if one exists).

**Symbol Segment**

\texttt{PRG\_tsizet + PRG\_dsizet + 0x1C}  This area contains the program’s symbol table (if there is one). The symbol table area is used differently by different compiler vendors. Consult them for the format.

**Fixup Offset**

\texttt{PRG\_tsizet + PRG\_dsizet + PRG\_ssizet + 0x1C}  This \texttt{LONG} indicates the first location in the executable (as an offset from the beginning) containing a longword needing a fixup. A 0 means there are no fixups.
Fixup Information  \( PRG\_tsize + PRG\_dsize + PRG\_ssize + 0x20 \)  
This area contains a stream of **BYTES** containing fixup information. Each byte has a significance as follows:

<table>
<thead>
<tr>
<th>Value</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>End of list.</td>
</tr>
<tr>
<td>1</td>
<td>Advance 254 bytes.</td>
</tr>
<tr>
<td>2-254 (even)</td>
<td>Advance this many bytes and fixup the longword there.</td>
</tr>
</tbody>
</table>

**PRGFLAGS** is a bit field defined as follows:

<table>
<thead>
<tr>
<th>Definition</th>
<th>Bit(s)</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>PF_FASTLOAD</td>
<td>0</td>
<td>If set, clear only the BSS area on program load, otherwise clear the entire heap.</td>
</tr>
<tr>
<td>PF_TTRAMLOAD</td>
<td>1</td>
<td>If set, the program may be loaded into alternative RAM, otherwise it must be loaded into standard RAM.</td>
</tr>
<tr>
<td>PF_TTRAMMEM</td>
<td>2</td>
<td>If set, the program’s <strong>Malloc()</strong> requests may be satisfied from alternative RAM, otherwise they must be satisfied from standard RAM.</td>
</tr>
<tr>
<td></td>
<td>3</td>
<td>Currently unused.</td>
</tr>
<tr>
<td>See left.</td>
<td>4 &amp; 5</td>
<td>If these bits are set to 0 (<strong>PF_PRIVATE</strong>), the processes’ entire memory space will be considered private (when memory protection is enabled).</td>
</tr>
<tr>
<td></td>
<td></td>
<td>If these bits are set to 1 (<strong>PF_GLOBAL</strong>), the processes’ entire memory space will be readable and writable by any process (i.e. global).</td>
</tr>
<tr>
<td></td>
<td></td>
<td>If these bits are set to 2 (<strong>PF_SUPERVISOR</strong>), the processes’ entire memory space will only be readable and writable by itself and any other process in supervisor mode.</td>
</tr>
<tr>
<td></td>
<td></td>
<td>If these bits are set to 3 (<strong>PF_READABLE</strong>), the processes’ entire memory space will be readable by any application but only writable by itself.</td>
</tr>
<tr>
<td></td>
<td>6-15</td>
<td>Currently unused.</td>
</tr>
</tbody>
</table>

When a process is started by **GEMDOS**, it allocates all remaining memory, loads the process into that memory, and JMP’s to the first byte of the application’s TEXT segment with the address of the program’s basepage at 4(sp). An application should use the basepage information to decide upon the amount of memory it actually needs and **Mshrink()** to return the rest to the system. The exception to this is that desk accessories are only given as much space as they need (as indicated by their program header) and their stack space is pre-assigned.
The following code illustrates the proper way to release system memory and allocate your stack (most ‘C’ startup routines do this for you):

```assembly
stacksize = $2000 ; 8K
.text
_start:
  move.l 4(sp),a0 ; Obtain pointer to basepage
  move.l a0,basepage ; Save a copy
  move.l $18(a0),a1 ; BSS Base address
  adda.l $1C(a0),a1 ; Add BSS size
  adda.l #stacksize,a1 ; Add stack size
  move.l a1,sp ; Move your stack pointer to
                 ; your new stack.
  suba.l basepage,a1 ; TPA size
  move.l a1,-(sp)
  move.l basepage,-(sp)
  clr.w -(sp)
  move.w #$4a,-(sp) ; Mshrink()
  trap #1
  lea 12(sp),sp ; Fix up stack
              ; and fall through to main
_main:
...
.bss
basepage: ds.l 1
.end
```

The **GEMDOS BASEPAGE** structure has the following members:

<table>
<thead>
<tr>
<th>Name</th>
<th>Offset</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>p_lowtpa</td>
<td>0x00</td>
<td>This LONG contains a pointer to the Transient Program Area (TPA).</td>
</tr>
<tr>
<td>p_hitpa</td>
<td>0x04</td>
<td>This LONG contains a pointer to the top of the TPA + 1.</td>
</tr>
<tr>
<td>p_tbase</td>
<td>0x08</td>
<td>This LONG contains a pointer to the base of the text segment</td>
</tr>
<tr>
<td>p_tlen</td>
<td>0x0C</td>
<td>This LONG contains the length of the text segment.</td>
</tr>
<tr>
<td>p_dbase</td>
<td>0x10</td>
<td>This LONG contains a pointer to the base of the data segment.</td>
</tr>
<tr>
<td>p_dlen</td>
<td>0x14</td>
<td>This LONG contains the length of the data segment.</td>
</tr>
<tr>
<td>p_bbbase</td>
<td>0x18</td>
<td>This LONG contains a pointer to the base of the BSS segment.</td>
</tr>
<tr>
<td>p_blen</td>
<td>0x1C</td>
<td>This LONG contains the length of the BSS segment.</td>
</tr>
<tr>
<td>p_dta</td>
<td>0x20</td>
<td>This LONG contains a pointer to the processes’ DTA.</td>
</tr>
</tbody>
</table>
Processes terminate themselves with either `Pterm0()`, `Pterm()`, or `Ptermres()`. `Ptermres()` allows a segment of a file to remain behind in memory after the file itself terminates (this is mainly useful for TSR utilities).

The Atari Extended Argument Specification

When a process calls `Pexec()` to launch a child, the child may receive a command line up to 125 characters in length. The command line does not normally contain information about the process itself (what goes in `argv[0]` in ‘C’). The Atari Extended Argument Specification (ARGV) allows command lines of any length and correctly passes the child the command that started it. The ARGV specification works by passing the command tail in the child’s environment rather than in the command line buffer.

Both the parent and child have responsibilities when wanting to correctly handle the ARGV specification. If a process wishes to launch a child with a command line of greater than 125 characters it should follow these steps:

1. Allocate a block of memory large enough to hold the existing environment, the string ‘ARGV=’ and its terminating `NULL`, a string containing the complete path and filename of the child process and its terminating `NULL`, and a string containing the child’s command line arguments and its terminating `NULL`.
2. Next, copy these elements into the reserved block in the order given above.
3. Finally, call `Pexec()` with this environment string and a command line containing a length byte of 127 and the first 125 characters of the command line with a terminating `NULL`.

For a child to correctly establish that a parent process is using ARGV it should check for the length byte of 127 and the ARGV variable. Some parents may assign a value to ARGV (found between the ‘ARGV=’ and the terminating `NULL` byte). It should be skipped over and ignored. If a child detects that its parent is using ARGV, it then has the responsibility of breaking down the environment into its components to properly obtain its command line elements.

It should be noted that many compilers include ARGV parsing in their basic startup stubs. In addition, applications running under MultiTOS should use the AES call `shel_write()` as it automatically creates an ARGV environment string.
**GEMDOS Vectors**

GEMDOS reserves eight system interrupt vectors (of which only three are used) for various system housekeeping. The BIOS function `Setexc()` should be used to redirect these vectors when necessary. The GEMDOS vectors are as follows:

<table>
<thead>
<tr>
<th>Name</th>
<th>Setexc() Vector Number</th>
<th>Usage</th>
</tr>
</thead>
<tbody>
<tr>
<td>VEC_TIMER</td>
<td>0x0100</td>
<td>Timer Tick Vector: This vector is jumped through 50 times per second to maintain the time-of-day clock and accomplish other system housekeeping. A process intercepting this vector does not have to preserve any registers but should jump through the old vector when completed. Heavy use of this vector can severely affect system performance. Return from this handler with RTS.</td>
</tr>
<tr>
<td>VEC_CRITICALERR</td>
<td>0x0101</td>
<td>Critical Error Handler: This vector is used by the BIOS to service critical alerts (an <code>Rwabs()</code> disk error or media change request). When called, the WORD at 4(sp) is a GEMDOS error number. On return, D0.L should contain 0x0001000 to retry the operation, 0 to ignore the error, or <code>0xFFFFFFxx</code> to return an error code (xx). D3-D7 and A3-A6 must be preserved by the handler. Return from this handler with RTS.</td>
</tr>
<tr>
<td>VEC_PROCTERM</td>
<td>0x0102</td>
<td>Process Terminate Vector: This vector is called just prior to the termination of a process ended with CTRL-C. Return from this handler with RTS.</td>
</tr>
<tr>
<td></td>
<td>0x103-0x0107</td>
<td>Currently unused.</td>
</tr>
</tbody>
</table>

**MiNT**

MiNT is Now TOS (MiNT) is the extension to GEMDOS that allows GEMDOS to multitask under MultiTOS. MiNT also provides memory protection (on a 68030 or higher) to protect an errant process from disturbing another.

**Processes**

MiNT assigns each process a process identifier and a process priority value. The identifier is used to distinguish the process from others in the multitasking environment. `Pgetpid()` is used to obtain the MiNT ID of the process and `Pgetppid()` can be used to obtain the ID of the processes’ parent.

MiNT also supports networking file systems that support the concept of user and process group control. The `Pgetpgrp()`, `Psetpgrp()`, `Pgetuid()`, `Psetuid()`, `Pgeteuid()`, and `Pseteuid()` get and set the process, user, and effective user ID for a process.

MiNT has complete control over the amount of time allocated to individual processes. It is possible, however, to set a process ‘delta’ value with `Pnice()` or `Prenice()` which will be used by MiNT to decide the amount of processor time a process will get per timeslice. `Syield()` can be used to surrender the remaining portion of a timeslice.
Information about a processes’ resource usage can be obtained by calling `Prusage()`. These values can be modified with `Psetlimit()`. System configuration capabilities may be obtained with `Sysconf()`.

Each process can have a user-defined longword value assigned to itself with `Pusrval()`.

The functions `Pwait()`, `Pwait3()`, and `Pwaitpid()` attempt to determine the exit codes of stopped child processes.

**Threads**

It is possible under MiNT to split a single process into ‘threads’. These threads continue execution independently as unique processes. The `Pfork()` and `Pvfork()` calls are used to split a process into threads.

The original process that calls `Pfork()` or `Pvfork()` is considered the parent and the newly created process is considered the child.

Child processes created with `Pfork()` share the TEXT segment of the parent, however they are given a copy of the DATA and BSS segments. Both the parent and child execute concurrently.

Child processes created with `Pvfork()` share the entire program code and data space including the processor stack. The parent process is suspended until the child exits or calls `Pexec()`’s mode 200.

Child processes started with either call may make GEM calls but a child process started with `Pfork()` must call `appl_init()` to force GEM to uniquely recognize it as an independent process. This is not necessary with `Pvfork()` because all program variables are shared.

The following is a simple example of using a thread in a GEM application:

```c
VOID UserSelectedPrint( VOID )
{
    /* Prevent the user from editing buffer being printed. */
    LockBufferFromEdits();

    if( Pfork() == 0 )
    {
        /* Child enters here */
        appl_init(); /* Required for GEM threads. */
        DisplayPrintingWindow(); /* Do our task. */
        PrintBuffer();

        /* Send an AES message to the parent telling it to unlock buffer. */
        SendCompletedMessageToParent();

        /* Cleanup and exit thread. */
        appl_exit();
    }
}
```
File System Extensions

MiNT provides several new file and directory manipulation functions that work with TOS and other loadable file systems. The Fcntl() function performs a large number of file-based tasks many of which apply to special files like terminal emulators and ‘U:\’ files. Fxattr() is used to obtain a file’s extended attributes. Some extended attributes are not relevant to the TOS file system and will not return meaningful values (see the Function Reference for details).

Fgetchar() and Fputchar() can be used to get and put single characters to a file. Fstat() and Foutstat() are used to determine the input or output status of a file. Fselect() is used to select from a group of file handles those ready to be read from or written to (often used for pipes).

Flink(), Fsymlink(), and Freadlink() are used to create hard and symbolic links to another file. Links are not supported by all file systems (see the entries for these functions for more details).

Some file systems may support the concept of file ownership and access permissions (TOS does not). The Fchown() and Fchmod() calls are used to adjust the ownership flags and access permissions of a file. Pumask() can be used to set the minimum access permissions assigned to each subsequently created file.

Fmidipipe() is used to redirect the file handles used for MIDI input and output.

MiNT provides four new functions for directory enumeration (they provide similar functionality to Fsfirst() and Fsnext() with a slightly easier interface). Dopendir() is used to open a directory for enumeration. D readdir() steps through each entry in a directory. Drewinddir() resets the file pointer to the beginning of the directory. Dclosedir() closes a directory.

Dlock() allows disk-formatters and other utilities which require exclusive access to a drive the ability to lock a physical device from other processes.

Dgetcwd() allows a process to obtain the current GEMDOS working directory for any process in the system (including itself).

Dcntl() performs device and file-system specific operations (consult the Function Reference for more details).

Pseudo Drives

MiNT creates a pseudo drive ‘U:’ which provides access to device drivers, processes, and other system resources. In addition to creating a directory on drive U: for each system drive, MiNT may create any of the following directories at the ROOT of the drive:

<table>
<thead>
<tr>
<th>Folder Name</th>
<th>Contents</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
</tr>
</tbody>
</table>
Drive directories on ‘U:’ act as if they were accessed by their own drive letter. Folder ‘U:\C\’ contains the same files and folders as ‘C:\’.

### The ‘U:\PROC’ Directory

Each system process has a file entry in the ‘U:\PROC’ directory. The filename given a process in this directory is the basename for the file (without extension) with an extension consisting of the MiNT process identifier. The MINIWIN.PRG application might have an entry named ‘MINIWIN.003’.

The file size listed corresponds to the amount of memory the process is using. The time and date stamp contains the length of time the process has been executing as if it were started on Jan. 1st, 1980 at midnight. The file attribute bits tell special information about a process as follows:

<table>
<thead>
<tr>
<th>Name</th>
<th>Attribute Byte</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>PROC_RUN</td>
<td>0x00</td>
<td>The process is currently running.</td>
</tr>
<tr>
<td>PROC_READY</td>
<td>0x01</td>
<td>The process is ready to run.</td>
</tr>
<tr>
<td>PROC_TSR</td>
<td>0x02</td>
<td>The process is a TSR.</td>
</tr>
<tr>
<td>PROC_WAITEVENT</td>
<td>0x20</td>
<td>The process is waiting for an event.</td>
</tr>
<tr>
<td>PROC_WAITIO</td>
<td>0x21</td>
<td>The process is waiting for I/O.</td>
</tr>
<tr>
<td>PROC_EXITED</td>
<td>0x22</td>
<td>The process has been exited but not yet released.</td>
</tr>
<tr>
<td>PROC_STOPPED</td>
<td>0x24</td>
<td>The process was stopped by a signal.</td>
</tr>
</tbody>
</table>

### Loadable Devices

MiNT contains a number of built-in devices and also supports loadable device drivers. Current versions of MiNT may contain any of the following devices:

<table>
<thead>
<tr>
<th>Device Filename</th>
<th>Device</th>
</tr>
</thead>
<tbody>
<tr>
<td>CENTR</td>
<td>Centronics Parallel Port</td>
</tr>
<tr>
<td>MODEM1</td>
<td>Modem Port 1</td>
</tr>
<tr>
<td>MODEM2</td>
<td>Modem Port 2</td>
</tr>
<tr>
<td>SERIAL1</td>
<td>Serial Port 1</td>
</tr>
<tr>
<td>SERIAL2</td>
<td>Serial Port 2</td>
</tr>
<tr>
<td>MIDI</td>
<td>MIDI ports</td>
</tr>
<tr>
<td>PRN</td>
<td>PRN: device (usually the Centronics Parallel Port)</td>
</tr>
<tr>
<td>AUX</td>
<td>AUX: device (usually the RS232 Port)</td>
</tr>
<tr>
<td>CON</td>
<td>Current Terminal</td>
</tr>
<tr>
<td>TTY</td>
<td>Current Terminal (same as CON)</td>
</tr>
<tr>
<td>STDIN</td>
<td>Current File Handle 0 (standard input)</td>
</tr>
<tr>
<td>STDOUT</td>
<td>Current File Handle 1 (standard output)</td>
</tr>
<tr>
<td>STDERR</td>
<td>Current File Handle 2 (standard error)</td>
</tr>
<tr>
<td>CONSOLE</td>
<td>Physical Console (keyboard/screen)</td>
</tr>
</tbody>
</table>
Each of these devices is represented by a filename (as shown in the table above) in the ‘U:\DEV\’ directory. Using standard GEMDOS calls (ex: Fread() and Fwrite()) on these files yields the same results as accessing the device directly. New devices, including those directly accessible by the BIOS, may be added to the system with the Dcntl() call using a parameter of DEV_INSTALL, DEV_NEWBIOS, or DEV_NEWTTY. See the Dcntl() call for details.

MiNT versions 1.08 and above will automatically load device drivers with an extension of ‘.XDD’ found in the root or ‘\MULTITOS’ directory. ‘.XDD’ files are special device driver executables which are responsible for installing one (or more) new devices. MiNT will load the file and JSR to the first instruction in the TEXT segment (no parameters are passed). The device driver executable should not attempt to Mshrink() or create a stack (one has already been created).

The ‘.XDD’ may then either install its device itself with Dcntl() and return DEV_SELFINST (1L) in register D0 or return a pointer to a DEVDRV structure to have the MiNT kernel install it (the ‘U:\DEV\’ filename will be the same as the first eight characters of the ‘.XDD’ file). If for some reason, the device can not be initialized, 0L should be returned in D0.

When creating a new MiNT device with Dcntl( DEV_INSTALL, devname, &dev_descr) the structure dev_descr contains a pointer to your DEVDRV structure defined as follows:

```c
typedef struct devdrv
{
    LONG (*open)( FILEPTR *f );
    LONG (*write)( FILEPTR *f, char *buf, LONG bytes );
    LONG (*read)( FILEPTR *f, char *buf, LONG bytes );
    LONG (*lseek)( FILEPTR *f, LONG where, LONG whence );
    LONG (*ioctl)( FILEPTR *f, WORD mode, VOIDP buf );
    LONG (*datime)( FILEPTR *f, WORD *timeptr, WORD rwflag );
    LONG (*close)( FILEPTR *f, WORD pid );
    LONG (*select)( FILEPTR *f, LONG proc, WORD mode );
    LONG (*unselect)( FILEPTR *f, LONG proc, WORD mode );
    LONG reserved[3];
} DEVDRV;
```

Each of the assigned members of this structure should point to a valid routine that provides the named operation on the device. The routine must preserve registers D2-D7 and A2-A7 returning its completion code in D0. No operating system TRAPs should be called from within these routines, however, using the vector tables provided in the kerinfo structure returned from the Dcntl() call, GEMDOS and BIOS calls may be used. The specific function that each routine is responsible for is as follows:
### open

This routine is called by the MiNT kernel after a FILEPTR structure has been created for a file determined to be associated with the device. The routine should perform whatever initialization is necessary and exit with a standard GEMDOS completion code.

This routine is responsible for validating the sharing mode and other file flags to verify that the file may be legally opened and should respond with an appropriate error code if necessary.

### write

This routine should write bytes number of BYTES from buf to the file specified in FILEPTR. If the file pointer has the O_APPEND bit set, the kernel will perform an lseek() call to the end of the file prior to calling this function. If the lseek()/write() series of calls does not guarantee that data will be written at the end of a file associated with your device, this function must ensure that the data specified is actually written at the end of the file.

This function should return with a standard GEMDOS error code or the actual number of BYTES written to the file when complete.

### read

This routine should read bytes number of BYTES from the file specified in FILEPTR and place them in the buffer buf. This function should return with a standard GEMDOS error code or the actual number of bytes read by the routine.

### lseek

This routine should move the file position pointer to the appropriate location in the file as specified by the parameter where in relation to the seek mode whence. Seek modes are the same as with Fseek(). The routine should return a GEMDOS error code or the absolute new position from the start of the file if successful.

### ioctl

This routine is called from the system's perspective as Fcntl() and is used to perform file system/device specific functions. At the very least, your device should support FIONREAD, FIONWRITE, and the file/record locking modes of Fcntl(). The arg parameter of Fcntl() is passed as buf.

### datime

This routine is used to read or modify the date/time attributes of a file. timeptr is a pointer to two LONGs containing the time and date of the file respectively. These LONGs should be used to set the file date and time if rwflag is non-zero or filled in with the file's creation date and time if rwflag is 0.

This function should return with a standard GEMDOS error code or E_OK (0) if successful.

### close

This routine is used by the kernel to close an open file. Be aware that if f->links is non-zero, additional processes still have valid handles to the file. If f->links is 0 then the file is really being closed. pid specifies the process closing the file and may not necessarily be the same as the process that opened it.

Device drivers should set the O_LOCK bit on f->flag when the F_SETLK or F_SETLKW ioctl() call is made. This bit can be tested for when a file is closed and all locks on all files associated with the same physical file owned by process pid should be removed. If the file did not have any locks created on it by process pid, then no locks should be removed.

This routine should return with a standard GEMDOS error code or E_OK (0) if successful.

### select

This routine is called when a call to Fselect() names a file handled by this device. If mode is O_RDONLY then the select is for reading, otherwise, if mode is O_WRONLY then it is for writing. If the user Fselect()’s for both reading and writing then two calls to this function will be made.

The routine should return 1L if the device is ready for reading or writing (as appropriate) or it should return 0L and arrange to ‘wake up’ process proc when I/O becomes possible. This is usually accomplished by calling the wakeselect() member function of the kernel structure. Note that the value in proc is not the same as a PID and is actually a pointer to a PROC structure private to the MiNT kernel.

### unselect

This routine is called when a device waiting for I/O should no longer be waited for. The mode and...
proc parameters are the same as with select(). As with select(), if neither reading nor writing is to be waited for, two calls to this function will be made.

This routine should return a standard GEMDOS error code or E_OK (0) if successful.

The FILEPTR structure pointed to by a parameter of each of the above calls is defined as follows:

```c
typedef struct fileptr
{
    WORD     links;
    UWORD    flags;
    LONG     pos;
    LONG     devinfo;
    fcookie  fc;
    struct devdrv *dev;
    struct fileptr *next;
} FILEPTR;
```

The members of FILEPTR have significance as follows:

<table>
<thead>
<tr>
<th>Member</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>links</td>
<td>This member contains a value indicating the number of copies of this file descriptor currently in existence.</td>
</tr>
<tr>
<td>flags</td>
<td>This member contains a bit mask which indicates several attributes (logically OR’ed together) of the file as follows:</td>
</tr>
<tr>
<td>Name</td>
<td>Mask</td>
</tr>
<tr>
<td>O_RDONLY</td>
<td>0x0000</td>
</tr>
<tr>
<td>O_WRONLY</td>
<td>0x0001</td>
</tr>
<tr>
<td>O_RDWR</td>
<td>0x0002</td>
</tr>
<tr>
<td>O_EXEC</td>
<td>0x0003</td>
</tr>
<tr>
<td>O_APPEND</td>
<td>0x0008</td>
</tr>
<tr>
<td>O_COMPAT</td>
<td>0x0000</td>
</tr>
<tr>
<td>O_DENYRW</td>
<td>0x0010</td>
</tr>
<tr>
<td>O_DENYW</td>
<td>0x0020</td>
</tr>
<tr>
<td>O_DENYR</td>
<td>0x0030</td>
</tr>
<tr>
<td>O_DENYNONE</td>
<td>0x0040</td>
</tr>
<tr>
<td>O_NOINHERIT</td>
<td>0x0080</td>
</tr>
<tr>
<td>O_NDELAY</td>
<td>0x0100</td>
</tr>
<tr>
<td>O_CREAT</td>
<td>0x0200</td>
</tr>
<tr>
<td>O_TRUNC</td>
<td>0x0400</td>
</tr>
<tr>
<td>O_EXCL</td>
<td>0x0800</td>
</tr>
<tr>
<td>O_TTY</td>
<td>0x2000</td>
</tr>
<tr>
<td>O_HEAD</td>
<td>0x4000</td>
</tr>
<tr>
<td>O_LOCK</td>
<td>0x8000</td>
</tr>
<tr>
<td>pos</td>
<td>This field is initialized to 0 when a file is created and should be used by the device driver to store the file position pointer.</td>
</tr>
<tr>
<td>devinfo</td>
<td>This field is reserved for use between the file system and the device driver and may be used as desired. The exception to this is if the file is a TTY, in which case devinfo must be a pointer to a tty structure.</td>
</tr>
<tr>
<td>fc</td>
<td>This is the file cookie for the file as follows:</td>
</tr>
</tbody>
</table>

```c
typedef struct f_cookie
```
{  
    FILESYS *fs;
    UWORD dev;
    UWORD aux;
    LONG index;
}

\textit{fs} is a pointer to the file system structure responsible for this device. \textit{dev} is a \texttt{UWORD} giving a useful device ID (such as the \texttt{Rwabs()} device number). The meaning of \textit{aux} is file system dependent. \textit{index} should be used by file systems to provide a unique means of identifying a file.

\textit{dev} This is a pointer to the \texttt{DEVDRV} structure of the device driver responsible for this file.

\textit{next} This pointer may be used by device drivers to link copies of duplicate file descriptors to implement file locking or sharing code.

Upon successful return from the \texttt{Dcntl()} call, a pointer to a \texttt{kerinfo} structure will be returned. The \texttt{kerinfo} structure is defined below:

define LONG (*Func)();

struct kerinfo
{
    WORD maj_version;
    WORD min_version;
    UWORD default_mode;
    WORD reserved1;
    Func *bios_tab;
    Func *dos_tab;
    VOID (*drvchng)( UWORD dev );
    VOID (*trace)( char *, ... );
    VOID (*debug)( char *, ... );
    VOID (*alert)( char *, ... );
    VOID (*fatal)( char *, ... );
    VOIDP (*kmalloc)( LONG size );
    VOID (*kfree)( VOIDP memptr );
    VOIDP (*umalloc)( LONG size );
    VOID (*ufree)( LONG memptr );
    WORD (*strnicmp)( char *str1, char *str2, WORD maxsrch );
    WORD (*strcmp)( char *str1, char *str2 );
    char (*)(char *str);
    char (*)(char *str);
    WORD (*sprintf)( char *strbuf, const char *fmtstr, ... );
    VOID (*millis_time)( ULONG ms, WORD *td );
    LONG (*unixtim)( UWORD time, UWORD date );
    LONG (*dostim)( LONG unixtime );
    VOID (*nap)( UWORD n );
    VOID (*sleep)( WORD que, WORD cond );
    VOID (*wake)( WORD que, WORD cond );
    VOID (*wakeselect)( LONG proc );
    WORD (*denyshare)( FILEPTR *list, FILEPTR *f );
    LOCK (*denylock)( LOCK *list, LOCK *new );
}
The members of the kerinfo structure are defined as follows:

<table>
<thead>
<tr>
<th>Member</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>maj_version</td>
<td>This WORD contains the kernel version number.</td>
</tr>
<tr>
<td>min_version</td>
<td>This WORD contains the minor kernel version number.</td>
</tr>
<tr>
<td>default_mode</td>
<td>This UWORD contains the default access permissions for a file.</td>
</tr>
<tr>
<td>reserved1</td>
<td>Reserved.</td>
</tr>
<tr>
<td>bios_tab</td>
<td>This is a pointer to the BIOS function jump table. Calling bios_tab<a href="">0x00</a> is equivalent to calling Getmpb() and is the only safe way from within a device driver or file system.</td>
</tr>
<tr>
<td>dos_tab</td>
<td>This is a pointer to the GEMDOS function jump table. Calling dos_tab<a href="">0x3D</a> is equivalent to calling Fopen() and is the only safe way from within a device driver or file system.</td>
</tr>
<tr>
<td>drvchng</td>
<td>This function should be called by a device driver if a media change was detected on the device during an operation. The parameter dev is the BIOS device number of the device.</td>
</tr>
<tr>
<td>trace</td>
<td>This function is used to send information messages to the kernel for debugging purposes.</td>
</tr>
<tr>
<td>debug</td>
<td>This function is used to send error messages to the kernel for debugging purposes.</td>
</tr>
<tr>
<td>alert</td>
<td>This function is used to send serious error messages to the kernel for debugging purposes.</td>
</tr>
<tr>
<td>fatal</td>
<td>This function is used to send fatal error messages to the kernel for debugging purposes.</td>
</tr>
<tr>
<td>kmalloc</td>
<td>Use this internal heap memory management function to allocate memory.</td>
</tr>
<tr>
<td>kfree</td>
<td>Use this internal heap memory management function to free memory allocated with kmalloc().</td>
</tr>
<tr>
<td>umalloc</td>
<td>Use this internal heap memory management function to allocate memory and attach it to the current process. The memory will be released automatically when the current process exits.</td>
</tr>
<tr>
<td>ufree</td>
<td>Use this internal heap memory management function to allocate memory allocated with ufree().</td>
</tr>
<tr>
<td>strnicmp</td>
<td>This function compares maxsrch characters of str1 to str2 and returns a negative value if str1 is lower than str2, a positive value if str1 is higher than str2, or 0 if they are equal.</td>
</tr>
<tr>
<td>stricmp</td>
<td>This function compares two NULL terminated strings, str1 to str2, and returns a negative value if str1 is lower than str2, a positive value if str1 is higher than str2, or 0 if they are equal.</td>
</tr>
<tr>
<td>strlwr</td>
<td>This function converts all alphabetic characters in str to lower case.</td>
</tr>
<tr>
<td>strupr</td>
<td>This function converts all alphabetic characters in str to upper case.</td>
</tr>
<tr>
<td>sprintf</td>
<td>This function is the same as the ‘C’ library sprintf() function except that it will only convert SPRINTF_MAX characters (defined in TOSDEFS.H).</td>
</tr>
<tr>
<td>millis_time</td>
<td>This function converts the millisecond time value in ms to a GEMDOS time in td[0] and date in td[1].</td>
</tr>
<tr>
<td>unixtim</td>
<td>This function converts a GEMDOS time and date in a UNIX format LONG.</td>
</tr>
<tr>
<td>dostim</td>
<td>This function converts a UNIX format LONG time/date value into a GEMDOS time/date value. The return value contains the time in the upper WORD and the date in the lower WORD.</td>
</tr>
<tr>
<td>nap</td>
<td>This function causes a delay of n milliseconds.</td>
</tr>
<tr>
<td>sleep</td>
<td>This function causes the current process to sleep, placing it on the system queue que until condition cond is met.</td>
</tr>
<tr>
<td>wake</td>
<td>This function causes all processes in que que, waiting for condition cond, to be woken.</td>
</tr>
<tr>
<td>wakeselect</td>
<td>This function wakes a process named by the code proc currently doing a select operation.</td>
</tr>
<tr>
<td>denyshare</td>
<td>This function determines whether the sharing mode of f conflicts with any of the files given in the linked list list.</td>
</tr>
<tr>
<td>denylock</td>
<td>This function determines whether a new lock new conflicts with any existing lock in the linked list list. The LOCK structure is used internally by the kernel and is defined as follows:</td>
</tr>
</tbody>
</table>
**Loadable File Systems**

MiNT supports loadable file systems to provide support for those other than TOS (such as POSIX, HPFS, ISO 9660 CD-ROM, etc.) The MiNT kernel will automatically load file system `.XFS` executables found in the `\MULTITOS` or root directory. As of MiNT version 1.08, it is also possible to have a TSR program install a file system with the `Dcntl()` call.

When the file system is executed by MiNT (i.e. not via `Dcntl()`), MiNT creates an 8K stack and shrinks the TPA so a call to `Mshrink()` is not necessary. The first instruction of the code segment of the file is JSR’ed to with a pointer to a `kerinfo` (as defined above) structure at 4(sp). The file system should use this entry point to ensure that it is running on the minimum version of MiNT needed and that any other aspects of the system are what is required for the file system to operate.

It is not necessary to scan existing drives to determine if they are compatible with the file system as that is accomplished with the file system `root()` function (defined below). If the file system needs to make MiNT aware of drives that would not be automatically recognized by the system, it should update the longword variable `_drvbits` at location 0x04F2 appropriately.

If the file system was unable to initialize itself or the host system is incapable of supporting it, the entry stub should return with a value of 0L in d0. If the file system installs successfully, it should return a pointer to a `FILESYS` (defined below) structure in d0. A file system should never call `Pterm()` or `Ptermres()`.

All file system functions, including the entry stub, must preserve registers d2-d7 and a2-a7. Any return values should be returned in d0. Function arguments are passed on the stack. The following listing defines the `FILESYS` structure:

```
typedef struct filesys
{
struct filesys  *next;
LONG   fsflags;
LONG   (*root)( WORD drv, fcookie *fc);
LONG   (*lookup)( fcookie *dir, char *name, fcookie *fc);
LONG   (*creat)( fcookie *dir, char *name, UWORD mode, WORD attrib,
               fcookie *fc);
DEVDRV  *(getdev)( fcookie *fc, LONG *devspecial);
}FILESYS;
```
The members of the FILESYS structure are interpreted by MiNT as follows:

<table>
<thead>
<tr>
<th>Member</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>next</strong></td>
<td>This member is a pointer to the next FILESYS structure in the kernel's linked list. It should be left as NULL.</td>
</tr>
<tr>
<td><strong>fsflags</strong></td>
<td>This is a bit mask of flags which define attributes of the file system as follows:</td>
</tr>
<tr>
<td>Name</td>
<td>Mask</td>
</tr>
<tr>
<td>------</td>
<td>-----</td>
</tr>
<tr>
<td>FS_KNOPARSE</td>
<td>0x01</td>
</tr>
<tr>
<td>FS_CASESENSITIVE</td>
<td>0x02</td>
</tr>
<tr>
<td>FS_NOXBIT</td>
<td>0x04</td>
</tr>
<tr>
<td><strong>root</strong></td>
<td>This function is called by the kernel to retrieve a file cookie for the root directory of the drive associated with BIOS device dev. When initializing, the kernel will query each file system, in turn, to determine which file system should handle a particular drive. If your file system recognizes the drive specified by dev it should fill in the fcookie structure as appropriate and return E_OK. If the drive is not compatible with your file system, return an appropriate negative GEMDOS error code (usually EDRIVE).</td>
</tr>
</tbody>
</table>
lookup

This function should translate a file name into a cookie. If the FS_KNOPARSE bit of fsflags is not set, name will be the name of a file in the directory specified by the fcookie dir. If the FS_KNOPARSE bit was set, name will be a path name relative to the specified directory dir.

If the file is found, the fcookie structure fc should be filled in with appropriate details and either E_OK or EMount (if name is ‘.’ and dir specifies the root directory) should be returned, otherwise an appropriate error code (like EFILNF) should be returned.

A lookup() call with a NULL name or with a name of ‘.’ should always succeed and return a cookie representing the current directory. When creating a file cookie, symbolic links should never be followed.

creat

This function is used by the kernel to instruct the file system to create a file named name in the directory specified by dir with attrib attributes (as defined by Fattrib()) and mode permissions as follows:

<table>
<thead>
<tr>
<th>Name</th>
<th>Mask</th>
<th>Permission</th>
</tr>
</thead>
<tbody>
<tr>
<td>S_IXOTH</td>
<td>0x0001</td>
<td>Execute permission for all others.</td>
</tr>
<tr>
<td>S_IWOTH</td>
<td>0x0002</td>
<td>Write permission for all others.</td>
</tr>
<tr>
<td>S_IROTH</td>
<td>0x0004</td>
<td>Read permission for all others.</td>
</tr>
<tr>
<td>S_IXGRP</td>
<td>0x0008</td>
<td>Execute permission for processes with same group ID.</td>
</tr>
<tr>
<td>S_IWGRP</td>
<td>0x0010</td>
<td>Write permission for processes with same group ID.</td>
</tr>
<tr>
<td>S_IRGRP</td>
<td>0x0020</td>
<td>Read permission for processes with same group ID.</td>
</tr>
<tr>
<td>S_IXUSR</td>
<td>0x0040</td>
<td>Execute permission for processes with same user ID.</td>
</tr>
<tr>
<td>S_IWUSR</td>
<td>0x0080</td>
<td>Write permission for processes with same user ID.</td>
</tr>
<tr>
<td>S_IRUSR</td>
<td>0x0100</td>
<td>Read permission for processes with same user ID.</td>
</tr>
<tr>
<td>S_ISVTX</td>
<td>0x0200</td>
<td>Unused</td>
</tr>
<tr>
<td>S_ISGID</td>
<td>0x0400</td>
<td>Alter effective group ID when executing this file.</td>
</tr>
<tr>
<td>S_ISUID</td>
<td>0x0800</td>
<td>Alter effective user ID when executing this file.</td>
</tr>
<tr>
<td>S_IFCHR</td>
<td>0x2000</td>
<td>File is a BIOS special file.</td>
</tr>
<tr>
<td>S_IFDIR</td>
<td>0x4000</td>
<td>File is a directory.</td>
</tr>
<tr>
<td>S_IFREG</td>
<td>0x8000</td>
<td>File is a regular file.</td>
</tr>
<tr>
<td>S_IFIFO</td>
<td>0xA000</td>
<td>File is a FIFO.</td>
</tr>
<tr>
<td>S_IMEM</td>
<td>0xC000</td>
<td>File is a memory region.</td>
</tr>
<tr>
<td>S_IFLNK</td>
<td>0xE000</td>
<td>File is a symbolic link.</td>
</tr>
</tbody>
</table>

If the file is created successfully, the fcookie structure fc should be filled in to represent the newly created file and E_OK should be returned. On an error, an appropriate GEMDOS error code should be returned.

getdev

This function is used by the kernel to identify the device driver that should be used to do file I/O on the file named by fc. The function should return a pointer to the device driver and place a user-defined value in the longword pointed to by devspecial. If the function fails, the function should return and place a negative GEMDOS error code in the longword pointed to by devspecial.

getxattr

This function should fill in the XATTR structure pointed to by xattr with the extended attributes of file fc. If the function succeeds, the routine should return E_OK, otherwise a negative GEMDOS error code should be returned.

chattr

This function is called by the kernel to instruct the file system to change the attributes of file fc to those in attr (with only the low eight bits being significant). The function should return a standard GEMDOS error code on exit.

chown

This function is called by the kernel to instruct the file system to change the file fc’s group and user ownership to gid and uid respectively. The kernel checks access permissions prior to calling this function so the file system does not have to.
<table>
<thead>
<tr>
<th>Function</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>chmode</td>
<td>This function is called by the kernel to instruct the file system to change the access permissions of file fc to those in mode. The mode parameter passed to this function will never contain anything but access permission information (i.e. no file type information will be contained in mode). The call should return a standard GEMDOS error code on exit.</td>
</tr>
<tr>
<td>mkdir</td>
<td>This function should create a new subdirectory called name in directory dir with access permissions of mode. The file system should ensure that directories such as ‘.’ and ‘..’ are created and that a standard GEMDOS error code is returned.</td>
</tr>
<tr>
<td>rmdir</td>
<td>This function should remove the directory whose name is name and whose cookie is pointed to by dir. This call should allow the removal of symbolic links to directories and return a standard GEMDOS error code.</td>
</tr>
<tr>
<td>remove</td>
<td>This function should delete the file named name that resides in directory dir. If more than one 'hard' link to this file exists, then only this link should be destroyed and the file contents should be left untouched. Symbolic links to file fc, however, should be removed. This function should not allow the deletion of directories and should return with a standard GEMDOS error code.</td>
</tr>
<tr>
<td>getname</td>
<td>This function should fill in the buffer pointed to by pathname with as many as PATH_MAX (128) characters of the path name of directory dir expressed relatively to directory relto. If relto and dir point to the same directory, a NULL string should be returned. For example, if relto points to directory “FOO” and dir points to directory “FOO\BAR\SUB” then pathname should be filled in with “BAR\SUB”.</td>
</tr>
<tr>
<td>rename</td>
<td>This function should rename the file oldname which resides in directory olddir to the new name newname which resides in newdir. The file system may choose to support or not support cross-directory renames. The function should return a standard GEMDOS error code. If no renames at all are supported then EINVFN should be returned.</td>
</tr>
<tr>
<td>opendir</td>
<td>This function opens directory dirh for reading. The parameter tosflag is a copy of the flags member of the DIR structure as defined below:</td>
</tr>
</tbody>
</table>

```c
typedef struct dirstruct {
    fcookie fc; /* Directory cookie */
    UID index;  /* Index of current entry */
    UID flags;  /* TOS_SEARCH (1) or 0 */
    char fsstuff[60]; /* File system dependent */
} DIR;
```

If tosflags (dirstruct.flags) contains the mask TOS_SEARCH the file system is responsible for parsing the names into something readable by TOS domain applications. The file system should initialize the index and fsstuff members of dirh and return an appropriate GEMDOS error code. |
| readdir  | This function should read the next filename from directory dirh. The fcookie structure fc should be filled in with the details of this file. If dirh->flags does not contain the mask TOS_SEARCH then the filename should be copied into the buffer pointed to by name. If dirh->flags contains the mask TOS_SEARCH then the first four bytes of name should be treated as a longword and filled in with an index value uniquely identifying the file and the filename should be copied starting at &name[4]. |

In either case, if the filename is longer than namelen, rather than filling in the buffer name, the function should return with ENAMETOOLOONG. If this is the last file in the directory, ENMFIL should be returned, otherwise return E_OK. |
| rewinddir| This function should reset the members of dirh so that any internal pointers point at the first file of directory dirh. This function should return a standard GEMDOS error code. |
| closedir| This function should clear any allocated memory and clean up any structures used by the search on dirh. This function should return a standard GEMDOS error code. |


### MiNT Interprocess Communication

**Pipelines**

A pipeline is a special file used for data communication in which the data being read or written is kept in memory. Pipes are created by `Fcreate()`’ing a file in the special directory ‘U:\PIPE’. A process which initially opens a pipe is considered the ‘server.’ Processes writing to or reading from the open pipe are called ‘clients.’ Both servers and clients may read to and write from the pipe.

`Fcreate()`’s `attr` byte takes on a special meaning with pipes as follows:

---

<table>
<thead>
<tr>
<th>Function</th>
<th>Description</th>
</tr>
</thead>
</table>
| `pathconf` | This function should return information about the directory `dir` based on mode `mode`. For mode values and return values, see `Dpathconf()`.
| `dfree` | This function should return free space information about the drive directory `dir` is located on. The format of the buffer pointed to by `buf` is the same as is used by `Dfree()`. This function should return a standard GEMDOS error code.
| `writelabel` | This function is used to change the volume name of a drive which contains the directory `dir`. The new name `name` should be used to write (or rename the volume label). If the write is actually an attempt to rename the label and the file system does not support this function then `EACCSDN` should be returned. If the file system does not support the concept of volume labels then `EINVFN` should be returned. Otherwise, a return value of `E_OK` is appropriate.
| `readlabel` | This function should copy the volume label name of the drive on which directory `dir` is contained in the buffer `name`. If `namelen` is less than the size of the volume name, `ENAMETOOLONG` should be returned. If the concept of volume names is not supported by the file system, `EINVFN` should be returned. If no volume name was ever created, `EFILNF` should be returned. Upon successful error of the call, `E_OK` should be returned.
| `symlink` | This function should create a symbolic link in directory `dir` named `name`. The symbolic link should contain the `NULL` terminated string in `to`. If the file system does not support symbolic links it should return `EINVFN`, otherwise a standard GEMDOS error code should be returned.
| `readlink` | This function should copy the contents of symbolic link `file` into buffer `buf`. If the length of the contents of the symbolic link is greater than `buflen`, `ENAMETOOLONG` should be returned. If the file system does not support symbolic links, `EINVFN` should be returned. In all other cases, a standard GEMDOS error code should be returned.
| `hardlink` | This function should create a ‘hard’ link called `toname` residing in `todir` from the file named `fromname` residing in `fromdir`. If the file system does not support hard links, `EINVFN` should be returned. Otherwise, a standard GEMDOS error code should be returned.
| `fscntl` | This function performs a file system specific function on a file whose name is `name` that resides in directory `dir`. The `cmd` and `arg` functions parallel those of `Dcntl()`. In most cases, this function should simply return `EINVFN`. If your file system wishes to expose special features to the user through `Dcntl()` then your file system should handle them here as it sees fit.
| `dskchng` | This function is used by the kernel to confirm a ‘media change’ state reported by `Mediach()`. If the file system agrees that a media change has taken place, it should invalidate any appropriate buffers, free any allocated memory associated with the device, and return 1. The kernel will then invalidate any open files and relog the drive with the `root()` functions of each installed file system.

If a media change has not taken place, simply return a value of 0.

| `zero` | This member is reserved for future expansion and must be set to 0L. |
MiNT Interprocess Communication – 2.27

<table>
<thead>
<tr>
<th>Name</th>
<th>Bit</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>FA_UNIDIR</td>
<td>0x01</td>
<td>If this bit is set, the pipe will be unidirectional (the server can only write, the client can only read).</td>
</tr>
<tr>
<td>FA_SOFTPIPE</td>
<td>0x02</td>
<td>Setting this bit causes reads when no one is writing to return EOF and writes when no one is reading to raise the signal SIGPIPE.</td>
</tr>
<tr>
<td>FA_TTY</td>
<td>0x04</td>
<td>Setting this bit will make the pipe a pseudo-TTY, i.e. any characters written by the server will be interpreted (CTRL-C will cause a SIGINT signal to be generated to all clients).</td>
</tr>
</tbody>
</table>

Fpipe() can also be used to create pipes quickly with the MiNT kernel resolving any name conflicts. A pipe is deleted when all processes that had obtained a handle to it Fclose() it.

A single process may serve as both the client and the server if it maintains two handles (one obtained from Fopen() and one from Fcreate() ). In addition, child processes of the server may inherit the file handle, and thus the server end of the pipe.

A special system call, Salert(), sends a string to a pipe called ‘U:\PIPE\ALERT’. If a handler is present that reads from this pipe, an alert with the text string will be displayed.

**Signals**

Signals are messages sent to a process that interrupt normal program flow in a way that may be defined by the receiving application. Signals are sent to a process with the function Pkill(). The call is named Pkill() because the default action for most signals is the termination of the process. If a process expects to receive signals it should use Psignal(), Psigsetmask(), Psigblock(), or Psigaction() to modify that behavior by installing a handler routine, ignoring the signal, or blocking the signal completely.

Signal handlers should return by executing a 680x0 RTS instruction or by calling Psigreturn().

Current signals sent and recognized by MiNT processes are as follows:

<table>
<thead>
<tr>
<th>Signal</th>
<th>Number</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>SIGNED</td>
<td>0</td>
<td>This signal is actually a dead signal since it has no effect and is never delivered. Its only purpose is to determine if a child process has exited. A Pkill() call with this signal number will return successfully if the process is still running or fail if not.</td>
</tr>
<tr>
<td>SIGHUP</td>
<td>1</td>
<td>This signal indicates that the terminal connected to the process is no longer valid. This signal is sent by window managers to processes when the user has closed your window. The default action for this signal is to kill the process.</td>
</tr>
<tr>
<td>SIGINT</td>
<td>2</td>
<td>This signal indicates that the user has interrupted the process with CTRL-C. The default action for this signal is to kill the process.</td>
</tr>
<tr>
<td>SIGNED</td>
<td>3</td>
<td>This signal is sent when the user presses CTRL-. The default action for this signal is to kill the process.</td>
</tr>
<tr>
<td>Signal</td>
<td>Number</td>
<td>Description</td>
</tr>
<tr>
<td>--------</td>
<td>--------</td>
<td>-------------</td>
</tr>
<tr>
<td>SIGILL</td>
<td>4</td>
<td>This signal is sent after a 680x0 Illegal Instruction Exception has occurred. The default action for this signal is to kill the process. Catching this signal is un recommended.</td>
</tr>
<tr>
<td>SIGTRAP</td>
<td>5</td>
<td>This signal is sent after each instruction is executed when the system is in single-step trace mode. Debuggers should catch this signal, other processes should not.</td>
</tr>
<tr>
<td>SIGABRT</td>
<td>6</td>
<td>This signal is sent when something has gone wrong internally and the program should be aborted immediately. The default action for this signal is to kill the process. It is un recommended that you catch this signal.</td>
</tr>
<tr>
<td>SIGPRIV</td>
<td>7</td>
<td>This signal is sent to a process that attempts to execute an instruction that may only be executed in supervisor mode while in user mode. The default action for this signal is to kill the process.</td>
</tr>
<tr>
<td>SIGFPE</td>
<td>8</td>
<td>This signal is sent when a division by 0 or floating-point exception occurs. The default action for this signal is to kill the process.</td>
</tr>
<tr>
<td>SIGKILL</td>
<td>9</td>
<td>This signal forcibly kills the process. There is no way to catch or ignore this signal.</td>
</tr>
<tr>
<td>SIGBUS</td>
<td>10</td>
<td>This signal is sent when a 680x0 Bus Error Exception occurs. The default action for this signal is to kill the process.</td>
</tr>
<tr>
<td>SIGSEGV</td>
<td>11</td>
<td>This signal is sent when a 680x0 Address Error Exception occurs. The default action for this signal is to kill the process.</td>
</tr>
<tr>
<td>SIGSYS</td>
<td>12</td>
<td>This signal is sent when an argument to a system call is bad or out of range and the call doesn’t have a way to report errors. For instance, <code>Super(0L)</code> will send this signal when already in supervisor mode. The default action for this signal is to kill the process.</td>
</tr>
<tr>
<td>SIGPIPE</td>
<td>13</td>
<td>This signal is sent when a pipe you were writing to has no readers. The default action for this signal is to kill the process.</td>
</tr>
<tr>
<td>SIGALRM</td>
<td>14</td>
<td>This signal is sent when an alarm sent by <code>Talarm()</code> is triggered. The default action for this signal is to kill the process.</td>
</tr>
<tr>
<td>SIGTERM</td>
<td>15</td>
<td>This signal indicates a ‘polite’ request for the process to cleanup &amp; exit. This signal is sent when a process is dragged to the trashcan on the desktop. The default action for this signal is to kill the process.</td>
</tr>
<tr>
<td>SIGSTOP</td>
<td>17</td>
<td>This signal is sent to a process to suspend it. It cannot be caught, blocked, or ignored. This signal is usually used by debuggers.</td>
</tr>
<tr>
<td>SIGTSTP</td>
<td>18</td>
<td>This signal is sent when the user presses CTRL-Z requesting that the process suspend itself. The default action for this signal is to suspend the process until a <code>SIGCONT</code> signal is caught.</td>
</tr>
<tr>
<td>SIGCONT</td>
<td>19</td>
<td>This signal is sent to restart a process stopped with <code>SIGSTOP</code> or <code>SIGTSTP</code>. The default action for this signal is to resume the process.</td>
</tr>
<tr>
<td>Signal</td>
<td>Number</td>
<td>Description</td>
</tr>
<tr>
<td>----------</td>
<td>--------</td>
<td>-------------</td>
</tr>
<tr>
<td>SIGCHLD</td>
<td>20</td>
<td>This signal is sent when a child process has exited or has been suspended. As a default, this signal causes no action.</td>
</tr>
<tr>
<td>SIGTTIN</td>
<td>21</td>
<td>This signal is sent when a process attempts to read from a terminal in a process group other than its own. The default action is to suspend the process.</td>
</tr>
<tr>
<td>SIGTTOU</td>
<td>22</td>
<td>This signal is sent when a process attempts to write to a terminal in a process group other than its own. The default action is to suspend the process.</td>
</tr>
<tr>
<td>SIGIO</td>
<td>23</td>
<td>This signal is sent to indicate that I/O is possible on a file descriptor. The default action for this signal is to kill the process.</td>
</tr>
<tr>
<td>SIGXCPU</td>
<td>24</td>
<td>This signal is sent when the maximum CPU time allocated to a process has been used. This signal will continue to be sent to a process until it exits. The default action for this signal is to kill the process.</td>
</tr>
<tr>
<td>SIGXFSZ</td>
<td>25</td>
<td>This signal is sent to a process when it attempts to modify a file in a way that causes it to exceed the processes’ maximum file size limit. The default action for this signal is to kill the process.</td>
</tr>
<tr>
<td>SIGVTALRM</td>
<td>26</td>
<td>This signal is sent to a process which has exceeded its maximum time limit. The default action for this signal is to kill the process.</td>
</tr>
<tr>
<td>SIGPROF</td>
<td>27</td>
<td>This signal is sent to a process to indicate that its profiling time has expired. The default action for this signal is to kill the process.</td>
</tr>
<tr>
<td>SIGWINCH</td>
<td>28</td>
<td>This signal indicates that the size of the window in which your process was running has changed. If the process cares about window size it can use Fcntl() to obtain the new size. The default action for this signal is to do nothing.</td>
</tr>
<tr>
<td>SIGUSR1</td>
<td>29</td>
<td>This signal is one of two user-defined signals. The default action for this signal is to kill the process.</td>
</tr>
<tr>
<td>SIGUSR2</td>
<td>30</td>
<td>This signal is one of two user-defined signals. The default action for this signal is to kill the process.</td>
</tr>
</tbody>
</table>

Memory Sharing

With the enforcement of memory protection under MultiTOS, the availability of shared memory blocks is important for applications wishing to share blocks of memory. A shared memory block is opened by Fcreate()’ing a file in the directory ‘U:\SHM’. After that, a memory block allocated with Malloc() or Mxalloc() may be attached to the file with Fcntl( handle, memptr, SHMSETBLK ).

Any process which uses Fopen() and Fcntl() with a parameter of SHMGETBLK can now read that memory as if it were a disk file. After a process obtains the address of a shared memory block with SHMGETBLK the memory is guaranteed to be valid until it calls Mfree() on that block even if it Fclose()’s the original file handle.

Note that the address returned by Fcntl() may be different in different processes. Because of this, data in shared memory blocks should not contain absolute pointers.
When a process is finished with a shared memory block, it should \texttt{Mfree()} the address returned by the \texttt{Fcntl()} call. A shared memory block is also deleted by the \texttt{Fdelete()} call if the file is currently unopened by any other processes.

**Other Methods of Communication**

\texttt{Psemaphore()} can be used to create named flags which can synchronize the behavior of multiple applications (if adhered to). \texttt{Pmsg()} is used to send simple messages between two processes.

**MiNT Debugging**

\texttt{MiNT} allows a processes’ TEXT, DATA, and BSS space to be read and written to with standard \texttt{GEMDOS} file commands by opening the process on ‘U:\PROC\’ A file named “TEST” with a \texttt{MiNT} identification of 10 could be opened by specifying the name as ‘U:\PROC\TEST.10’ or ‘U:\PROC\10’. Opening a file to ‘U:\PROC\-1’ will open your own process whereas opening a file to ‘U:\PROC\-2’ will open your parent process.

**Tracing**

A process may be setup for tracing in a number of ways. A child process may be started in trace mode by OR’ing 0x8000 with the \texttt{Pexec()} mode number in a \texttt{Pexec()} call. A process may also trace another process by opening it as described above and using the \texttt{Fcntl()} call with a parameter of \texttt{PTRACESFLAGS}. Processes may start tracing on themselves if their parent is prepared for it.

When in trace mode, the process being traced halts and generates a \texttt{SIGCHLD} signal to its tracer after every instruction (unless this action is modified). The example below shows how to obtain the process ID of the stopped child and the signal that caused the child to stop.

```c
#define WIFSTOPPED(x) (((int)((x) & 0xFF)==0x7F) && ((int)(((x)>>8)&0xFF)!=0))
#define WSTOPSIG(x) ((int)(((x)>>8) & 0xFF))

void HandleSignal( LONG signo )
{
    WORD pid;
    WORD childsignal;
    ULONG r;

    if( signo == SIGCHLD )
    {
        r = Pwait3( 0x2, 0L );
        if( WIFSTOPPED( r ) )
        {
            pid = r >> 16;
            childsignal = WSTOPSIG( r );
        }
    }
}
```

After reception of this signal, the child process may be restarted with \texttt{Fcntl()} using either the \texttt{PTRACEGO}, \texttt{PTRACEFLOW}, or \texttt{PTRACESTEP} commands. Setting \texttt{PTRACEFLOW} or
PTRACESTEP causes a SIGTRAP signal to be raised on the next program flow change (ex: BRA or JMP) or the instruction respectively.

Modifying the Process Context
A processes’ registers may be modified during tracing using the method as illustrated in the following example:

```c
struct context
{
    LONG regs[15];         // Registers d0-d7, a0-a6
    LONG usp;              // User stack pointer
    WORD sr;               // Status register
    LONG pc;               // Program counter
    LONG ssp;              // Supervisor stack pointer
    LONG tvec;             // GEMDOS terminate vector
    char fstate[216];      // Internal FPU state
    LONG fregs[3*8];       // Registers FP0-FP7
    LONG fctrl[3];         // Registers FPCR/FPSR/FPIAR

    // More undocumented fields exist here
} c;

void ModifyContext( LONG handle )
{
    LONG curprocaddr, ctxtsize;

    Fcntl( handle, &curprocaddr, PPROCADDR );
    Fcntl( handle, &ctxtsize, PCTXTSIZE );

    curprocaddr -= 2 * ctxtsize;

    Fseek( curprocaddr, handle, SEEK_SET );
    Fread( handle, (LONG)sizeof(struct context), &c );

    /* Modify context c here */

    Fseek( curprocaddr, handle, SEEK_SET );
    Fwrite( handle, (LONG)sizeof(struct context), &c );
}
```

MiNT Debugging Keys
MiNT may be programmed to output special debugging messages to the debugging device through the use of special system keys. The supported system keys are shown in the table below:

<table>
<thead>
<tr>
<th>Key Combination</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>CTRL-ALT-F1</td>
<td>Increase the system debugging level by one.</td>
</tr>
<tr>
<td>CTRL-ALT-F2</td>
<td>Decrease the system debugging level by one.</td>
</tr>
<tr>
<td>CTRL-ALT-F3</td>
<td>Cycle the BIOS output device number used for system debugging messages. This key cycles BIOS devices in the order 1-6-7-8-9-2.</td>
</tr>
<tr>
<td>CTRL-ALT-F4</td>
<td>Restore debugging output to the console device.</td>
</tr>
<tr>
<td>CTRL-ALT-F5</td>
<td>Output a memory usage map to the debugging device.</td>
</tr>
<tr>
<td>CTRL-ALT-F6</td>
<td>Output a list of all system processes to the debugging device.</td>
</tr>
</tbody>
</table>
The Atari Compendium

CTRL-ALT-F7

**CTRL-ALT-F7**

Toggles debug ‘logging’ off and on. When debug logging is on, a 50-line buffer is maintained which contains recent debugging messages. Each time a new debugging message is output, the entire 50 line buffer is output as well.

CTRL-ALT-F8

**CTRL-ALT-F8**

Outputs the 50-line debug log to the debugging device.

CTRL-ALT-F9

**CTRL-ALT-F9**

Outputs the system memory map to the debugging device. The memory protection flags of each page are shown.

CTRL-ALT-F10

**CTRL-ALT-F10**

Outputs an extended system memory map to the debugging device. The memory protection status, owner’s PID, and format of each memory block are output to the debugging device.

CTRL-ALT-F1 and CTRL-ALT-F2 alter the current system debugging level. **MiNT** supports four debugging levels as follows:

<table>
<thead>
<tr>
<th>Level</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>Only fatal OS errors are reported to the debugging device (this is the default mode).</td>
</tr>
<tr>
<td>1</td>
<td>Processor exceptions are output to the debugging device.</td>
</tr>
<tr>
<td>2</td>
<td>Processor exceptions and failed system calls are output to the debugging device.</td>
</tr>
<tr>
<td>3</td>
<td>Constant <strong>MiNT</strong> status reports, processor exceptions, and failed system calls are output to the debugging device.</td>
</tr>
</tbody>
</table>

**The MINT.CNF File**

**MultiTOS** looks for an ASCII text file upon bootup called ‘MINT.CNF’ which may be used to execute commands or set **MiNT** variables. The following table illustrates what commands are recognized in the ‘MINT.CNF’ file:

<table>
<thead>
<tr>
<th>Command</th>
<th>Example</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>cd</td>
<td>cd c:\multitos</td>
<td>Change the GEMDOS working directory.</td>
</tr>
<tr>
<td>echo</td>
<td>echo &quot;Atari Computer Booting...&quot;</td>
<td>Echo a string to the screen.</td>
</tr>
<tr>
<td>ren</td>
<td>ren c:\test.prg c:\test.app</td>
<td>Rename a file.</td>
</tr>
<tr>
<td>sln</td>
<td>sln c:\level1\level2\level3 u:\deep</td>
<td>Create a symbolic link on drive ‘U’.</td>
</tr>
<tr>
<td>alias</td>
<td>alias x: u:\proc</td>
<td>Create an alias drive.</td>
</tr>
<tr>
<td>exec</td>
<td>exec c:\sam.prg</td>
<td>Execute a program.</td>
</tr>
</tbody>
</table>

The following **MiNT** variables may be set in the ‘MINT.CNF’ file:
GEMDOS Character Functions

GEMDOS provides a number of functions to communicate on a character basis with the default system devices. Because of irregularities with these calls in some TOS versions, usage of the BIOS functions is usually recommended instead (the BIOS does not support redirection, however).

The GEMDOS character functions are illustrated in the table below:

<table>
<thead>
<tr>
<th>Variable</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>INIT</td>
<td>Execute the named TOS program. For example:</td>
</tr>
<tr>
<td></td>
<td>INIT=c:\multitos\sam.prg</td>
</tr>
<tr>
<td>GEM</td>
<td>Execute the named GEM program. For example:</td>
</tr>
<tr>
<td></td>
<td>GEM=c:\multitos\miniwin.app</td>
</tr>
<tr>
<td>CON</td>
<td>Redirect console input and output to the named file. For example:</td>
</tr>
<tr>
<td></td>
<td>CON=u:\dev\modem1</td>
</tr>
<tr>
<td>PRN</td>
<td>Redirect printer output to the named file. For example:</td>
</tr>
<tr>
<td></td>
<td>PRN=c:\spool.txt</td>
</tr>
<tr>
<td>DEBUG_LEVEL</td>
<td>Set the MiNT debugging level (default is 0). For example:</td>
</tr>
<tr>
<td></td>
<td>DEBUG_LEVEL=1</td>
</tr>
<tr>
<td>DEBUG_DEVNO</td>
<td>Set the BIOS device number that MiNT will send debugging messages to. For example:</td>
</tr>
<tr>
<td></td>
<td>DEBUG_DEVNO=1</td>
</tr>
<tr>
<td>SLICES</td>
<td>Set the number of 20ms time slices given to an application at a time (the default is 2). For example:</td>
</tr>
<tr>
<td></td>
<td>SLICES=3</td>
</tr>
<tr>
<td>MAXMEM</td>
<td>Set the maximum amount of memory (in kilobytes) any application can be allocated (the default is unlimited). For example:</td>
</tr>
<tr>
<td></td>
<td>MAXMEM=8192</td>
</tr>
<tr>
<td>BIOSBUF</td>
<td>Enable/Disable Bconout() optimizations. The parameter should be ‘Y’ to enable or ‘N’ to disable these optimizations. For example:</td>
</tr>
<tr>
<td></td>
<td>BIOSBUF=Y</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Device:</th>
<th>Input</th>
<th>Output</th>
<th>Status</th>
</tr>
</thead>
<tbody>
<tr>
<td>con:</td>
<td>Cconin() - Character</td>
<td>Cconout() - Character</td>
<td>Cconis() - Input</td>
</tr>
<tr>
<td></td>
<td>Cnecin() - No Echo</td>
<td>Cconws() - String</td>
<td>Cconos() - Output</td>
</tr>
<tr>
<td>prn:</td>
<td>None</td>
<td>Cprnout()</td>
<td>Cprnos()</td>
</tr>
</tbody>
</table>
### GEMDOS Time & Date Functions

GEMDOS provides four functions for the manipulation of time. **Tsetdate()** and **Tsettime()** set the date and time respectively. **Tgetdate()** and **Tgettime()** get the date and time respectively.

As of TOS 1.02, the GEMDOS time functions also update the BIOS time.

### GEMDOS Function Calling Procedure

GEMDOS system functions are called via the TRAP #1 exception. Function arguments are pushed onto the current stack in reverse order followed by the function opcode. The calling application is responsible for correctly resetting the stack pointer after the call.

GEMDOS may utilize registers D0-D2 and A0-A2 as scratch registers and their contents should not be depended upon at the completion of a call. In addition, the function opcode placed on the stack will be modified.

The following example for **Super()** illustrates calling GEMDOS from assembly language:

```assembly
clr.l -(sp)        ; Save return address
move.w #$20,-(sp) ; Function arguments
trap #1            ; Call GEMDOS
addq.l #4,sp       ; Restore return address
```

‘C’ compilers often provide a reusable interface to GEMDOS that allows new GEMDOS calls to be added with a macro as in the following example:

```c
#define Super( a ) gemdos( 0x20, a )
```

The gemdos() function used in the above macro can be written in assembly language as follows:

```assembly
.globl _gemdos
.text
_gemdos:
move.l (sp)+, t1sav ; Save return address
trap #1            ; Call GEMDOS
move.l t1sav,-(sp) ; Restore return address
rts
.bss
t1sav: ds.l 1       ; Return address storage
.end
```

---

The Atari Compendium
GEMDOS is not guaranteed to be re-entrant and therefore should not be called from an interrupt handler.
Cauxin()

WORD Cauxin( VOID )

Cauxin() waits for the next available data byte from GEMDOS handle 2 (normally device ‘aux:’) and when available, returns it in the low byte of the returned WORD.

OPCODE 3 (0x03)

AVAILABILITY All GEMDOS versions.

BINDING

move.w #$3,-(sp)
trap #1
addq.l #2,sp

RETURN VALUE The WORD value contains the retrieved byte in the lower eight bits. The contents of the upper 8 bits are currently undefined.

CAVEATS This function can cause flow control problems.

When using this function while its handle is redirected, an end-of-file condition will hang the system. GEMDOS version 0.30 and all MiNT versions correct this bug. MiNT returns MiNT_EOF (0xFF1A) when the end-of-file is reached.

In addition, if this handle is redirected to something other than ‘aux:’, an end-of-file will hang the system. Besides these known bugs, this function is used by many ‘C’ compilers to redirect standard error messages. It is therefore advisable to use Bconin() instead.

SEE ALSO Cauxis(), Cauxout(), Bconin()

Cauxis()

WORD Cauxis( VOID )

Cauxis() indicates whether GEMDOS handle 2 (normally device ‘aux:’) has at least one character waiting.

OPCODE 18 (0x12)

AVAILABILITY All GEMDOS versions.

BINDING

move.w #$12,-(sp)
The return value will be **DEV_READY** (-1) if at least one character is available for reading or **DEV_BUSY** (0) if not.

**Caveats**

When using this function while its handle is redirected, an end-of-file condition will hang the system. **GEMDOS** version 0.30 and all **MiNT** versions correct this bug. **MiNT** returns **MINT_EOF** (0xFF1A) when the end-of-file is reached.

In addition, some ‘C’ compilers use this handle as a standard error device. It is therefore advisable to use **Bconstat()**.

**See Also**

**Cauxin()**, **Cauxout()**, **Cauxos()**, **Bconstat()**

---

### Cauxos()

**WORD Cauxos( VOID )**

**Cauxos()** indicated whether **GEMDOS** handle 2 (normally device ‘aux:’) is ready to receive characters.

**OPCODE**

19 (0x13)

**Availability**

All **GEMDOS** versions

**Binding**

```
move.w  $13,-(sp)
trap    #1
addq.l  #2,sp
```

**RETURN VALUE**

A value of **DEV_READY** (-1) is returned if the output device is ready to receive characters or **DEV_BUSY** (0) if it is not.

**Caveats**

This function actually returns the status of whatever device **GEMDOS** handle 2 is redirected to. In addition, some ‘C’ compilers use this handle as a standard error device. It is therefore recommended that **Bconstat()** be used instead.

**See Also**

**Cauxin()**, **Cauxis()**, **Cauxout()**, **Bconstat()**.
Cauxout()

VOID Cauxout( ch )
WORD ch;

Cauxout() outputs a character to GEMDOS handle 2, normally the ‘aux:’ device.

OPCODE
4 (0x04)

AVAILABILITY
All GEMDOS versions.

PARAMETERS
ch is a WORD value, however, only the lower eight bits are sent. The upper eight bits must be 0.

BINDING
move.w #ch,-(sp)
movw.w #4,-(sp)
trap #1
addq.l #4,sp

CAVEATS
This function can cause flow control to fail when GEMDOS handle 2 is directed to ‘aux:’.

In addition, some ‘C’ compilers use this function as a standard error device. It is therefore recommended that Bconout() be used in place of this function.

SEE ALSO
Cauxin(), Cauxis(), Cauxos(), Bconout()

Cconin()

LONG Cconin( VOID )

Cconin() reads a character (waiting until one is available) from GEMDOS handle 0 (normally ‘con:’).

OPCODE
1 (0x01)

AVAILABILITY
All GEMDOS versions.

BINDING
move.w #1,-(sp)
trap #1
addq.l #2,sp

RETURN VALUE
The LONG value returned is a bit array arranged as follows:
The ASCII code of the character will be 0 if a non-ascii keyboard key is struck.

**Caveats**

When using this function while its handle is redirected, an end-of-file condition will hang the system. GEMDOS version 0.30 and all MiNT versions correct this bug. MiNT returns MINT_EOF (0xFF1A) when the end-of-file is reached.

**Comments**

The shift key status will only be returned when bit 3 of the system variable `conterm` (char *(0x484)) is set. This is normally not enabled.

If the handle has been redirected, the inputted character will appear in the lower 8 bits of the return value.

**See Also**

Cconis(), Cconout(), Cconrs(), Cncin(), Crawin(), Bconin()

---

**Cconis()**

*WORD Cconis( VOID )*

Cconis() verifies that a character is waiting to be read from GEMDOS handle 0 (normally ‘con.’).

**Opcode**

11 (0xB)

**Availability**

All GEMDOS versions.

**Binding**

```
move.w #$0B,-(sp)
trap #1
addq.l #2,sp
```

**Return Value**

Cconis() returns a DEV_READY (-1) if a character is available or DEV_BUSY (0) if not.

**See Also**

Cconin(), Bconstat()
Cconos()

WORD Cconos( VOID )

Cconos() checks to see whether a character may be output to GEMDOS handle 1 (normally ‘con:’).

Opcode 16 (0x10)

Availability All GEMDOS versions.

Binding

move.w #$10,-(sp)
trap #1
addq.l #2,sp

ReturnValue This function returns DEV_READY (-1) if at least one character may be sent or DEV_BUSY (0) if not.

See Also Cconout(), Bcostat()

Cconout()

VOID Cconout( ch )
WORD ch;

Cconout() outputs one character via GEMDOS handle 1 (normally ‘con:’).

Opcode 2 (0x02)

Availability All GEMDOS versions.

Parameters ch is a WORD value, however, only the lower eight bits are sent through the output stream. The upper eight bits must be 0.

Binding

move.w ch,-(sp)
move.w #2,-(sp)
trap #1
addq.l #4,sp

Caveats With GEMDOS versions below 0.15, this handle should not be redirected to a write-only device as the call attempts to read from the output stream to process special keys.

Comments No line feed translation is done at the time of output. To start a new line, ASCII 13
and ASCII 10 must both be sent.

SEE ALSO Cconin(), Bconout()

Cconrs()

VOID Cconrs( str )
char *str;

Cconrs() reads a string from the standard input stream (GEMDOS handle 0) and echoes it to the standard output stream (GEMDOS handle 1).

OPCODE 10 (0x0A)

AVAILABILITY All GEMDOS versions.

PARAMETERS str should be a character pointer large enough to hold the inputted string. On function entry, str[0] should be equal to the maximum number of characters to read.

BINDING
pea str
move.w #$0A,-(sp)
trap #1
addq.l #6,sp

RETURN VALUE On return, the string buffer passed as a parameter will be filled in with the inputted characters. str[1] will contain the actual number of characters in the buffer. (char *) &str[2] is the pointer to the start of the actual string in memory.

Cconrs() will not terminate unless CTRL-C is pressed, the buffer is full or either RETURN or CTRL-J is pressed.

CAVEATS GEMDOS versions below 0.15 echoes the input to the console even if output has been redirected elsewhere.

COMMENTS The string Cconrs() creates is not null-terminated. The following keys are processed by the function:

<table>
<thead>
<tr>
<th>Key</th>
<th>Translation</th>
</tr>
</thead>
<tbody>
<tr>
<td>RETURN</td>
<td>End of input. Do not place RETURN in in buffer.</td>
</tr>
<tr>
<td>CTRL-J</td>
<td>End of line. Do not place CTRL-J in buffer.</td>
</tr>
<tr>
<td>CTRL-H</td>
<td>Kill last character.</td>
</tr>
<tr>
<td>DELETE</td>
<td>Kill last character.</td>
</tr>
<tr>
<td>CTRL-U</td>
<td>Echo input line and start over.</td>
</tr>
</tbody>
</table>
When the input stream is redirected, `Cconrs()` returns 0 in \texttt{str[1]} when the end-of-file marker is reached.

**SEE ALSO**  
Cconin(), Cconws()

---

### Cconws()

**VOID Cconws( str )**  
char *\textit{str};

\textbf{Cconws()} writes a string to \texttt{GEMDOS} handle 1 (normally ‘con:').

**OPCODE**  
9 (0x09)

**AVAILABILITY**  
All \texttt{GEMDOS} versions.

**PARAMETERS**  
\textit{str} is a pointer to a null-terminated character string to be written to the output stream.

**BINDING**

\begin{verbatim}
pea str
move.w #$09,-(sp)
trap 1
addq.l #6,sp
\end{verbatim}

**CAVEATS**  
With \texttt{GEMDOS} versions below 0.15, this handle should not be redirected to a write-only device as the call attempts to read from the output stream to process special keys.

**COMMENTS**  
No line feed translation is performed on outputted characters so both an ASCII 13 and ASCII 10 must be sent to force a new line. In addition, the system checks for special keys so a CTRL-C embedded in the string will terminate the process.

**SEE ALSO**  
Cconout(), Cconrs()
Cnecin()

WORD Cnecin( VOID )

Cnecin() is exactly the same as Cconin() except that the character fetched from the input stream is not echoed.

OPCODE  8 (0x08)

AVAILABILITY All GEMDOS versions.

PARAMETERS None.

BINDING

\[
\begin{align*}
\text{move.w} & \quad 8, -(sp) \\
\text{trap} & \quad 1 \\
\text{addq.l} & \quad 2, sp
\end{align*}
\]

RETURN VALUE The LONG value returned is a bit array arranged as follows:

<table>
<thead>
<tr>
<th>Bits 31-24</th>
<th>Bits 23-16</th>
<th>Bits 15-8</th>
<th>Bits 7-0</th>
</tr>
</thead>
<tbody>
<tr>
<td>Shift key status (see below)</td>
<td>Keyboard scancode</td>
<td>Unused (0)</td>
<td>ASCII code of character</td>
</tr>
</tbody>
</table>

The ASCII code of the character will be 0 if a non-ascii keyboard key is struck.

CAVEATS When using this function while its handle is redirected, an end-of-file condition will hang the system. GEMDOS version 0.30 and all MiNT versions correct this bug. MiNT returns MINT_EOF (0xFF1A) when the end-of-file is reached.

COMMENTS The shift key status will only be returned when bit 3 of the system variable conterm (char *(0x484)) is set. This is normally not enabled.

If the handle has been redirected, the inputted character will appear in the lower 8 bits of the return value.

SEE ALSO Cconin(), Bconin()

Cprnos()

WORD Cprnos( VOID )

Cprnos() returns the status of GEMDOS handle 3 (normally ‘prn.’).

OPCODE  17 (0x11)

THE ATARI COMPENDIUM
**Availability**

All GEMDOS versions.

**Parameters**

None.

**Binding**

```
move.w #$11,-(sp)
trap #1
addq.l #2,sp
```

**Return Value**

`Cprnout()` returns a `DEV_READY` (-1) if the output stream is ready to receive a character or `DEV_BUSY` (0) if not.

**See Also**

`Cprnout()`, `Bcostat()`

---

**Cprnout()**

WORD `Cprnout(ch)`

WORD `ch`;

`Cprnout()` sends one character to GEMDOS handle 3 (normally ‘prn:’).

**Opcode**

5 (0x05)

**Availability**

All GEMDOS versions.

**Parameters**

`ch` is a WORD value, however, only the lower 8 bits are sent to the output stream. The upper eight bits should be 0.

**Binding**

```
move.w ch,-(sp)
move.w #$5,-(sp)
trap #1
addq.l #4,sp
```

**Return Value**

`Cprnout()` returns a non-zero value if the function successfully wrote the character to the printer or 0 otherwise.

**Comments**

No input translation is performed with this call. Therefore, you must send an ASCII 13 and ASCII 10 to force a new line.

**See Also**

`Bconout()`
Crawcin()

LONG Crawcin( VOID )

*Crawcin()* is similar to *Cconout()* , however it does not process any special keys and does not echo the inputted character.

**OPCODE** 7 (0x07)

**AVAILABILITY** All *GEMDOS* versions.

**BINDING**

```
move.w #$07,-(sp)
trap #1
addq.l #2,sp
```

**RETURN VALUE** The *LONG* value returned is a bit array arranged as follows:

<table>
<thead>
<tr>
<th>Bits 31-24</th>
<th>Bits 23-16</th>
<th>Bits 15-8</th>
<th>Bits 7-0</th>
</tr>
</thead>
<tbody>
<tr>
<td>Shift key status (see below)</td>
<td>Keyboard scancode</td>
<td>Unused (0)</td>
<td>ASCII code of character</td>
</tr>
</tbody>
</table>

The ASCII code of the character will be 0 if a non-ascii keyboard key is struck.

**CAVEATS** When using this function while its handle is redirected, an end-of-file condition will hang the system. *GEMDOS* version 0.30 and all *MiNT* versions correct this bug. *MiNT* returns *MINT_EOF* (0xFF1A) when the end-of-file is reached.

**COMMENTS** The shift key status will only be returned when bit 3 of the system variable *conterm* (char *(0x484)) is set. This is normally not enabled.

If the handle has been redirected, the inputted character will appear in the lower 8 bits of the return value.

Under normal circumstances, when *GEMDOS* handle 0 is being read from, no special system keys, including *CTRL-C*, are checked.

**SEE ALSO** *Cconin()* , *Crawio()* , *Bconin()*
Crawio()

LONG Crawio( ch )
WORD ch;

Crawio() combines console input and output in one function.

Opcode
6 (0x06)

Availability
All GEMDOS versions.

Parameters
ch is a WORD value, however, only the lower eight bits are meaningful and the upper eight bits should be set to 0. If ch is 0x00FF on input, Crawio() returns the character read from GEMDOS handle 0 (normally ‘con:’).

Binding
move.w ch,-(sp)
movw #6,-(sp)
trap #1
addq.l #4,sp

Return Value
If ch is 0x00FF upon entry, Crawio() returns a bit array arranged as follows:

<table>
<thead>
<tr>
<th>Bits 31-24</th>
<th>Bits 23-16</th>
<th>Bits 15-8</th>
<th>Bits 7-0</th>
</tr>
</thead>
<tbody>
<tr>
<td>Shift key status (see below)</td>
<td>Keyboard scancode</td>
<td>Unused (0)</td>
<td>ASCII code of character</td>
</tr>
</tbody>
</table>

The ASCII code of the character will be 0 if a non-ascii keyboard key is struck.

If no character was waiting in the input stream, Crawio() returns a 0.

Caveats
When using this function while its handle is redirected, an end-of-file condition will hang the system. GEMDOS version 0.30 and all MiNT versions correct this bug. MiNT returns MINT_EOF (0xFF1A) when the end-of-file is reached.

Due to the definition of this call it is impossible to write 0x00FF to the output stream or read a zero from this call.

Comments
The shift key status will only be returned when bit 3 of the system variable conterm (char *(0x484)) is set. This is normally not enabled.

If the handle has been redirected, the inputted character will appear in the lower 8 bits of the return value.

Under normal circumstances, when GEMDOS handle 0 is being read from, no special system keys, including CTRL-C, are checked.
Dclosedir()

LONG Dclosedir( dirhandle )
LONG dirhandle;

Dclosedir() closes the specified directory.

OPCODE 299 (0x12B)

AVAILABILITY Available when a ‘MiNT’ cookie with a version of at least 0.90 exists.

PARAMETERS dirhandle is a valid directory handle which specifies the directory to close.

BINDING
move.l dirhandle,-(sp)
move.w #$12B,-(sp)
trap #1
addq.l #6,sp

RETURN VALUE Dclosedir() returns E_OK (0) if successful or EIHNDL (-37) if the directory handle was invalid.

SEE ALSO Dopendir(), Dreaddir(), Drewinddir()

Dcntl()

LONG Dcntl( cmd, name, arg )
WORD cmd;
char *name;
LONG arg;

Dcntl() performs file system specific operations on directories or files.

OPCODE 304 (0x130)

AVAILABILITY Available when a ‘MiNT’ cookie with a version of at least 0.90 exists.

PARAMETERS The only two built-in file systems that support Dcntl() calls are ‘U:\’ and ‘U:\DEV.’ cmd specifies what operation to perform and affects the meaning of name and arg. Valid cmd arguments for ‘U:\’ are
<table>
<thead>
<tr>
<th>cmd</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>FS_INSTALL</strong></td>
<td>This mode installs a new file system. <em>name</em> must be 'U:' and <em>arg</em> should point to a <em>fs_descr</em> structure as follows:</td>
</tr>
<tr>
<td>(0xF001)</td>
<td></td>
</tr>
<tr>
<td></td>
<td>struct <em>fs_descr</em></td>
</tr>
<tr>
<td></td>
<td>{</td>
</tr>
<tr>
<td></td>
<td>FILESYS  *file_system;</td>
</tr>
<tr>
<td></td>
<td>WORD  dev_no;</td>
</tr>
<tr>
<td></td>
<td>LONG  flags;</td>
</tr>
<tr>
<td></td>
<td>LONG  reserved[4];</td>
</tr>
<tr>
<td></td>
<td>};</td>
</tr>
<tr>
<td></td>
<td>If this call is successful, a pointer to a <em>kerinfo</em> structure is returned, otherwise the return value is <strong>NULL</strong>. The file system itself is not accessible until this call is made and it is mounted with <strong>FS_MOUNT</strong>.</td>
</tr>
<tr>
<td><strong>FS_MOUNT</strong></td>
<td>This mode mounts an instance of an installed file system. <em>name</em> should be in the format ‘U:???’ where ‘???’ is the name which the file system will be accessed by. <em>arg</em> should point to the <em>fs_descr</em> structure as above. If the file system is mounted correctly, the <em>dev_no</em> field will be updated to reflect the instance number of the mount (file systems may be mounted multiple times).</td>
</tr>
<tr>
<td>(0xF002)</td>
<td></td>
</tr>
<tr>
<td><strong>FS_UNMOUNT</strong></td>
<td>This mode unmounts an instance of a file system. <em>name</em> is the name of the file system in the form ‘U:???’ where ‘???’ is the name of the file system instance. <em>arg</em> should point to the file system <em>fs_descr</em> structure.</td>
</tr>
<tr>
<td>(0xF003)</td>
<td></td>
</tr>
<tr>
<td><strong>FS_UNINSTALL</strong></td>
<td>This mode uninstalls a file system identified by the <em>fs_descr</em> structure passed in <em>arg</em>. A file system can only be sucessfully uninstalled after all instances of it have been unmounted. <em>name</em> should be ‘U:\’.</td>
</tr>
<tr>
<td>(0xF004)</td>
<td></td>
</tr>
</tbody>
</table>

Valid *cmd* arguments for ‘U:\DEV’ are:
2.52 – GEMDOS Function Reference

<table>
<thead>
<tr>
<th>cmd</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>DEV_INSTALL (0xDE02)</td>
<td>This command attempts to install a device driver. <em>name</em> should be in the format ‘U:\DEV???’ where ‘???’ is the name of the device to install. arg is a pointer to a <em>dev_descr</em> structure as follows:</td>
</tr>
<tr>
<td></td>
<td></td>
</tr>
<tr>
<td>struct dev_descr</td>
<td></td>
</tr>
<tr>
<td></td>
<td>{</td>
</tr>
<tr>
<td></td>
<td>/* Pointer to a device driver structure */ DEVDRV *driver;</td>
</tr>
<tr>
<td></td>
<td>/* Placed in aux field of file cookies */ WORD dinfo;</td>
</tr>
<tr>
<td></td>
<td>/* 0 or O_TTY (0x2000) for TTY */ WORD flags;</td>
</tr>
<tr>
<td></td>
<td>/* If O_TTY is set, points to tty struct */ struct tty *tty;</td>
</tr>
<tr>
<td></td>
<td>/* Reserved for future expansion */ LONG reserved[4];</td>
</tr>
<tr>
<td></td>
<td>}</td>
</tr>
<tr>
<td></td>
<td>If the device is successfully installed, Dcntl() will return a pointer to a <em>kerinfo</em> structure which contains information about the kernel. On failure, Dcntl() will return NULL. See the section on loadable file systems earlier in this chapter for more information.</td>
</tr>
<tr>
<td>DEV_NEWTTY (0xDE00)</td>
<td>This command identifies a BIOS terminal device whose name is <em>name</em> (in the form ‘U:\DEV\DEVNAME’ and whose device number is <em>arg</em>. This call simply makes the MiNT kernel aware of the device. It should have been previously installed by Bconmap(). Any attempt to access the device prior to installing it with the BIOS will result in an EUNDEV (-15) unknown device error. If the device is installed, Dcntl() returns a 0 or positive value. A negative return code signifies failure.</td>
</tr>
<tr>
<td>DEV_NEWBIOS (0xDE01)</td>
<td>This command is the same as DEV_NEWTTY except that it is designed for devices which must have their data transmitted raw (SCSI devices, for example).</td>
</tr>
</tbody>
</table>

**BINDING**

move.l arg, -(sp)
pea name
move.w cmd, -(sp)
move.w $130, -(sp)
trap #1
lea 12(sp), sp

**VERSION NOTES**
The *FS_* group of *cmd* arguments are only available as of MiNT version 1.08.

Due to a bug in MiNT versions 1.08 and below, calling this function with a parameter of DEV_NEWBIOS will not have any effect.

**RETURN VALUE**
See above.

**SEE ALSO**
Bconmap(), Fcntl()
Dcreate()

Dcreate() creates a GEMDOS directory on the specified drive.

**OPCODE**  
57 (0x39)

**AVAILABILITY**  
All GEMDOS versions.

**PARAMETERS**  
`path` is a pointer to a string containing the directory specification of the directory to create. `path` should **not** contain a trailing backslash. Below are some examples and their results.

<table>
<thead>
<tr>
<th>path</th>
<th>Result</th>
</tr>
</thead>
<tbody>
<tr>
<td>C:\ONE\ATARI</td>
<td>Creates a folder named “ATARI” as a subdirectory of “ONE” on drive ‘C:’.</td>
</tr>
<tr>
<td>\ONE\ATARI</td>
<td>Creates a folder named “ATARI” as a subdirectory of “ONE” on the current GEMDOS drive.</td>
</tr>
<tr>
<td>ATARI</td>
<td>Creates a folder named “ATARI” as a subdirectory of the current GEMDOS path on the current GEMDOS drive.</td>
</tr>
</tbody>
</table>

**BINDING**  

`pea path`  
`move.w #39,-(sp)`  
`trap #1`  
`addq.l #6,sp`

**RETURN VALUE**  
Upon return one of three codes may result:

- **E_OK** (0): Operation successful
- **EPTHNF** (-34): Path not found
- **EACCDN** (-36): Access denied

**CAVEATS**  
Prior to GEMDOS version 0.15 GEMDOS did not detect if the creation of a subdirectory failed and could therefore leave partially created directories on disk.

**SEE ALSO**  
Ddelete()
Ddelete()

LONG Ddelete( path )
char *path;

Ddelete() removes a directory on the specified drive.

OPCODE 58 (0x3A)

AVAILABILITY All GEMDOS versions.

PARAMETERS path contains the directory specification of the directory you wish to remove. path should not contain a trailing backslash. For valid examples of path, see the entry for Dcreate().

BINDING

pea path
move.w #$3A,-(sp)
trap #1
addq.l #6,sp

RETURN VALUE Upon return one of four codes may result:

E_OK (0) : Operation successful
EPTHNF (-34): Path not found
EACCDN (-36): Access denied
EINTRN (-65): Internal error

CAVEATS Prior to GEMDOS version 0.15 a Ddelete() on a directory recently created will fail but a second attempt will not.

COMMENTS The directory being deleted must be empty or the call will fail.

SEE ALSO Dcreate()

Dfree()

LONG Dfree( buf, drive )
DISKINFO *buf;
WORD drive;

Dfree() returns information regarding the storage capacity/current usage of the specified drive.

OPCODE 54 (0x36)
**Availability**

All GEMDOS versions.

**Parameters**

`buf` is a DISKINFO pointer which will be filled in on function exit. DISKINFO is defined as:

```c
typedef struct
{
    /* No. of Free Clusters */
    ULONG b_free;

    /* Clusters per Drive */
    ULONG b_total;

    /* Bytes per Sector */
    ULONG b_secsize;

    /* Sectors per Cluster */
    ULONG b_clsize;
} DISKINFO;
```

`drive` is a WORD which indicates the drive to perform the operation on. A value of `DEFAULT_DRIVE` (0) indicates the current GEMDOS drive. A value of 1 indicates drive ‘A:’, a 2 indicates ‘B:’, etc...

**Binding**

```assembly
move.w       drive, -(sp)
pea           info
move.w       #$36, -(sp)
trap          #1
addq.l       #8, sp
```

**Return Value**

Upon return, a value of 0 indicates success. Otherwise, a negative GEMDOS error code is returned.

**Caveats**

Prior to GEMDOS version 0.15 this function is very slow when used on a hard disk.

**Comments**

To obtain the free number of bytes on a disk, use the formula `(info.b_free * info.b_secsize * info.b_clsize)`. To obtain the total number of bytes available on a disk, use the formula `(info.b_total * info.b_secsize * info.b_clsize)`. 
Dgetcwd()

LONG Dgetcwd( path, drv, size )
char *path;
WORD drv, size;

Dgetcwd() returns the processes’ current working directory for the specified drive.

OPCODE 315 (0x13B)

AVAILABILITY Available when a ‘MiNT’ cookie with a version of at least 0.96 exists.

PARAMETERS path is a pointer to a buffer with room for at least size characters into which will be copied the complete working path of drive drv.

BINDING

move.w size,-(sp)
move.w drv,-(sp)
move.w #$13B,-(sp)
trap #1
add.l #10,sp

RETURN VALUE Dgetcwd() returns 0 if successful or a GEMDOS error code otherwise.

SEE ALSO Dgetpath(), Dgetdrv()

Dgetdrv()

WORD Dgetdrv( VOID )

Dgetdrv() returns the current GEMDOS drive code.

OPCODE 25 (0x19)

AVAILABILITY All GEMDOS versions.

BINDING

move.w #$19,-(sp)
trap #1
addq.1 #2,sp

RETURN VALUE Dgetdrv() returns the current GEMDOS drive code. Drive ‘A:’ is represented by a return value of 0, ‘B:’ by a return value of 1, and so on.

SEE ALSO Dsetdrv()
Dgetpath()

LONG Dgetpath (buf, drive)
char *buf;
WORD drive;

Dgetpath() returns the current GEMDOS path specification.

OPCODE
71 (0x47)

AVAILABILITY
All GEMDOS versions.

PARAMETERS
buf is a pointer to a character buffer which will contain the current GEMDOS path specification on function exit. drive is the number of the drive whose path you want returned. drive should be DEFAULT_DRIVE (0) for the current GEMDOS drive, 1 for drive ‘A:’, 2 for drive ‘B:’, and so on.

BINDING
move.w drive,-(sp)
pea buf
trap #1
addq.l #6,sp

RETURN VALUE
Dgetpath() will return one of two errors on function exit:

E_OK (0): Operation successful
EDRIVE (-49): Invalid drive specification

COMMENTS
As there is no way to specify the buffer size to this function you should allow at least 128 bytes of buffer space. This will allow for up to 8 folders deep. Newer file systems (CD-ROM drives) may demand up to 200 bytes.

SEE ALSO
Dsetpath()

Dlock()

LONG Dlock( mode, drv)
WORD mode, drv;

Dlock() locks a BIOS disk device against GEMDOS usage.

OPCODE
309 (0x135)
2.58 – GEMDOS Function Reference

**Availability**
Available when a ‘MiNT’ cookie with a version of at least 0.93 exists.

**Parameters**
Setting `mode` to **DRV_LOCK** (1) places a lock on **BIOS** device `drv` whereas a `mode` setting of **DRV_UNLOCK** (0) unlocks `drv`.

**Binding**

- `move.w drv,-(sp)`
- `move.w move,-(sp)`
- `move.w #$135,-(sp)`
- `trap #1`
- `addq.l #6,sp`

**Return Value**
`Dlock()` returns 0 if successful or a negative GEMDOS error code otherwise.

**Comments**
Locking a device provides a method for device formatters to prevent other processes from simultaneously attempting to access a drive. If a process which locked a device terminates, that device is automatically unlocked.

**BIOS** device numbers and GEMDOS drive letters do not necessarily have a one to one correspondence. To lock a GEMDOS drive use **Fxattr()** to determine the device number of the drive you wish to lock.

**See Also**
**Fxattr()**

---

**Dopendir()**

LONG Dopendir( *name*, *flag* )
char *name;
WORD flag;

**Description**
`Dopendir()` opens the specified directory for reading.

**Opcode**
296 (0x128)

**Availability**
Available when a ‘MiNT’ cookie with a version of at least 0.90 exists.

**Parameters**
- *name* is a pointer to a null-terminated directory specification of the directory to open. *name* should not contain a trailing backslash.
- *flag* determines whether to open the file in normal or compatibility mode. A value of **MODE_NORMAL** (0) for *flag* signifies normal mode whereas a value of **MODE_COMPAT** (1) signifies compatibility mode.

Compatibility mode forces directory searches to be performed much like **Fsfirst()** and **Fsnext()** (restricting filenames to the DOS 8 + 3 standard in uppercase). In normal mode, filenames returned by **Dreaddir()** will be in the format native to the
file system and a **Unix** style file index will be returned.

### Binding

```
move.w flag,-(sp)
lea name
move.w #$128,-(sp)
trap #1
addq.l #8,sp
```

### Return Value

`Dopendir()` returns a **Long** directory handle (which may be positive or negative) if successful. A negative **GEMDOS** error code will be returned if the call fails.

### Caveats

Failure to properly close directory handles may cause the system to eventually run out of handles which will cause the OS to fail.

### Comments

Negative directory handles and negative **GEMDOS** error codes may be differentiated by checking for 0xFF in the high byte. Returned values with 0xFF in the high byte are errors.

### See Also

`Dclosedir()`, `Dreaddir()`, `Drewinddir()`

---

**Dpathconf()**

```c
LONG Dpathconf( name, mode )
char *name;
WORD mode;
```

`Dpathconf()` returns information regarding limits and capabilities of an installed file system.

### Opcode

292 (0x124)

### Availability

This function is available under all **MiNT** versions integrated with **MultiTOS**.

### Parameters

`name` specifies the file system you wish information about. `mode` dictates the return value as follows:

<table>
<thead>
<tr>
<th>Name</th>
<th><code>mode</code></th>
<th>Return Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>DP_INQUIRE</td>
<td>-1</td>
<td>Returns the maximum legal value for the mode parameter in <code>Dpathconf()</code></td>
</tr>
<tr>
<td>DP_IOPEN</td>
<td>0</td>
<td>Returns the possible maximum number of open files at one time. If <strong>UNLIMITED</strong> (0x7FFFFFFF) is returned, then the number of open files is limited only by available memory.</td>
</tr>
<tr>
<td>DP_MAXLINKS</td>
<td>1</td>
<td>Returns the maximum number of links to a file. If <strong>UNLIMITED</strong> (0x7FFFFFFF) is returned, then the number of links to a file is limited only by available memory.</td>
</tr>
<tr>
<td>Function</td>
<td>Value</td>
<td>Description</td>
</tr>
<tr>
<td>--------------</td>
<td>-------</td>
<td>-------------</td>
</tr>
<tr>
<td>DP_PATHMAX</td>
<td>2</td>
<td>Returns the maximum length of a full path name in bytes. If <strong>UNLIMITED</strong> (0x7FFFFFFF) is returned, then the maximum size of a pathname is unlimited.</td>
</tr>
<tr>
<td>DP_NAMEMAX</td>
<td>3</td>
<td>Returns the maximum length of a file name in bytes. If <strong>UNLIMITED</strong> (0x7FFFFFFF) is returned, then the maximum length of a filename is unlimited.</td>
</tr>
<tr>
<td>DP_ATOMIC</td>
<td>4</td>
<td>Returns the number of bytes that can be written per write operation. If <strong>UNLIMITED</strong> (0x7FFFFFFF) is returned, then the number of bytes that can be written at once is limited only by available memory.</td>
</tr>
<tr>
<td>DP_TRUNC</td>
<td>5</td>
<td>Returns a code indicating the type of filename truncation as follows:</td>
</tr>
<tr>
<td></td>
<td></td>
<td><strong>DP_NOTRUNC</strong> (0)</td>
</tr>
<tr>
<td></td>
<td></td>
<td>File names are not truncated. If a file name in any system call exceeds the filename size limit then an <strong>ERANGE</strong> (-64) range error is returned.</td>
</tr>
<tr>
<td></td>
<td></td>
<td><strong>DP_AUTOTRUNC</strong> (1)</td>
</tr>
<tr>
<td></td>
<td></td>
<td>File names are truncated automatically to the maximum allowable length.</td>
</tr>
<tr>
<td></td>
<td></td>
<td><strong>DP_DOSTRUNC</strong> (2)</td>
</tr>
<tr>
<td></td>
<td></td>
<td>File names are truncated to the <strong>DOS</strong> standard (maximum 8 character node with 3 character extension).</td>
</tr>
<tr>
<td>DP_CASE</td>
<td>6</td>
<td>Returns a code which indicates case sensitivity as follows:</td>
</tr>
<tr>
<td></td>
<td></td>
<td><strong>DP_SENSITIVE</strong> (0)</td>
</tr>
<tr>
<td></td>
<td></td>
<td>File system is case-sensitive.</td>
</tr>
<tr>
<td></td>
<td></td>
<td><strong>DP_NOSENSITIVE</strong> (1)</td>
</tr>
<tr>
<td></td>
<td></td>
<td>File system is not case-sensitive (file and path names are always converted to upper-case).</td>
</tr>
<tr>
<td></td>
<td></td>
<td><strong>DP_SAVEONLY</strong> (2)</td>
</tr>
<tr>
<td></td>
<td></td>
<td>File system is not case-sensitive, however, file and path names are saved in their original case. Ex: A file called ‘Compendi.um’ will appear as ‘Compendi.um’ but may be referenced as ‘compendi.um’ or ‘COMPENDI.UM’.</td>
</tr>
</tbody>
</table>

**Binding**

```
move.w mode, -(sp)
pea name
move.w #$124, -(sp)
trap #1
addq.l #8, sp
```

**Return Value**

See above.

**See Also**

`Sysconf()`
Dreaddir()

LONG Dreaddir( len, dirhandle, buf )
WORD len;
LONG dirhandle;
char *buf;

Dreaddir() enumerates the contents of the specified directory.

OPCODE 297 (0x129)

AVAILABILITY Available when a ‘MiNT’ cookie with a version of at least 0.90 exists.

PARAMETERS Dreaddir() fetches information about the next file contained in the directory
specified by dirhandle. len specifies the length of the buffer pointed to by buf
which should be enough to hold the size of the filename, NULL byte, and index (if
in normal mode).

BINDING

    pea    buf
    move.l dirhandle
    move.w len
    move.w #$129,-(sp)
    trap #1
    lea 12(sp),sp

RETURN VALUE Dreaddir() returns a 0 if the operation was successful, ERANGE (-64) if the
buffer was not large enough to hold the index and name, or ENMFIL (-47) if there
were no more files to read.

COMMENTS In normal mode, Dreaddir() returns a 4-byte file index in the first four bytes of
buf. The filename then follows starting at the fifth byte of buf. The file index is
present to prevent confusion under some file systems when two files of the same
name exist. In some file systems this is legal, however, in all file systems, the 4-
byte index will be unique.

    When in compatibility mode, the filename begins at &buf[0].

SEE ALSO Dopendir(), Dclosedir(), Drewinddir()
Drewinddir()

LONG Drewinddir( handle )
LONG handle;

Drewinddir() rewinds the specified directory pointer to its first file.

OPCODE 298 (0x12A)

AVAILABILITY Available when a ‘MiNT’ cookie with a version of at least 0.90 exists.

PARAMETERS handle specifies the directory handle of the directory to rewind.

BINDING
move.l handle,-(sp)
move.w #$12A,-(sp)
trap #1
addq.l #6,sp

RETURN VALUE Drewinddir() returns a 0 if successful or a negative GEMDOS error code otherwise.

SEE ALSO Dopendir(), Dreaddir(), Drewinddir()

Dsetdrv()

LONG Dsetdrv( drive )
WORD drive;

Dsetdrv() sets the current GEMDOS drive and returns a bitmap of mounted drives.

OPCODE 14 (0x0E)

AVAILABILITY All GEMDOS versions.

PARAMETERS drive is the code of the drive to set as the default GEMDOS disk drive. Calling the function as:

bmap = Dsetdrv(Dgetdrv());

will return the bitmap of mounted drives without changing the current GEMDOS drive.

BINDING
move.w drive,-(sp)
Dsetpath() - 2.63

Dsetpath() returns a LONG bit array that indicates which drives are mounted on the system. Bit 0 indicates drive ‘A:’, bit 1 drive ‘B:’, etc.

SEE ALSO  
Dgetdrv()

Dsetpath()

LONG Dsetpath(path)  
char *path;

Dsetpath() sets the path of the current GEMDOS drive.

OPCODE  
59 (0x3B)

AVAILABILITY  
All GEMDOS versions.

PARAMETERS  
path is a pointer to a character buffer containing the new path specification for the current GEMDOS drive.

BINDING  
pea path  
move.w #$3B,-(sp)  
trap #1  
addq.l #6,sp

RETURN VALUE  
Dsetpath() returns one of two return codes on function exit:

- E_OK (0): Operation successful
- EPTHNF (-34): Path not found

CAVEATS  
You may specify a drive letter and colon in the input path specification to set the path of a particular drive but this feature is unstable in all versions of GEMDOS and may confuse drive assignments. It is therefore advised that this feature be avoided.

SEE ALSO  
Dgetpath()
Fattrib()

LONG Fattrib(fname, flag, attr)
char *fname;
WORD flag, attr;

Fattrib() reads or modifies the attribute bits of a GEMDOS file.

OPCODE 67 (0x43)

AVAILABILITY All GEMDOS versions.

PARAMETERS fname is a pointer to a null-terminated string which contains the GEMDOS filename of the file to manipulate. flag should be set to FA_INQUIRE (0) to read the file’s attributes and FA_SET (1) to set them. If you are setting attributes, attr contains the file’s new attributes.

BINDING

move.w attr,-(sp)
move.w flag,-(sp)
pea fname
move.w #$43,-(sp)
trap #1
lea 10(sp),sp

RETURN VALUE If reading the attributes, Fattrib() returns a bit array of attributes as defined below. If setting the attributes, Fattrib() returns the file’s old attributes. In any case, a negative return code indicates that a GEMDOS error occurred.

<table>
<thead>
<tr>
<th>Name</th>
<th>Bit</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>FA_READONLY</td>
<td>0</td>
<td>Read-only flag</td>
</tr>
<tr>
<td>FA_HIDDEN</td>
<td>1</td>
<td>Hidden file flag</td>
</tr>
<tr>
<td>FA_SYSTEM</td>
<td>2</td>
<td>System file flag</td>
</tr>
<tr>
<td>FA_VOLUME</td>
<td>3</td>
<td>Volume label flag</td>
</tr>
<tr>
<td>FA_DIR</td>
<td>4</td>
<td>Subdirectory</td>
</tr>
<tr>
<td>FA_ARCHIVE</td>
<td>5</td>
<td>Archive flag</td>
</tr>
<tr>
<td>—</td>
<td>6...</td>
<td>Currently reserved</td>
</tr>
</tbody>
</table>

CAVEATS GEMDOS versions below 0.15 did not set the archive bit correctly. The archive bit is now correctly set by TOS when a file is created or written to.
Fchmod()

LONG Fchmod(name, mode)

char *name;
WORD mode;

Fchmod() alters file access permissions of the named file.

OPCODE

306 (0x132)

AVAILABILITY

Available when a ‘MiNT’ cookie with a version of at least 0.90 exists.

PARAMETERS

name specifies a valid GEMDOS file specification of the file whose access permissions you wish to modify. mode is a bit mask composed by OR’ing together values defined as follows:

<table>
<thead>
<tr>
<th>Name</th>
<th>Mask</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>S_IRUSR</td>
<td>0x100</td>
<td>Read permission for the owner of the file.</td>
</tr>
<tr>
<td>S_IWUSR</td>
<td>0x80</td>
<td>Write permission for the owner of the file.</td>
</tr>
<tr>
<td>S_IXUSR</td>
<td>0x40</td>
<td>Execute permission for the owner of the file.</td>
</tr>
<tr>
<td>S_IRGRP</td>
<td>0x20</td>
<td>Read permission for members of the same group the file belongs to.</td>
</tr>
<tr>
<td>S_IWGRP</td>
<td>0x10</td>
<td>Write permission for members of the same group the file belongs to.</td>
</tr>
<tr>
<td>S_IXGRP</td>
<td>0x08</td>
<td>Execute permission for members of the same group the file belongs to.</td>
</tr>
<tr>
<td>S_IROTH</td>
<td>0x04</td>
<td>Read permission for all others.</td>
</tr>
<tr>
<td>S_IWOTH</td>
<td>0x02</td>
<td>Write permission for all others.</td>
</tr>
<tr>
<td>S_IXOTH</td>
<td>0x01</td>
<td>Execute permission for all others.</td>
</tr>
</tbody>
</table>

BINDING

move.w mode, -(sp)
pea name
move.w #$132, -(sp)
trap #1
addq.l #8, sp

RETURN VALUE

Fchmod() returns E_OK (0) if successful or a negative GEMDOS error code otherwise.

CAVEATS

Not all file systems support all bits. Unrecognized bits will be ignored.

COMMENTS

Only the owner of a file may change a file’s permission status.

‘Execute’ status refers to the permission to search the named directory for a file name or component.
Fchown()

LONG Fchown( name, uid, gid )
char *name;
WORD uid, gid;

Fchown() changes a file’s ownership.

OPCODE 305 (0x131)

AVAILABILITY Available when a ‘MiNT’ cookie with a version of at least 0.90 exists.

PARAMETERS name specifies the file whose ownership status you wish to change. uid sets the
new owner and gid sets the new group.

BINDING

move.w gid,-(sp)
move.w uid,-(sp)
pea name
move.w #$131,-(sp)
trap #1
lea 10(sp),sp

RETURN VALUE Fchown() returns 0 if the operation was successful or a negative GEMDOS error
code otherwise.

CAVEATS Most file systems don’t understand the concept of file ownership (including TOS).

COMMENTS uid may only be modifies if the caller’s uid is 0. gid may only be changed to the
group id of a group the caller belongs to.

SEE ALSO Fchmod(), Fxattr()

Fclose()

LONG Fclose( handle )
WORD handle;

Fclose() closes the file specified.

OPCODE 62 (0x3E)
**Availability**

All GEMDOS versions.

**Parameters**

`handle` is a valid WORD file handle which will be closed as a result of this call.

**Binding**

```assembly
move.w handle, -(sp)
mov.w #$3E, -(sp)
trap #1
addq.l #4, sp
```

**Return Value**

`Fclose()` returns **E_OK (0)** if the file was closed successfully or **EIHNDL (-37)** if the handle given was invalid.

**Caveats**

Calling this function with an invalid file handle will crash the system on GEMDOS versions below 0.15. In addition, GEMDOS versions below 0.15 will become confused if you close a standard GEMDOS handle (0-5).

**Comments**

As of GEMDOS version 0.15, closing a standard GEMDOS handle (0-5) will simply reset it to its default BIOS state.

**See Also**

`Fcreate()`, `Fopen()`

---

**Fcnte()**

```assembly
LONG Fcntl( handle, arg, cmd )
WORD handle;
LONG arg;
WORD cmd;
```

**Fcntl()** performs a command on the specified file.

**Opcode**

260 (0x104)

**Availability**

This function is available under all MiNT versions integrated with MultiTOS.

**Parameters**

`handle` specifies the GEMDOS file handle of the file on which the operation specified by `cmd` will affect. `arg` varies with each command. Valid commands are:

<table>
<thead>
<tr>
<th>cmd</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>F_DUPFD</td>
<td>Duplicate the given file handle. Fcntl() will return a file handle in the range <code>arg</code> – 32. If no file handles exist within that range, an error will be returned.</td>
</tr>
<tr>
<td>F_GETFD</td>
<td>Return the inheritance flag for the specified file. A value of 1 indicates that child processes started with <code>Pexec()</code> will inherit this file handle, otherwise a value of 0 is returned. <code>arg</code> is ignored.</td>
</tr>
</tbody>
</table>
F_SETFD (0x0002)
Set the inheritance flag for the named file. arg specifies if child processes started with Pexec() will inherit the file handle. A value of 0 indicates that they will not. A value of 1 indicates that they will.
GEMDOS handles 0-5 default to a value of 1 whereas other handles default to a value of 0.

F_GETFL (0x0003)
Return the file descriptor flags for the specified file. These are the same flags passed to Fopen(). arg is ignored.

F_SETFL (0x0004)
Set the file descriptor flags for the specified file to arg. Only user-modifiable bits are considered. All others should be 0. It is not possible to change a file's read/write mode or sharing modes with this call. Attempts to do this will fail without returning an error code.

F_GETLK (0x0005)
Test for the presence of a lock on the specified file. arg is a pointer to a FLOCK structure defined as follows:

```c
typedef struct flock {
    /* Type of lock
     * 0 = Read-only lock
     * 1 = Write-only lock
     * 2 = Read/Write lock */
    WORD l_type;
    /* 0 = offset from beginning of file
     * 1 = offset from current position
     * 2 = offset from end of file */
    WORD l_whence;
    /* Offset to start of lock */
    LONG l_start;
    /* Length of lock (0 for rest of file) */
    LONG l_len;
    /* Process ID maybe filled in by call */
    WORD l_pid;
} FLOCK;
```

If a prior lock exists which would prevent the specified lock from being applied, the interfering lock is copied into the structure with the process ID of the locking process. Otherwise, Fcntl() returns F_UNLCK (3).

F_SETLK (0x0005)
Set or remove an advisory lock on the specified file. arg points to a FLOCK structure as defined above.

Setting l_type to F_RDLOCK or F_WRLCK will cause a lock to be set. Setting l_type to F_UNLCK will attempt to remove the specified lock.

When locking and unlocking FIFO's, l_whence, l_start, and l_len should be 0.

The command returns 0 if successful or a negative GEMDOS error code otherwise.

F_SETLKW (0x0007)
The calling procedure is the same as above, however, if other processes already have a conflicting lock set, it will suspend the calling process until the lock is freed.

FSTAT (0x4600)
Get the extended attributes for a file. arg points to a XATTR structure which is filled in with the file's extended attributes. If successful, the function returns 0, otherwise a negative GEMDOS error code is returned. See Fxattr() for an explanation of the XATTR structure.
<table>
<thead>
<tr>
<th>Function</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>FIONREAD (0x4601)</td>
<td>Return an estimate of the number of bytes available for reading from the specified file without causing the process to block (wait for more input) in the LONG pointed to by arg.</td>
</tr>
<tr>
<td>FIONWRITE (0x4602)</td>
<td>Return an estimate of the number of bytes that may be written from the specified file without causing the process to block in the LONG pointed to by arg.</td>
</tr>
<tr>
<td>SHMGETBLK (0x4D00)</td>
<td>Returns the address of a memory block associated with the file. arg should be NULL for future compatibility. Note: Different processes may receive different addresses for a shared block.</td>
</tr>
<tr>
<td>SHMSETBLK (0x4D01)</td>
<td>arg points to a block of memory (previously allocated) which is to be associated with the file. The file must have been created at ‘U:\SHM:’ or the call will fail.</td>
</tr>
<tr>
<td>PPROCADDR (0x5001)</td>
<td>Return the address of the specified processes’ control structure (opened as a file) in arg. See the discussion of MINT processes for information about this structure.</td>
</tr>
<tr>
<td>PBASEADDR (0x5002)</td>
<td>Return the address of the specified processes’ GEMDOS basepage (opened as a file) in arg.</td>
</tr>
<tr>
<td>PCTXTSIZE (0x5003)</td>
<td>Return the length of the specified processes’ context structure (opened as a file) in arg. Seeking to the offset returned by PPROCADDR minus this number and reading this many bytes will yield the current user context of the process. Seeking back this many bytes more and reading will yield the last system context of the process. This structure is volatile and is likely to change from one MINT version to the next.</td>
</tr>
<tr>
<td>PSETFLAGS (0x5004)</td>
<td>arg is a pointer to a LONG from which the processes’ memory allocation flags (PRGFLAGS) will be set.</td>
</tr>
<tr>
<td>PGGETFLAGS (0x5005)</td>
<td>arg is a pointer to a LONG into which the processes’ memory allocation flags (PRGFLAGS) will be placed.</td>
</tr>
<tr>
<td>PTRACEGFLAGS (0x5006)</td>
<td>arg points to a WORD which will be filled in with the trace flags of a process. Setting bit #0 of arg causes the parent process to receive signals destined for the child. See the discussion on program debugging for more information.</td>
</tr>
<tr>
<td>PTRACESFLAGS (0x5007)</td>
<td>arg points to a WORD which will be used to set the trace flags of a process. See the discussion on program debugging for more information.</td>
</tr>
<tr>
<td>PTRACEGO (0x5008)</td>
<td>This call restarts a process which was stopped because of a signal. arg points to a WORD which contains 0 to clear all of the child processes’ pending signals or the signal value to send to the process.</td>
</tr>
<tr>
<td>PTRACEFLOW (0x5009)</td>
<td>This call restarts a process in a special tracing mode in which the process is stopped and a SIGTRACE signal is generated whenever program flow changes (ex: JSR/BSR/JMP/BEQ). arg should be set to 0 to clear all of the pending signals of the process being traced or a signal value which is to be sent to the child.</td>
</tr>
<tr>
<td>PTRACESTEP (0x500A)</td>
<td>This call restarts a process and allows it to execute one instruction before a SIGTRAP instruction is generated.</td>
</tr>
<tr>
<td>Function</td>
<td>Description</td>
</tr>
<tr>
<td>--------------</td>
<td>-------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------</td>
</tr>
<tr>
<td><strong>PLOADINFO</strong></td>
<td><strong>arg</strong> points to a structure as follows:</td>
</tr>
</tbody>
</table>
| (0x500C)     | ```
struct ploadinfo
{
   WORD fnamelen;
   char * cmdlin;
   char * fname;
};
```                                                                                                                                                      |         |
|              | *cmdlin* should point to a 128 byte character buffer into which the processes’ command line will be copied.                                                                                                |         |
|              | *fname* should point to a buffer *fnamelen* bytes long into which the complete path and filename of the process’ parent will be copied. If the buffer is too short the call will return **ENAMETOOLONG**. |         |
| **TIOCGETP** | Get terminal parameters from the TTY device with the specified file handle. **arg** is a pointer to an **sgttyb** structure which is filled in by this command.                                                    |         |
| (0x5400)     | ```
struct sgttyb
{
   /* Reserved */
   char sg_ispeed;
   /* Reserved */
   char sg_ospeed;
   /* Erase character */
   char sg_erase;
   /* Line kill character */
   char sg_kill;
   /* Terminal control flags */
   WORD sg_flags;
};
```                                                                                                                                                      |         |
| **TIOCSETP** | Set the terminal parameters of the TTY device specified. **arg** is a pointer to an **sgttyb** structure as defined above. You should first get the terminal control parameters, modify what you wish to change, and then set them with this call. |         |
| (0x5401)     |                                                                perlmcases                                                                                                                                        |         |
| **TIOCGETC** | Get the terminal control characters of the TTY device specified. **arg** is a pointer to a **tchars** structure filled in by this call which is defined as follows:                                                  |         |
| (0x5402)     | ```
struct tchars
{
   /* Raises SIGINT */
   char t_intrc;
   /* Raises SIGKILL */
   char t_quitc;
   /* Starts terminal output */
   char t_startc;
   /* Stops terminal output */
   char t_stopc;
   /* Marks end of file */
   char t_eofc;
   /* Marks end of line */
   char t_brkc;
};
```                                                                                                                                                      |         |
<p>| <strong>TIOCSETC</strong> | Set the terminal control characters of the TTY device specified. <strong>arg</strong> is a pointer to a <strong>tchars</strong> structure as defined above. Setting any structure element to 0 disables that feature.                         |         |
| (0x5403)     |                                                                perlmcases                                                                                                                                        |         |</p>
<table>
<thead>
<tr>
<th>Function</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>TIOCGLTC</td>
<td>Get the extended terminal control characters from the TTY device specified.</td>
</tr>
<tr>
<td>(0x5404)</td>
<td><em>arg</em> is a pointer to a <em>ltchars</em> structure which is filled in by this call</td>
</tr>
<tr>
<td></td>
<td>defined as follows:</td>
</tr>
<tr>
<td></td>
<td>struct ltchars</td>
</tr>
<tr>
<td></td>
<td>{</td>
</tr>
<tr>
<td></td>
<td>/* Raise SIGTSTP now */</td>
</tr>
<tr>
<td></td>
<td>char t_suspc;</td>
</tr>
<tr>
<td></td>
<td>/* Raise SIGTSTP when read */</td>
</tr>
<tr>
<td></td>
<td>char t_dsuspc;</td>
</tr>
<tr>
<td></td>
<td>/* Redraws the input line */</td>
</tr>
<tr>
<td></td>
<td>char t_rprntc;</td>
</tr>
<tr>
<td></td>
<td>/* Flushes output */</td>
</tr>
<tr>
<td></td>
<td>char t_flushc;</td>
</tr>
<tr>
<td></td>
<td>/* Erases a word */</td>
</tr>
<tr>
<td></td>
<td>char t_werasc;</td>
</tr>
<tr>
<td></td>
<td>/* Quotes a character */</td>
</tr>
<tr>
<td></td>
<td>char t_lnextc;</td>
</tr>
<tr>
<td></td>
<td>};</td>
</tr>
<tr>
<td>TIOCSLTC</td>
<td>Set the extended terminal control characters for the TTY device specified.</td>
</tr>
<tr>
<td>(0x5405)</td>
<td>from the <em>ltchars</em> structure pointed to by <em>arg</em>.</td>
</tr>
<tr>
<td>TIOCGPGRP</td>
<td>Return the process group ID for the TTY specified in the <em>LONG</em> pointed to</td>
</tr>
<tr>
<td>(0x5406)</td>
<td>by <em>arg</em>.</td>
</tr>
<tr>
<td>TIOCSPGRP</td>
<td>Set the process group ID of the TTY specified in the <em>LONG</em> pointed to</td>
</tr>
<tr>
<td>(0x5407)</td>
<td>by <em>arg</em>.</td>
</tr>
<tr>
<td>TIOCSTOP</td>
<td>Stop terminal output (as if the user had pressed CTRL-S). <em>arg</em> is ignored.</td>
</tr>
<tr>
<td>(0x5409)</td>
<td>TIOCSTART</td>
</tr>
<tr>
<td>(0x540A)</td>
<td>Restart output to the terminal (as if the user had pressed CTRL-Q) if it</td>
</tr>
<tr>
<td></td>
<td>had been previously stopped with CTRL-S or a TIOCSTOP command. <em>arg</em> is</td>
</tr>
<tr>
<td></td>
<td>ignored.</td>
</tr>
<tr>
<td>TIOCGWINSZ</td>
<td>Get information regarding the window for this terminal. <em>arg</em> points to a</td>
</tr>
<tr>
<td>(0x540B)</td>
<td><em>winsize</em> structure which is filled in by this command.</td>
</tr>
<tr>
<td></td>
<td>struct winsize</td>
</tr>
<tr>
<td></td>
<td>{</td>
</tr>
<tr>
<td></td>
<td>/* # of Text Rows */</td>
</tr>
<tr>
<td></td>
<td>WORD ws_row;</td>
</tr>
<tr>
<td></td>
<td>/* # of Text Columns */</td>
</tr>
<tr>
<td></td>
<td>WORD ws_column;</td>
</tr>
<tr>
<td></td>
<td>/* Width of window in pixels */</td>
</tr>
<tr>
<td></td>
<td>WORD ws_xpixel;</td>
</tr>
<tr>
<td></td>
<td>/* Height of window in pixels */</td>
</tr>
<tr>
<td></td>
<td>};</td>
</tr>
<tr>
<td>TIOCSWINSZ</td>
<td>Change the extents of the terminal window for the specified TTY. <em>arg</em></td>
</tr>
<tr>
<td>(0x540C)</td>
<td>points to a <em>winsize</em> structure which contains the new window information.</td>
</tr>
<tr>
<td></td>
<td>It is up to the window manager to modify the window extents and raise the</td>
</tr>
<tr>
<td></td>
<td>SIGWINCH signal if necessary.</td>
</tr>
</tbody>
</table>
### TIOCGXKEY

(0x540D)

Return the current definition of a system key. `arg` points to a structure `xkey` as follows:

```c
struct xkey {
    WORD xk_num;
    char xk_def[8];
};
```

`xk_def` will be filled in with the **NULL** terminated name associated with the key specified in `xk_num` as follows:

<table>
<thead>
<tr>
<th><code>xk_num</code></th>
<th>Key</th>
</tr>
</thead>
<tbody>
<tr>
<td>0-9</td>
<td>F1-F10</td>
</tr>
<tr>
<td>10-19</td>
<td>F11-F20</td>
</tr>
<tr>
<td>20</td>
<td>Cursor up</td>
</tr>
<tr>
<td>21</td>
<td>Cursor down</td>
</tr>
<tr>
<td>22</td>
<td>Cursor right</td>
</tr>
<tr>
<td>23</td>
<td>Cursor left</td>
</tr>
<tr>
<td>24</td>
<td>Help</td>
</tr>
<tr>
<td>25</td>
<td>Undo</td>
</tr>
<tr>
<td>26</td>
<td>Insert</td>
</tr>
<tr>
<td>27</td>
<td>Clr/Home</td>
</tr>
<tr>
<td>28</td>
<td>Shift+Cursor up</td>
</tr>
<tr>
<td>29</td>
<td>Shift+Cursor down</td>
</tr>
<tr>
<td>30</td>
<td>Shift+Cursor right</td>
</tr>
<tr>
<td>31</td>
<td>Shift+Cursor left</td>
</tr>
</tbody>
</table>

### TIOCSXKEY

(0x540E)

Set the current definition of a system key. `arg` must point to an `xkey` structure (as defined above). `xk_num` and `xk_def` are used to set the text associated with a system key.

If a terminal recognizes special keys (by having its **XKEY** bit set in the `sg_flags` field of its `sgttyb` structure) then setting a system key will cause the text specified by `xk_def` to be sent to a process whenever the key is struck. Note: this works only if the terminal is reading characters using `Fread()`.

### TIOCIBAUD

(0x5412)

Read/Write the input baud rate for the specified terminal device. If `arg` points to a `LONG` then the input baud rate will be set to that value. If `arg` is 0, the DTR on the terminal will be dropped (if this feature is supported). If `arg` is negative, the baud rate will not be changed. The old baud rate is returned in the value pointed to by `arg`.

If the terminal does not support separate input and output baud rates then this call will set both rates.

### TIOCOBAUD

(0x5413)

Read/Write the output baud rate for the specified terminal device. If `arg` points to a `LONG` then the output baud rate will be set to that value. If `arg` is 0, the DTR on the terminal will be dropped (if this feature is supported). If `arg` is negative, the baud rate will not be changed. The old baud rate is returned in the value pointed to by `arg`.

If the terminal does not support separate input and output baud rates then this call will set both rates.

### TIOCCBRK

(0x5414)

Clear the break condition on the specified device. `arg` is ignored.

### TIOSCBRK

(0x5415)

Set the break condition on the specified device. `arg` is ignored.
### TIOCGFLAGS (0x5416)
Return the current stop bit/data bit configuration for the terminal device in the lower 16 bits of the LONG pointed to by `arg`. See the entry for TIOCSFLAGS for the flags required to parse `arg`.

### TIOCSFLAGS (0x5417)
Set the current stop bit/data bit configuration for the terminal device. The new configuration is contained in `arg`. Valid mask values for `arg` are as follows:

<table>
<thead>
<tr>
<th>Name</th>
<th>Mask</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>TF_18BIT</td>
<td>0x0000</td>
<td>8 data bits</td>
</tr>
<tr>
<td>TF_7BIT</td>
<td>0x0004</td>
<td>7 data bits</td>
</tr>
<tr>
<td>TF_6BIT</td>
<td>0x0008</td>
<td>6 data bits</td>
</tr>
<tr>
<td>TF_5BIT</td>
<td>0x000C</td>
<td>5 data bits</td>
</tr>
</tbody>
</table>

### TCURSOFF (0x6300)
Hide the cursor on the selected terminal device. `arg` is ignored.

### TCURSON (0x6301)
Show the cursor on the selected terminal device. `arg` is ignored.

### TCURSBLINK (0x6302)
Enable cursor blinking on the selected terminal device. `arg` is ignored.

### TCURSSTEADY (0x6303)
Disable cursor blinking on the selected terminal device. `arg` is ignored.

### TCURSRRATE (0x6304)
Set the cursor blink rate to the WORD pointed to by `arg`.

### TCURSGRATE (0x6305)
Return the current cursor blink rate in the WORD pointed to by `arg`.

### Binding
```
move.w   cmd, -(sp)
mov.e.l  arg, -(sp)
mov.w   handle, -(sp)
mov.w   #$260, -(sp)
trap     #1
lea      10(sp), sp
```

### Return Value
Unless otherwise noted, `Fcntl()` returns a 0 if the operation was successful or a negative GEMDOS error code otherwise.

### See Also
`Flock()`, `Fopen()`, `Fxattr()`, `Pgetpgrp()`, `Psetpgrp()`

---

### Fcreate()

**LONG** `Fcreate(fname, attr)`

```c
char *fname;
WORD attr;
```

`Fcreate()` creates a new file (or truncates an existing one) with the specified name and attributes.

**Opcode**

60 (0x3C)
All GEMDOS versions.

_parameters_

fname is a character pointer to the GEMDOS file specification of the file to create or truncate. attr is a bit array which specifies the attributes of the new file. Valid mask values are given below:

<table>
<thead>
<tr>
<th>Name</th>
<th>Bit</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>FA_READONLY</td>
<td>0</td>
<td>Read-only file</td>
</tr>
<tr>
<td>FA_HIDDEN</td>
<td>1</td>
<td>Hidden file</td>
</tr>
<tr>
<td>FA_SYSTEM</td>
<td>2</td>
<td>System file</td>
</tr>
<tr>
<td>FA_VOLUME</td>
<td>3</td>
<td>Volume label</td>
</tr>
<tr>
<td></td>
<td>4</td>
<td>Reserved</td>
</tr>
<tr>
<td>FA_ARCHIVE</td>
<td>5</td>
<td>Archive bit</td>
</tr>
</tbody>
</table>

_BINDING_

move.w attr,-(sp)
pea fname,-(sp)
move.w #$3C,-(sp)
trap #1
addq.l #8,sp

RETURN VALUE

Fcreate() returns a LONG value. If the LONG is negative, it should be interpreted as a GEMDOS error. Possible errors are EPTHNF (-34), ENHNDL (-35), or EACCDN (-36).

If positive, the WORD portion of the returned LONG should be regarded as the file handle.

CAVEATS

With GEMDOS version 0.13, creating a read-only file returns a read-only file handle which is of little use. GEMDOS versions below 0.15 incorrectly allow more than one volume label per disk.

COMMENTS

GEMDOS versions 0.15 and above automatically set the archive bit. You may set it yourself on versions below 0.15.

SEE ALSO

Fopen(), Fclose()
Fdatime()

LONG Fdatime( timeptr, handle, flag )
DATETIME *timeptr;
WORD handle, flag;

Fdatime() reads or modifies a file’s time and date stamp.

OPCODE 87 (0x57)

AVAILABILITY All GEMDOS versions.

PARAMETERS timeptr is a pointer to a DATETIME structure which is represented below. handle is a valid GEMDOS file handle to the file to modify. flag is FD_INQUIRE (0) to fill timeptr with the file’s date/timestamp and FD_SET (1) to change the file’s date/timestamp to the contents of timeptr.

typedef struct
{
  unsigned hour:5;
  unsigned minute:6;
  unsigned second:5;
  unsigned year:7;
  unsigned month:4;
  unsigned day:5;
} DATETIME;

BINDING
move.w flag,-(sp)
move.w handle,-(sp)
pea timeptr
move.w #$57,-(sp)
trap #1
lea 10(sp),sp

RETURN VALUE Fdatime() returns a 0 if the date/time was successfully read/modified. Otherwise, it returns a negative GEMDOS error code.

CAVEATS GEMDOS versions below 0.15 yielded very unpredictable results with this call and should therefore be avoided.

COMMENTS timeptr.second should be multiplied times two to obtain the actual value. timeptr.year is expressed as an offset from 1980.
Fdelete()

LONG Fdelete( fname )
char *fname;

Fdelete() deletes the specified file.

OPCODE 65 (0x41)

AVAILABILITY All GEMDOS versions.

PARAMETERS fname is the GEMDOS file specification of the file to be deleted.

BINDING
   pea   fname
   move.w #41, -(sp)
   trap  	#1
   addq.l #6, sp

RETURN VALUE Fdelete() returns E_OK (0) if the operation was successful or a negative GEMDOS error code if it fails.

CAVEATS Do not attempt to delete a file that is currently open or unpredictable results will occur.

COMMENTS Ddelete() must be used to delete subdirectories.

SEE ALSO Ddelete()

Fdup()

LONG Fdup( shandle )
WORD shandle;

Fdup() duplicates a standard file handle (0-5) and assigns it a new handle (>6).

OPCODE 69 (0x45)

AVAILABILITY All GEMDOS versions.

PARAMETERS shandle is the standard GEMDOS handle to be duplicated.

BINDING
   move.w shandle, -(sp)
   move.w #45, -(sp)
   trap 	#1
Fforce() - 2.77

RETURN VALUE
Fdup() returns a normal GEMDOS file handle in the lower WORD of the returned LONG. If the LONG return value is negative then it should be treated as a GEMDOS error code.

COMMENTS
This function is generally used to save a standard file handle so that an Fforce() operation may be undone.

SEE ALSO
Fforce()

Fforce()

LONG Fforce( shandle, nhandle )
WORD shandle, nhandle;

Fforce() is used to redirect the standard input or output from a GEMDOS standard handle to a specific handle created by the application.

OPCODE
70 (0x46)

AVAILABILITY
All GEMDOS versions.

PARAMETERS
shandle is a standard GEMDOS handle to be redirected. nhandle is the new handle you wish to direct it to. Valid values for shandle and nhandle are as follows:

<table>
<thead>
<tr>
<th>Name</th>
<th>Handle</th>
<th>GEMDOS Filename</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>GSH_CONIN</td>
<td>0</td>
<td>con:</td>
<td>Standard input (defaults to whichever BIOS device is mapped to GEMDOS handle -1)</td>
</tr>
<tr>
<td>GSH_CONOUT</td>
<td>1</td>
<td>con:</td>
<td>Standard output (defaults to whichever BIOS device is mapped to GEMDOS handle -1)</td>
</tr>
<tr>
<td>GSH_AUX</td>
<td>2</td>
<td>aux:</td>
<td>Currently mapped serial device (defaults to whichever BIOS device is mapped to GEMDOS handle -2)</td>
</tr>
<tr>
<td>GSH_PRN</td>
<td>3</td>
<td>prn:</td>
<td>Printer port (defaults to whichever BIOS device is currently mapped to GEMDOS handle -3).</td>
</tr>
<tr>
<td></td>
<td>4</td>
<td>None</td>
<td>Reserved</td>
</tr>
<tr>
<td></td>
<td>5</td>
<td>None</td>
<td>Reserved</td>
</tr>
<tr>
<td>GSH_BIOSCON</td>
<td>-1</td>
<td>None</td>
<td>Refers to BIOS handle 2. This handle may only be redirected under the presence of MINT. Doing so redirects output of the BIOS.</td>
</tr>
</tbody>
</table>
**GEMDOS Function Reference**

<table>
<thead>
<tr>
<th>Function</th>
<th>Handle</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>GSH_BIOSAUX</td>
<td>-2</td>
<td>None</td>
</tr>
<tr>
<td>GSH_BIOSPRN</td>
<td>-3</td>
<td>None</td>
</tr>
<tr>
<td>GSH_MIDIIN</td>
<td>-4</td>
<td>None</td>
</tr>
<tr>
<td>GSH_MIDIOUT</td>
<td>-5</td>
<td>None</td>
</tr>
</tbody>
</table>

**BINDING**

```assembly
move.w nhandle,-(sp)
movw shandle,-(sp)
mov.w #$46,-(sp)
trap #1
addq.l #6,sp
```

**RETURN VALUE**

Fforce() returns **E_OK** (0) if no error occurred or **EIHNDL** (-37) if a bad handle is given.

**CAVEATS**

Prior to GEMDOS versions 0.15, handles forced to the printer would not work properly.

**COMMENTS**

This function is often used to redirect the input or output of a child process. It should be used in conjunction with Fdup() to restore the standard handle before process termination. In addition, you should be aware that any file handle redirected to a standard handle (‘con:’ for example) will be closed when the child exits and should not be closed by the parent.

Standard GEMDOS file handles which have been redirected will revert to their original mapping upon Fclose().

**SEE ALSO**

Fdup()

---

**Fgetchar()**

LONG Fgetchar(handle, mode)

WORD handle, mode;

Fgetchar() reads a character from the specified handle.

**OPCODE**

263 (0x107)

**AVAILABILITY**

This function is available under all MiNT versions integrated with MultiTOS.

---

**THE ATARI COMpendium**
**PARAMETERS**  
*handle* is a valid GEMDOS handle to read from. If *handle* is a TTY then *mode* (a bit mask) has meaning as follows:

<table>
<thead>
<tr>
<th>Name</th>
<th>mode</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>TTY_COOKED</td>
<td>0x01</td>
<td>Cooked mode. Special control characters such as CTRL-C and CTRL-Z are checked and acted upon. In addition, flow control with CTRL-S and CTRL-Q is activated.</td>
</tr>
<tr>
<td>TTY_ECHO</td>
<td>0x02</td>
<td>Echo mode. Characters read are echoed back to the TTY.</td>
</tr>
</tbody>
</table>

**BINDING**

```
mov w          mode, -(sp)
mov w          handle, -(sp)
mov w          #$107, -(sp)
trap           #1
addq l         #6, sp
```

**RETURN VALUE**  
`Fgetchar()` returns the character read in the low byte of the returned LONG. If the device is a terminal where scan codes are available, the LONG will be mapped in the same manner as `Bconin()`. If an end-of-file is reached, the value 0xFF1A will be returned.

**SEE ALSO**  
`Bconin()`, `Fputchar()`, `Fread()`

---

**Fgetdta()**

DTA *Fgetdta( VOID )*

`Fgetdta()` returns current DTA (Disk Transfer Address)

**_OPCODE**  
47 (0x2F)

**AVAILABILITY**  
All GEMDOS versions.

**PARAMETERS**  
None.

**BINDING**

```
mov w          #$2F,-(sp)
trap           #1
addq l         #2, sp
```

**RETURN VALUE**  
`Fgetdta()` returns a pointer to the current Disk Transfer Address. The structure DTA is defined as:

```c
typedef struct
{
    BYTE d_reserved[21];
    BYTE d_attrib;
    UWORD d_time;
} DTA;
```
UWORD d_date;
LONG d_length;
char d_fname[14];
} DTA;

**COMMENTS**

When an application starts, its **DTA** overlaps the command line string in the processes’ basepage. Any use of the **Fsfirst()** or **Fsnext()** call without first reallocating a new **DTA** will cause the processes’ command line to be corrupted.

To prevent this, you should use **Fsetdta()** to define a new **DTA** structure for your process prior to using **Fsfirst()** or **Fsnext()**. Be careful to avoid assigning your **DTA** to a local or automatic variable without setting it to its original value before the variable goes out of scope.

**SEE ALSO**

**Fsetdta(), Fsfirst(), Fsnext()**

---

**Finstat()**

LONG Finstat( handle )
WORD handle;

**Finstat()** determines the input status of a file.

**OPCODE**

261 (0x105)

**AVAILABILITY**

This function is available under all **MiNT** versions integrated with **MultiTOS**.

**PARAMETERS**

*handle* specifies the **GEMDOS** file handle of the file to return information about.

**BINDING**

move.w handle,-(sp)
move.w #$105,-(sp)
trap #1
addq.l #4,sp

**RETURN VALUE**

**Finstat()** returns 0 or a positive number of characters waiting to be read if successful. A negative **GEMDOS** error code is returned otherwise.

**CAVEATS**

Currently **Finstat()** always returns 0 for disk files.

**SEE ALSO**

**Cauxis(), Cconis(), Fcntl(), Foutstat()**
**Flink()**

```c
LONG Flink( oldname, newname )
char *oldname, *newname;
```

`Flink()` creates a new name for the specified file. After the call the file may be referred to by either name. An `Fdelete()` call on one filename will not affect the other.

**OPCODE** 301 (0x12D)

**AVAILABILITY** Available when a ‘MiNT’ cookie with a version of at least 0.90 exists.

**PARAMETERS**

- `oldname` points to the GEMDOS path specification of the currently existing file
- `newname` specifies the name of the alias to create.

**BINDING**

```
pea newname
pea oldname
move.w #$12D,-(sp)
trap #1
lea 10(sp),sp
```

**RETURN VALUE** `Flink()` returns a 0 if successful or a negative GEMDOS error code otherwise.

**CAVEATS** Not all file systems support ‘hard links’.

**COMMENTS** The filenames given must reside on the same physical device.

**SEE ALSO** `Frename()`, `Fsymlink()`

---

**Flock()**

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

`Flock()` sets or removes a lock on a portion of a file which prevents other processes from accessing it.

**OPCODE** 92 ($5C)

**AVAILABILITY** Only present when ‘_FLK’ cookie exists.
PARAMETERS

handle specifies the GEMDOS handle of the file. mode is FLK_LOCK (0) to create a lock and FLK_UNLOCK (1) to remove it. start specifies the byte offset from the beginning of the file which indicates where the lock starts. length specifies the length of the lock in bytes.

BINDING

move.l length,-(sp)
move.l start,-(sp)
move.w mode,-(sp)
move.w handle,-(sp)
lea 12(sp),sp

RETURN VALUE

Flock() returns E_OK (0) if the call was successful, E_LOCKED (-58) if an overlapping section of the file was already locked, ENSLOCK (-59) if a matching lock was not found for removal, or another GEMDOS error code as appropriate.

COMMENTS

To remove a lock, you must specify identical start and length parameters as you originally set.

MiNT allows locks to be set on devices by locking their entry in ‘U:\DEV\’ as shown in the example below:

```c
handle = Fopen( "U:\DEV\MODEM1", 3 );
if( handle < 0 )
    return ERR_CODE;    /* Unable to open. */
retcode = Flock( (WORD)handle, 0, 0, 0 );    /* Lock */
if( retcode != E_OK )
    return FILE_IN_USE;    /* File is already locked */
/*
 * Now do device input/output.
*/
Flock( (WORD)handle, 1, 0, 0 );    /* Unlock */
Fclose( (WORD)handle );
```

SEE ALSO

Fopen(), Fwrite(), Fread()

---

Fmidipipe()

LONG Fmidipipe( pid, in, out )
WORD pid, in, out;

Fmidipipe() is used to change the file handles used for MIDI input and output.

OPCODE

294 (0x126)
**Availability**

Available when a ‘MiNT’ cookie with a version of at least 0.90 exists.

**Parameters**

`pid` is the process id of the process whose MIDI devices you wish to alter. If `pid` is 0, then the current process will be modified. `in` specifies the GEMDOS file handle of the device to handle MIDI input. `out` specifies the GEMDOS file handle of the device to handle MIDI output.

**Binding**

```asm
move.w out,-(sp)
move.w in,-(sp)
move.w pid,-(sp)
move.w #$126,-(sp)
trap #1
addq.l #8,sp
```

**Return Value**

`Fmidipipe()` returns a 0 if successful or a negative GEMDOS error code otherwise.

**Comments**

An `Fmidipipe(0, in, out)` call is essentially the same as:

```asm
Fforce( -4, in);
Fforce( -5, out);
```

After this call, any `Bconin()` calls to MIDI device 5 will translate to a one character read from handle `in`. Likewise any `Bconout()` calls to MIDI device 5 will translate to a one character write to handle `out`.

**See Also**

`Fdup()`, `Fforce()`

---

**Fopen()**

```asm
LONG Fopen( fname, mode )
char *fname;
WORD mode;
```

`Fopen()` opens the GEMDOS file specified.

**Opcode**

61 ($3D)

**Availability**

All GEMDOS versions. `mode` bits pertaining to file sharing/record locking are only valid when the ‘_FLK’ cookie is present.

**Parameters**

`fname` is the GEMDOS file specification of the file to be opened. `mode` specifies the mode the file is to be placed into once opened. `mode` is a bit array which may be formed by using the bit masks given as follows:
Bits 0-2 specify the file access code as follows:

<table>
<thead>
<tr>
<th>Bit 2</th>
<th>Bit 1</th>
<th>Bit 0</th>
<th>File Access Codes</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0</td>
<td>0</td>
<td>Read only access (S_READ)</td>
</tr>
<tr>
<td>0</td>
<td>0</td>
<td>1</td>
<td>Write only access (S_WRITE)</td>
</tr>
<tr>
<td>0</td>
<td>1</td>
<td>0</td>
<td>Read/Write access (S_READWRITE)</td>
</tr>
</tbody>
</table>

Bit 3 is reserved and should always be 0. Bits 4-6 specify the file sharing mode of the file to be opened as follows:

<table>
<thead>
<tr>
<th>Bit 6</th>
<th>Bit 5</th>
<th>Bit 4</th>
<th>File Sharing Codes</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0</td>
<td>0</td>
<td>Compatibility Mode (S_COMPAT)</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>If the file’s read-only bit is set, then this is the same as Deny Writes, otherwise it is the same as Deny Read/Writes.</td>
</tr>
<tr>
<td>0</td>
<td>0</td>
<td>1</td>
<td>Deny Read/Writes (S_DENYREADWRITE)</td>
</tr>
<tr>
<td>0</td>
<td>1</td>
<td>0</td>
<td>Deny Writes (S_DENYWRITE)</td>
</tr>
<tr>
<td>0</td>
<td>1</td>
<td>1</td>
<td>Deny Reads (S_DENYREAD)</td>
</tr>
<tr>
<td>1</td>
<td>0</td>
<td>0</td>
<td>Deny None (S_DENYNONE)</td>
</tr>
</tbody>
</table>

Bit 7 (S_INHERIT) is the file’s inheritance flag. If this flag is not set, a child process will inherit any open file handles and has the same access as the parent. If this flag is set, a child must re-open any files it wishes to use and must face the same sharing restrictions other processes must share.

**BINDING**

```
move.w mode,-(sp)
pea fname
move.w #$3D,-(sp)
trap #1
addq.l #8,sp
```

**RETURN VALUE**

Upon return, if the longword is positive, the lower WORD contains the new handle of the open file, otherwise the negative LONG should be regarded as a GEMDOS error code.

**COMMENTS**

Bits 7-3 of mode should be set to 0 unless the ‘_FLK’ cookie is present indicating the presence of the file sharing/record locking extensions to GEMDOS.

**SEE ALSO**

Fclosr(), Fcreate()
Foutstat()

LONG Foutstat( handle )
WORD handle;

Foutstat() determines the output status of a file.

Opcode 262 (0x106)

Availability This function is available under all MiNT versions integrated with MultiTOS.

Parameters handle specifies the GEMDOS file handle of the file to return information about.

Binding
move.w handle,-(sp)
move.w #$106,-(sp)
trap #1
addq.l #4,sp

Return Value Foutstat() returns a 0 or positive number indicating the number of characters which may be written to the specified file without blocking. If an error occurred, Foutstat() returns a negative GEMDOS error code.

Caveats Currently this function always returns 1 for disk files.

See Also Cconos(), Cauxos(), Cprnos(), Fcntl(), Finstat()

Fpipe()

LONG Fpipe( fhandle )
WORD fhandle[2];

Fpipe() creates a pipe named ‘SYS$PIPE.xxx’ (where ‘xxx’ is a three digit integer) on ‘U:\PIPE\’ and returns two file handles to it, one for reading and one for writing.

Opcode 256 (0x100)

Availability Available when a ‘MiNT’ cookie with a version of at least 0.90 exists.

Parameters fhandle is a pointer to an array of two WORDs. If the functions is successful, fhandle[0] will contain an open GEMDOS file handle to the pipe which may be used for reading only. fhandle[1] will contain an open GEMDOS file handle to the pipe which may be used for writing only.

The Atari Compendium
**RETURN VALUE**

`Fpipe()` returns `E_OK` (0) if successful or a negative GEMDOS error code otherwise.

**Caveats**

No more than 999 pipes created with `Fpipe()` may be in use at once.

**Comments**

This function is normally used by shells who wish to redirect the input and output of their child processes. Prior to launching a child process, the shell redirects its input and output (as necessary) to the read and write ends of the newly created pipe.

---

### Fputchar()

**Opcodes**

264 (0x108)

**Availability**

This function is available under all MiNT versions integrated with MultiTOS.

**Parameters**

- `handle` specifies the handle of the file to write a character to.

If the file specified by `handle` is a pseudo-terminal then all four bytes of `lchar` are written (it should be formatted as a character read from `Bconin()`), otherwise only the low byte of `lchar` is transmitted.

`mode` is only valid if `handle` refers to a terminal device. If `mode` is `TTY_COOKED` (0x0001) then control characters (which could cause SIGINT or SIGTSTP signals to be raised) passed through this function will be interpreted and acted upon. Setting `mode` to 0 will cause control characters to have no special effect.

---

**Binding**

```
move.w mode, -(sp)
move.l lchar, -(sp)
move.w handle, -(sp)
move.w #$108, -(sp)
trap #1
```
**RETURN VALUE**

`Fputchar()` returns 4L if the character was output to a terminal, 1L if the character was output to a non-terminal, 0L if the character could not be written (possibly because of flow control), **EIHNDL** (-37) if the handle was invalid, or a negative **BIOS** error code if an error occurred during I/O.

**SEE ALSO**

`Cconout()`, `Cauxout()`, `Crawid()`, `Cprnout()`, `Bconout()`, `Fgetchar()`, `Fwrite()`

---

**Fread()**

`LONG Fread( handle, length, buf )`

**WORD handle;**

**LONG length;**

**VOIDP buf;**

`Fread()` reads binary data from a specified file from the current file pointer.

**_OPCODE**

63 (0x3F)

**AVAILABILITY**

All **GEMDOS** versions.

**PARAMETERS**

`handle` is the **GEMDOS** file handle of the file to read from. `length` specifies the number of bytes of data to read. `buf` is a pointer to a buffer (at least `length` bytes long) where the read data will be stored.

**BINDING**

```assembly
lea 10(sp),sp
lea 12(sp),sp

move.l length,-(sp)
move.w handle,-(sp)
move.w #$3F,-(sp)
trap #1

lea 12(sp),sp
```

**RETURN VALUE**

`Fread()` returns either a positive amount indicating the number of bytes actually read (this number may be smaller than `length` if an **EOF** is hit) or a negative **GEMDOS** error code.

**CAVEATS**

`Fread()` will crash the system if given a parameter of 0 for `length` on **GEMDOS** versions lower than 0.15.

**SEE ALSO**

`Fwrite()`, `Fopen()`, `Fclos()`
Freadlink()

LONG Freadlink( bufsiz, buf, name )
WORD bufsiz;
char *buf, *name;

Freadlink() determines what file the specified symbolic link refers to.

OPCODE
303 (0x12F)

AVAILABILITY
Available when a ‘MiNT’ cookie with a version of at least 0.90 exists.

PARAMETERS
bufsiz specifies the length of buffer buf into which the original file pointed to by the symbolic link name is written.

BINDING
    pea          name
    pea          buf
    move.w      bufsiz,-(sp)
    move.w      #$12F,-(sp)
    trap         #1
    lea          12(sp),sp

RETURN VALUE
Freadlink() returns 0 if successful or a negative GEMDOS error code otherwise.

SEE ALSO
Fsymmlink()

Frename()

LONG Frename( reserved, oldname, newname )
WORD reserved;
char *oldname,*newname;

Frename() renames a standard GEMDOS file. It may also be used to move a file in the tree structure of a physical drive.

OPCODE
86 (0x56)

AVAILABILITY
All GEMDOS versions.

PARAMETERS
reserved is not currently used and should be 0. oldname is the GEMDOS file specification of the file’s current name/location. newname is the GEMDOS file specification of the new name/location of the file.

BINDING
    pea          newname

THE ATARI COMPENDIUM
Fseek() - 2.89

pea oldname
move.w #0,-(sp)
trap #1
lea 10(sp),sp

RETURN VALUE Frename() returns E_OK (0) if the operation was successful or a negative GEMDOS error code if not.

CAVEATS Prior to GEMDOS version 0.15, this command may not be used to rename folders. Also, do not attempt to rename a file that is currently open under any version of GEMDOS.

Fseek()

LONG Fseek( offset, handle, mode )
LONG offset;
WORD handle,mode;

Fseek() moves the file position pointer within a GEMDOS file.

OPCODE 66 (0x42)

AVAILABILITY All GEMDOS versions.

PARAMETERS handle specifies the GEMDOS file handle of the file pointer to modify. The meaning of offset varies with mode as follows:

<table>
<thead>
<tr>
<th>Name</th>
<th>mode</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>SEEK_SET</td>
<td>0</td>
<td>offset specifies the positive number of bytes from the beginning of the file.</td>
</tr>
<tr>
<td>SEEK_CUR</td>
<td>1</td>
<td>offset specifies the negative or positive number of bytes from the current file position.</td>
</tr>
<tr>
<td>SEEK_END</td>
<td>2</td>
<td>offset specifies the positive number of bytes from the end of the file.</td>
</tr>
</tbody>
</table>

BINDING move.w mode,-(sp)
move.w handle,-(sp)
move.l offset,-(sp)
move.w #$42,-(sp)
trap #1
lea 10(sp),sp

RETURN VALUE Fseek() returns a positive value representing the new absolute location of the file pointer from the beginning of the file or a negative GEMDOS error code.
Fselect()

WORD Fselect( timeout, rfds, wfds, reserved )
WORD timeout;
LONG *rfds, *wfds;
LONG reserved;

Fselect() enumerates file descriptors which are ready for reading and/or writing.

OPCODE
285 (0x11D)

AVAILABILITY
This function is available under all MiNT versions integrated with MultiTOS.

PARAMETERS
timeout specifies the maximum amount of time (in milliseconds) to wait for at least one of the specified file descriptors to become unblocked. If timeout is 0 then the process will wait indefinitely.

rfds and wfds each point to a LONG bitmap describing the read and write file descriptors to wait for. Setting bit #10 of the LONG pointed to by rfds, for example, will cause Fselect() to return when GEMDOS handle 10 is available for reading.

As many read or write file descriptors can be specified per call as desired. Specifying NULL for either rfds or wfds is the same as passing a pointer to a LONG with no bits set.

Upon return the LONGs pointed to by rfds and wfds will be filled in with a similar bitmap indicating which handles are ready to be read/written. reserved should always be set to 0L.

BINDING
move.l reserved,-(sp)
pea wfds
pea rfds
move.w timeout,-(sp)
move.w #$11D,-(sp)
trap #1
lea 16(sp),sp

RETURN VALUE
Fselect() returns the sum of bits set in both rfds and wfds. A return value of 0 indicates that the function timed out before any of the specified file handles became available. A negative GEMDOS error code is returned if the function failed.

CAVEATS
Fselect() does not currently work on any BIOS device except the keyboard.

COMMENTS
Fselect( 0L, 0L, 0L, 0L) will block the calling process forever.
**Fsetdta()**

VOID Fsetdta( ndta )
DTA *ndta;

*Fsetdta()* sets the location of a new DTA (Disk Transfer Address) in memory.

**OPCODE**
26 (0x1A)

**AVAILABILITY**
All GEMDOS versions.

**PARAMETERS**
*ndta* is a pointer to a valid memory area which will be used as the new DTA. The DTA structure is defined under the entry for *Fgetdta()*.

**BINDING**
pea ndta
move.w #$1A,-(sp)
trap #1
addq.l #6,sp

**COMMENTS**
When an application starts, its DTA overlaps the command line string in the processes’ basepage. Any use of the *Fsfirst()* or *Fsnext()* call without first reallocating a new DTA will cause the processes’ command line to be corrupted.

To prevent this, you should use *Fsetdta()* to define a new DTA structure for your process prior to using *Fsfirst()* or *Fsnext()*.

**SEE ALSO**
Fgetdta(), Fsfirst(), Fsnext()
**OPCODE** 78 (0x4E)

**AVAILABILITY** All GEMDOS versions.

**PARAMETERS** *fspec* is the GEMDOS file specification of the file or subdirectory to search for. This specification may use wildcard characters (\? or \*) within the filename, however they may not be used within the pathname. This function is the only GEMDOS function which accepts wildcard characters in the path specification.

*attribs* is a bit mask which can combine several file characteristics that further narrows the search as follows:

<table>
<thead>
<tr>
<th>Name</th>
<th>Bit Mask</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>FA_READONLY</td>
<td>0x01</td>
<td>Include files which are read-only.</td>
</tr>
<tr>
<td>FA_HIDDEN</td>
<td>0x02</td>
<td>Include hidden files.</td>
</tr>
<tr>
<td>FA_SYSTEM</td>
<td>0x04</td>
<td>Include system files.</td>
</tr>
<tr>
<td>FA_VOLUME</td>
<td>0x08</td>
<td>Include volume labels.</td>
</tr>
<tr>
<td>FA_DIR</td>
<td>0x10</td>
<td>Include subdirectories.</td>
</tr>
<tr>
<td>FA_ARCHIVE</td>
<td>0x20</td>
<td>Include files with archive bit set.</td>
</tr>
</tbody>
</table>

**BINDING**

```
move.w attribs,-(sp)
pea fspec
move.w #$4E,-(sp)
trap #1
addq.l #8,sp
```

**RETURN VALUE** 

Fsfirst() returns E_OK (0) if a file was found and the DTA was successfully filled in with the file information. Otherwise, it returns a negative GEMDOS error code.

The DTA structure is defined as:

```
typedef struct
{
    BYTE d_reserved[21];
    BYTE d_attrib;
    UWORD d_time;
    UWORD d_date;
    LONG d_length;
    char d_fname[14];
} DTA;
```

**COMMENTS** This function uses the application’s DTA which is initially located in the same memory location as the processes’ command line. Using this function without first assigning a new DTA will corrupt the command line.

When running in the MiNT domain (see Pdomain()), Fsfirst() and Fsnext() will fill in the DTA with lowercase filenames rather than the standard TOS uppercase.
Fsnext()

WORD Fsnext( VOID )

Fsnext() should be called as many times as necessary after a corresponding Fsfirst() call to reveal all files which match the search criteria.

OPCONE 79 (0x4F)

AVAILABILITY All GEMDOS versions.

BINDING

move.w #$4F,-(sp)
trap #1
addq.l #2,sp

RETURN VALUE Fsnext() returns E_OK (0) if another file matching the search criteria given in Fsfirst() is found and the DTA has been properly filled in with the file’s information. Otherwise, a negative GEMDOS error code is returned.

COMMENTS

This function uses the application’s DTA which is initially located in the same memory location as the processes’ command line. Using this function without first assigning a new DTA will corrupt the command line.

This call should only be used after Fsfirst() and the contents of the DTA should not be modified between the calls.

SEE ALSO Fsfirst()

Fsymlink()

LONG Fsymlink( oldname, newname )
char *oldname, *newname;

Fsymlink() creates a symbolic link to a file.

OPCONE 302 (0x12E)

AVAILABILITY Available when a ‘MiNT’ cookie with a version of at least 0.90 exists.

PARAMETERS oldname points to the file specification of the file to create a link to. newname

THE ATARI COMPENDIUM
points to the file specification of the link to create.

**BINDING**

```plaintext
pea    newname
pea    oldname
move.w #$12E,-(sp)
trap   #1
lea    10(sp),sp
```

**RETURN VALUE**

`Fsymlink()` returns 0 if successful or a negative **GEMDOS** error code otherwise.

**COMMENTS**

`Fsymlink()`, unlike `Flink()`, creates symbolic links, which, unlike hard links, can be setup between physical devices and file systems.

An `Fdelete()` call to a symbolic link will delete the link, not the file. A call to `Fdelete()` on the original file will cause future references to the created symbolic link to fail.

**SEE ALSO**

`Flink()`, `Freadlink()`

---

**Fwrite()**

```plaintext
LONG Fwrite( handle, count, buf )
WORD handle;
LONG count;
VOIDP buf;
```

**OPCODE**

64 (0x40)

**AVAILABILITY**

All **GEMDOS** versions.

**PARAMETERS**

`handle` is the handle of the file to write to. `count` specifies the number of bytes to write. `buf` indicates the starting address of the data to write.

**BINDING**

```plaintext
pea    buf
move.l count,-(sp)
move.w handle,-(sp)
trap   #1
lea    10(sp),sp
```

**RETURN VALUE**

`Fwrite()` returns the positive number of bytes actually written or a negative **GEMDOS** error code if the operation failed.

**CAVEATS**

Prior to **GEMDOS** version 0.15, calling `Fwrite()` with a `count` parameter of 0 will hang the system.
Fxattr() returns extended information about the specified file.

**OPCODE**
300 (0x12C)

**AVAILABILITY**
Available when a 'MiNT' cookie with a version of at least 0.90 exists.

**PARAMETERS**
*flag* specifies whether attributes returned by this call on symbolic links should be those of the file to which the link points or the link itself. A value of **FX_FILE** (0) causes the attributes to be those of the actual file whereas a value of **FX_LINK** (1) returns the attributes of the link itself.

*name* specifies the name of the file from which attributes are to be read and placed in the **XATTR** structure pointed to by *xattr*. **XATTR** is defined as follows:

```c
typedef struct
{
    UWORD mode;
    LONG index;
    UWORD dev;
    UWORD reserved1;
    UWORD nlink;
    UWORD uid;
    UWORD gid;
    LONG size;
    LONG blksize;
    LONG nblocks;
    WORD mtime;
    WORD mdate;
    WORD atime;
    WORD adate;
    WORD ctime;
    WORD cdate;
    WORD attr;
    WORD reserved2;
    LONG reserved3;
    LONG reserved4;
} XATTR;
```

**XATTR**’s members have the following meaning:

---

**SEE ALSO**
Fread()
### XATTR Element | Meaning
--- | ---
**mode** | Masking *mode* with 0xF000 reveals the file type as one of the following:
- **S_IFCHR** (0x2000)
- **S_IFDIR** (0x4000)
- **S_IFREG** (0x8000)
- **S_IFIFO** (0xA000)
- **S_IMEM** (0xC000)
- **S_IFLNK** (0xE000)

The lower three nibbles of *mode* is a bit mask which specifies the legal file access mode(s) as defined in `Fchmod()`.

**index** | This member combined with the *dev* field are designed to provide a unique identifier for a file under file systems which allow multiple files with the same filename.

**dev** | This value represents either a **BIOS** device number or an identifier created by the file system to represent a remote **device**.

**reserved1** | This structure element is currently reserved for future implementations of **MiNT**.

**nlink** | This value specifies the current number of hard links attached to the file. On a file system that does not support hard links and for most regular files, *nlink* is 1.

**uid** | *uid* is the user ID of the owner of the file.

**gid** | *gid* is the group ID of the owner of the file.

**size** | *size* is the length of the file in bytes.

**blksize** | *blksize* specifies the size of blocks (in bytes) in this file system.

**nbblocks** | *nbblocks* is the actual number of blocks the file is using on the device. This number may include data storage elements other used to keep track of the file (aside from the actual data).

**mtime, mdate** | Time and date of the last file modification in **GEMDOS** format.

**atime, adate** | Time and date of the last file access in **GEMDOS** format.

**ctime, cdate** | Time and date of the file’s creation in **GEMDOS** format.

**attr** | Standard file attributes (same as read by `Fattrib()`).

**reserved2** | This structure element is currently reserved for future implementations of **MiNT**.

**reserved3** | This structure element is currently reserved for future implementations of **MiNT**.

**reserved4** | This structure element is currently reserved for future implementations of **MiNT**.

### Binding

```
pea xattr
pea name
move.w flag,-(sp)
move.w #$12C,-(sp)
trap #1
lea 12(sp),sp
```

### RETURN VALUE

`Fxattr()` returns 0 if successful or a negative **GEMDOS** error code otherwise.

### SEE ALSO

`Fattrib()`
**Maddalt()**

```c
LONG Maddalt( start, size )
VOIDP start;
LONG size;
```

*Maddalt()* informs *GEMDOS* of the existence of additional ‘alternative’ RAM that would not normally have been identified by the system.

**Opcode**

20 (0x14)

**Availability**

Available as of *GEMDOS* version 0.19 only.

**Parameters**

*start* indicates the starting address for the block of memory to be added to the *GEMDOS* free list. *size* indicates the length of this block in bytes.

**Binding**

```assembly
move.l size,-(sp)
pea start
move.w #$14,-(sp)
trap #1
lea 10(sp),sp
```

**Return Value**

*Maddalt()* returns *E_OK* (0) if the call succeeds or a negative *GEMDOS* error code otherwise.

**Comments**

This call should only be used to identify RAM not normally identified by the *BIOS* at startup (added through a VME-card or hardware modification). Once this RAM has been identified to the system it may not be removed and should only be allocated and used via the standard system calls. In addition, programs wishing to use this RAM must have their alternative RAM load bit set or use *Mxalloc()* to specifically request alternative RAM.

See the discussion earlier in this chapter for more information about the types of available RAM.

**See Also**

*Mxalloc()*
Malloc()

VOIDP Malloc( amount )
LONG amount;

Malloc() requests a block of memory for use by an application.

OPCODE 72 (0x48)

AVAILABILITY All GEMDOS versions.

PARAMETERS amount specifies the amount of memory (in bytes) you wish to allocate. You may pass a value of -1L in which case the function will return the size of the largest free block of memory.

BINDING

move.l amount,-(sp)
mov.w #$48,-(sp)
trap #1
addq.l #6,sp

RETURN VALUE Malloc() returns NULL if there is no block large enough to fill the request or a pointer to the block if the request was satisfied. The memory allocated will be chosen based on the status of the processes’ load flags. To specify the memory requirements in more detail, use Mxalloc().

CAVEATS Prior to GEMDOS version 0.15, Malloc( 0L ) will return a pointer to invalid memory as opposed to failing as it should.

COMMENTS Because GEMDOS can only allocate a limited amount of blocks per process (as few as 20 depending on the version of GEMDOS), applications should limit their usage of this call by allocating a few large blocks instead of many small blocks or use a ‘C’ memory manager (like malloc() ) if possible.

SEE ALSO Mxalloc()

Mfree()

WORD Mfree( startadr )
VOIDP startadr;

Mfree() releases a block of memory previously reserved with Malloc() or Mxalloc() back into the GEMDOS free list.

OPCODE 73 (0x49)
**Mshrink()**

**WORD Mshrink( startadr, newsize )**

**VOIDP startadr;**

**LONG newsize;**

Mshrink() releases a portion of a block’s memory to the GEMDOS free list.

**OPCODE**

74 (0x4A)

**AVAILABILITY**

All GEMDOS versions.

**PARAMETERS**

startadr is the address of the block whose size you wish to decrease. newsize is the length you now desire for the block.

**BINDING**

move.l newsize,-(sp)  
pea startadr  
clr.w -(sp) // Required/Reserved Value  
move.w #$4A,-(sp)  
trap #1  
lea 12(sp),sp

**RETURN VALUE**

Mshrink() returns E_OK (0) if the operation was successful or a negative GEMDOS error code otherwise.

**CAVEATS**

This call should be used only to ‘shrink’ a memory block, not to enlarge it.

**SEE ALSO**

Malloc(), Mxalloc(), Mfree()
Mxalloc()

VOIDP Mxalloc( amount, mode )
LONG amount;
WORD mode;

Mxalloc() allocates a block of memory according to specified preferences.

_OPCODE 68 (0x44)

_AVAILABILITY Available from GEMDOS version 0.19.

_PARAMETERS amount specifies the length (in bytes) of the block requested. As with Malloc(), specifying -1L for amount will return the size of the largest block of memory available. With modes 0 or 1, the size of the largest block of available RAM from the specified type of RAM is returned. Modes 2 and 3 return the size of the largest available block or whichever type of RAM had the largest block.

mode is a WORD bit array which specifies the type of memory requested as follows:

<table>
<thead>
<tr>
<th>Bit</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>0-1</td>
<td>Bits 0-1 represent a possible value of 0-3 representing the type of RAM to allocate as follows:</td>
</tr>
<tr>
<td>Name</td>
<td>Value</td>
</tr>
<tr>
<td>MX_STRAM</td>
<td>0</td>
</tr>
<tr>
<td>MX_TTRAM</td>
<td>1</td>
</tr>
<tr>
<td>MX_PREFSTRAM</td>
<td>2</td>
</tr>
<tr>
<td>MX_PREFTTRAM</td>
<td>3</td>
</tr>
<tr>
<td>2</td>
<td>Not used (should be set to 0).</td>
</tr>
<tr>
<td>3</td>
<td>If set, refer to bits 4-7 for memory protection advice, otherwise default to protection specified in program header. This bit is only valid in the presence of MiNT.</td>
</tr>
<tr>
<td>4-7</td>
<td>Bits 4-7 represent a possible value of 0-7 representing the memory protection mode to place on the allocated block of memory. Currently valid values are:</td>
</tr>
<tr>
<td>Name</td>
<td>Value</td>
</tr>
<tr>
<td>MX_HEADER</td>
<td>0</td>
</tr>
<tr>
<td>MX_PRIVATE</td>
<td>1</td>
</tr>
<tr>
<td>MX_GLOBAL</td>
<td>2</td>
</tr>
<tr>
<td>MX_SUPERVISOR</td>
<td>3</td>
</tr>
<tr>
<td>MX_READABLE</td>
<td>4</td>
</tr>
<tr>
<td>8-15</td>
<td>Not used (should be set to 0).</td>
</tr>
</tbody>
</table>

These bits are only consulted if bit 3 is set and MiNT is present.
Pause()

VOID Pause( VOID )

Pause() suspends the process until a signal is received.

OPCODE
289 (0x121)

AVAILABILITY
This function is available under all MiNT versions integrated with MultiTOS.

BINDING
move.w #$121,-(sp)
trap #1
addq.l #2,sp

COMMENTS
If the signal handler does a ‘C’ longjmp() to a different point in the process or if the handler’s purpose is to exit the process, this call will never return.

SEE ALSO
Psigblock(), Psignal(), Psigsetmask()

Pdomain()

WORD Pdomain( domain )
WORD domain;

Pdomain() determines/modifies the calling processes’ execution domain.

OPCODE
281 (0x119)

AVAILABILITY
This function is available under all MiNT versions integrated with MultiTOS.

PARAMETERS
domain contains the domain code of the new process domain. Currently the only
valid values are `DOMAIN_TOS` (0) for the TOS compatibility domain and `DOMAIN_MINT` (1) for the MiNT domain. Passing a negative value for `domain` will not change domains but it will return the current domain.

**BINDING**

```
move.w   domain,-(sp)
move.w   #$119,-(sp)
trap     #1
addq.l   #4,sp
```

**RETURN VALUE**

`Pdomain()` returns the domain in effect prior to the call.

**COMMENTS**

Process domain affects system calls like `Fread()`, `fwrite()`, `Fsfirst()`, and `Fsnext()`. Processes behave as expected when under the TOS domain.

When processes run under the MiNT domain, however, the behavior of `Fread()` and `fwrite()` calls when dealing with terminals can be modified by `Fcntl()`. Also, `Fsfirst()` and `Fsnext()` may not necessarily return the standard DOS 8 + 3 file name format. MiNT domain processes must understand filenames formatted for different file systems.

**SEE ALSO**

`Fcntl()`

---

### Pexec()

LONG `Pexec( mode, fname, cmdline, envstr )`

WORD `mode;`

char *`fname`, *`cmdline`, *`envstr`;

`Pexec()` has many functions designed to spawn child processes depending on the selected mode.

**OPCODE**

75 (0x4B)

**AVAILABILITY**

`Pexec()` modes 0, 4, and 5, are available in all GEMDOS versions. Mode 6 is available as of GEMDOS version 0.15. Mode 6 is available as of GEMDOS version 0.19. Modes 100, 104, 106, and 200 are only available in the presence of MiNT.

**PARAMETERS**

`mode` defines the function of `Pexec()` and the meaning of its parameters and return value as defined below. For modes which load a program, `fname` specifies the GEMDOS file specification of the file to load. `cmdline` is pointer to a string containing the command line which will be passed to the calling program. The first byte of the string should indicate the length of the command line (maximum of 125 bytes). The actual command line starts at byte 2. `envstr` is a pointer to an environment which is copied and assigned to the child process. If `envstr` is NULL,
the child inherits a copy of the parent’s environment.

<table>
<thead>
<tr>
<th>Name</th>
<th>mode</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>PE_LOADGO</td>
<td>0</td>
<td>‘LOAD AND GO’ - Load and execute named program file and return a WORD exit code when the child terminates.</td>
</tr>
<tr>
<td>PE_LOAD</td>
<td>3</td>
<td>‘LOAD, DON’T GO’ - Load named program. If successful, the LONG return value is the starting address of the child processes' basepage. The parent owns the memory of the child’s environment and basepage and must therefore free them when completed with the child.</td>
</tr>
<tr>
<td>PE_GO</td>
<td>4</td>
<td>‘JUST GO’ - Execute process with basepage at specified address. With this mode, fname and envstr are NULL. The starting address of the basepage of the process to execute is given in the cmdline parameter.</td>
</tr>
<tr>
<td>PE_BASEPAGE</td>
<td>5</td>
<td>‘CREATE BASEPAGE’ - This mode allocates the largest block of free memory and creates a basepage in the first 256 bytes of it. fname should be set to NULL. It is the responsibility of the parent to load or define the child’s code, shrink the memory block as necessary, and initialize the basepage pointers to the TEXT, DATA, and BSS segments of the program. With MiNT, use of this mode in conjunction with mode PE_CGO can be used to emulate the Pvfork() call without blocking the parent.</td>
</tr>
<tr>
<td>PE_GOTHENFREE</td>
<td>6</td>
<td>‘JUST GO, THEN FREE’ - This mode is identical to mode PE_GO except that memory ownership of the child’s environment and basepage belong to the child rather than the parent so that when the child Pterm()’s, that memory is automatically freed.</td>
</tr>
<tr>
<td>PE_CLOADGO</td>
<td>100</td>
<td>‘LOAD, GO, DON’T WAIT’ - This mode is identical to mode PE_LOADGO except that the parent process is returned to immediately while the child continues to execute. The positive process ID of the child is returned. Environment and basepage memory blocks are freed automatically when the child Pterm()’s.</td>
</tr>
<tr>
<td>PE_CGO</td>
<td>104</td>
<td>‘JUST GO, DON’T WAIT’ - This mode is similar to mode PE.GO except that the parent process is returned to immediately while the child continues to execute concurrently. The positive process ID of the child is returned. Memory ownership of the environment and basepage are shared by the parent and child (this sharing extends to all memory owned by the parent). fname may be used to supply a name for the child, otherwise, if NULL is used, the name of the parent will be used. cmdline should point to the process basepage. envstr should be NULL.</td>
</tr>
<tr>
<td>PE_NOSHARE</td>
<td>106</td>
<td>‘JUST GO, DON’T WAIT, NO SHARING’ - This mode is exactly the same as mode PE_CGO except that the child process owns its own environment and basepage sharing no memory with the parent.</td>
</tr>
</tbody>
</table>
PE_REPLACE

<table>
<thead>
<tr>
<th>PE_REPLACE</th>
<th>200</th>
</tr>
</thead>
<tbody>
<tr>
<td>'REPLACE PROGRAM AND GO' - This mode works like mode PE_CLOADGO except that the parent process is terminated immediately and the child process completely replaces the parent in memory retaining the same process ID. fname, cmdline, and envstr, are all normally passed and valid.</td>
<td></td>
</tr>
</tbody>
</table>

**BINDING**

pea envstr
pea cmdline
pea fname
move.w word,-(sp)
move.w #$4B,-(sp)
trap #1
lea 16(sp),sp

**RETURN VALUE**

The value returned by Pexec() is dependent on the mode value and is therefore explained above. All Pexec() modes return a LONG negative GEMDOS error code when the call fails. A WORD negative value indicates the child was successfully run but it terminated returning a negative error code. In all cases, a process returning after having been interrupted with CTRL-C returns 0x0000FFE0 (-32).

**COMMENTS**

Command lines longer than 126 bytes may be passed to processes aware of the Atari Extended Command Line Specification (see discussion earlier in this chapter).

**SEE ALSO**

shel_write()

---

**Pfork()**

**WORD Pfork( VOID )

Pfork() creates a copy of the current process.**

**OPCODE**

283 (0x11B)

**AVAILABILITY**

This function is available under all MiNT versions integrated with MultiTOS.

**BINDING**

move.w #$11B,-(sp)
trap #1
addq.l #2,sp

**RETURN VALUE**

Pfork() returns the new process ID in the parent and a 0 in the child.

**CAVEATS**

If the parent is in supervisor mode when this call is made, the child is started in user mode anyway.

---

**THE ATARI COMPENDIUM**
After a \texttt{Pfork()} call, two instances of one process will exist in memory. Program execution in both processes continue at the same point in the TEXT segment following this call. The parent’s DATA and BSS segments are physically copied so that any variables that change in the child will not affect the parent and vice versa.

New processes started with this call should not call \texttt{Mshrink()} but are required to do any GEM initialization such as \texttt{appl\_init()} and \texttt{v\_opnvwk()} again (if GEM usage is needed). Both the parent and child use \texttt{Pterm()} or \texttt{Pterm0()} to terminate themselves.

\textbf{SEE ALSO} \texttt{Pexec()}, \texttt{Pvfork()}

\section*{Pgetegid()}

\textbf{WORD Pgetegid(VOID)}

\texttt{Pgetegid()} returns the effective group ID of the process.

\textbf{OPCODE} 313 (0x139)

\textbf{AVAILABILITY} Available when a ‘MiNT’ cookie with a version of at least 0.95 exists.

\textbf{BINDING}

\begin{verbatim}
move.w #139,-(sp)
trap #1
addq.l #2,sp
\end{verbatim}

\textbf{COMMENTS} The effective group ID of a process will be different than its actual group ID if its set gid bit is set. This mechanism allows users to grant file access to other users.

\textbf{SEE ALSO} \texttt{Pgetgid()}, \texttt{Pgeteuid()}

\section*{Pgeteuid()}

\textbf{WORD Pgeteuid(VOID)}

\texttt{Pgeteuid()} returns the effective user ID of the process.

\textbf{OPCODE} 312 (0x138)

\textbf{AVAILABILITY} Available when a ‘MiNT’ cookie with a version of at least 0.95 exists.

\textbf{BINDING}

\begin{verbatim}
move.w #138,-(sp)
trap #1
\end{verbatim}
addq.l #2,sp

**COMMENTS**
The effective group ID of a process will be different than its actual group ID if its set gid bit is set. This mechanism allows users to grant file access to other users.

**SEE ALSO**
Pgetuid(), Pgetegid()

---

**Pgetgid()**

**WORD Pgetgid( VOID )**

Pgetgid() returns the group ID (0-255) of the calling process.

**OPCODE**
271 (0x10F)

**AVAILABILITY**
This function is available under all MiNT versions integrated with MultiTOS.

**BINDING**
move.w #$10F,-(sp)
trap #1
addq.l #2,sp

**SEE ALSO**
Psetgid()

---

**Pgetpgrp()**

**WORD Pgetpgrp( VOID )**

Pgetpgrp() returns the process group ID code for the calling process.

**OPCODE**
269 (0x10D)

**AVAILABILITY**
This function is available under all MiNT versions integrated with MultiTOS.

**BINDING**
move.w #$10D,-(sp)
trap #1
addq.l #2

**COMMENTS**
Process groups are closely related processes which are used for job control and signaling purposes. Process groups usually terminate together rather than one at a time.

**SEE ALSO**
Psetpgrp(), Pkill()
Pgetpid()

WORD Pgetpid( VOID )

Pgetpid() returns the positive WORD process ID code for the calling process. This identifier uniquely identifies the process within the system.

OPCODE 267 (0x10B)

AVAILABILITY This function is available under all MiNT versions integrated with MultiTOS.

BINDING

move.w #$10B,-(sp)
trap  #1
addq.l #2,sp

Pgetppid()

WORD Pgetppid( VOID )

Pgetppid() returns the process ID for the calling processes’ parent.

OPCODE 268 (0x10C)

AVAILABILITY This function is available under all MiNT versions integrated with MultiTOS.

BINDING

move.w #$10C,-(sp)
trap  #1
addq.l #2,sp

RETURN VALUE Pgetppid() returns the process ID code for the parent of the calling process or 0 if it was started by the kernel (not a child process).

Pgetuid()

WORD Pgetuid( VOID )

Pgetuid() returns the user ID code (0-255) of the calling process which determines access permissions and can be used in a multi-user system to differentiate users.

OPCODE 271 (0x10F)

AVAILABILITY This function is available under all MiNT versions integrated with MultiTOS.
Pkill() sends a signal to one or more processes.

**OPCODE**

273 (0x111)

**AVAILABILITY**

This function is available under all MiNT versions integrated with MultiTOS.

**PARAMETERS**

Pkill() sends signal \textit{sig} to certain processes based on the value of \textit{pid}. If \textit{pid} is positive, the signal is sent to the process with process identifier \textit{pid}. If \textit{pid} is 0, the signal is sent to all processes who belong to the same process group as the caller as well as the caller itself. If \textit{pid} is negative, the signal is sent to all processes with process group number \textit{-pid}.

**BINDING**

\begin{verbatim}
move.w sig,-(sp)
move.w pid,-(sp)
move.w #$111,-(sp)
trap #1
addq.l #6,sp
\end{verbatim}

**RETURN VALUE**

Pkill() returns 0 if successful or a negative GEMDOS error code otherwise.

**COMMENTS**

If the caller is also a recipient of a signal and that signal causes program termination this call will never return.

**SEE ALSO**

Psignal()
Pmsg()

WORD Pmsg( mode, mboxid, msgptr )
WORD mode;
LONG mboxid;
PMSG *msgptr;

Pmsg() sends/receives a message to/from a ‘message box’.

Opcode

293 (0x125)

Availability

Available when a ‘MiNT’ cookie with a version of at least 0.90 exists.

Parameters

mode specifies the action to take as follows:

<table>
<thead>
<tr>
<th>Name</th>
<th>mode</th>
<th>Operation</th>
</tr>
</thead>
<tbody>
<tr>
<td>MSG_READ</td>
<td>0</td>
<td>Block the process and don’t return until a message is read from the specified mailbox ID mboxid and placed in the structure pointed to by msgptr.</td>
</tr>
<tr>
<td>MSG_WRITE</td>
<td>1</td>
<td>Block the process and don’t return until a process waiting for a message with mailbox ID mboxid has received the message contained in the structure pointed to by msgptr.</td>
</tr>
<tr>
<td>MSG_READWRITE</td>
<td>2</td>
<td>Block the process until a process waiting for a message with mailbox ID mboxid has received the message contained in the structure pointed to by msgptr and a return message is received with mailbox ID 0xFFFFxxxx where ‘xxxx’ is the process ID of the current process.</td>
</tr>
</tbody>
</table>

PMSG is defined as:

```
typedef struct
{
  LONG userlong1;
  LONG userlong2;
  WORD pid;
} PMSG;
```

On return from writes, pmsg.pid contains the process ID of the process who read your message, on return from reads, its the process ID of the writer. The contents of userlong1 and userlong2 is completely up to the sender.

By OR’ing mode with MSG_NOWAIT (0x8000), you can prevent the call from blocking the process and simply return -1 if another process wasn’t waiting to
read or send your process a message.

**BINDING**

```assembly
pea msgptr
move.l mboxid,-(sp)
move.w mode,-(sp)
move.w #$125,-(sp)
trap #1
lea 12(sp),sp
```

**RETURN VALUE**

`Pmsg()` returns 0 if successful, -1 if bit 0x8000 is set and no process was ready to receive/send the desired message, or a negative **GEMDOS** error code.

## Pnice()

**WORD** `Pnice( delta )`

**WORD** `delta;`

`Pnice()` alters the process priority of the calling process.

**OPCODE**

266 (0x10A)

**AVAILABILITY**

This function is available under all **MiNT** versions integrated with **MultiTOS**.

**PARAMETERS**

`delta` is a signed number which is added to the current process priority value. Positive values decrease process priority while negative values increase it.

**BINDING**

```assembly
move.w delta,-(sp)
move.w #$10A,-(sp)
trap #1
addq.l #4,sp
```

**RETURN VALUE**

`Pnice()` returns the prior process priority.

**COMMENTS**

The process priority value has no fixed formula so it is hard to be able to predict the results of this call with any accuracy. This call is the same as `Prenice( Pgetpid(), delta )`.

**SEE ALSO**

`Prenice()`
Prenice()

LONG Prenice( pid, delta )
WORD pid, delta;

Prenice() adjusts the process priority of the specified process.

OPCODE  295 (0x127)

AVAILABILITY Available when a ‘MiNT’ cookie with a version of at least 0.90 exists.

PARAMETERS The process priority for the process with process ID pid is adjusted by signed value delta. Positive values for delta decrease process priority while negative values increase it.

BINDING
  move.w     delta, -(sp)
  move.w     pid, -(sp)
  move.w     #$127,-(sp)
  trap       #1
  addq.l     #6

RETURN VALUE Prenice() returns a 32-bit negative GEMDOS error code if unsuccessful. Otherwise, the lower 16-bit signed value can be interpreted as the previous process priority code.

COMMENTS The exact effect adjusting process priorities will have is difficult to determine.

SEE ALSO Pnice()

Prusage()

VOID Prusage( rusg )
LONG *rusg;

Prusage() returns resource information about the current process.

OPCODE  286 (0x11E)

AVAILABILITY This function is available under all MiNT versions integrated with MultiTOS.

PARAMETERS rusg is a pointer to an array of 8 LONGs as follows:
### PRU KERNELTIME
- **Name**: PRU_KERNELTIME
- **Meaning**: Time spent by process in MiNT kernel.

### PRU PROCESSTIME
- **Name**: PRU_PROCESSTIME
- **Meaning**: Time spent by process in its own code.

### PRU CHILDKERNELTIME
- **Name**: PRU_CHILDKERNELTIME
- **Meaning**: Total MiNT kernel time spent by children of this process.

### PRU CHILDPROCESSTIME
- **Name**: PRU_CHILDPROCESSTIME
- **Meaning**: Total user code time spent by children of this process.

### PRU MEMORY
- **Name**: PRU_MEMORY
- **Meaning**: Total memory allocated by process (in bytes).

#### Binding
- `pea rusg
move.w #$11E,-(sp)
trap #1
addq.l #6,sp`

#### Comments
- All times given are in milliseconds.

#### See Also
- `Psemaphore()`

## Psemaphore()

```assembly
LONG Psemaphore( mode, id, timeout )
WORD mode;
LONG id;
LONG timeout;
```

**Psemaphore()** creates a semaphore which may only be accessed by one process at a time.

### Opcode
- 308 (0x134)

### Availability
- Available when a ‘MiNT’ cookie with a version of at least 0.92 exists.

### Parameters
- **mode** specifies the mode of the operation which affects the other two parameters as follows:

<table>
<thead>
<tr>
<th>Name</th>
<th>mode</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>SEM_CREATE</td>
<td>0</td>
<td>Create a semaphore with called id and grant ownership to the calling process. timeout is ignored.</td>
</tr>
<tr>
<td>SEM_DESTROY</td>
<td>1</td>
<td>Destroy the semaphore called id. This only succeeds if the semaphore is owned by the caller. timeout is ignored.</td>
</tr>
</tbody>
</table>
Psetgid() sets the group ID of the calling process.

**OPCODE**

277 (0x115)

**AVAILABILITY**

This function is available under all MiNT versions integrated with MultiTOS.

**PARAMETERS**

*gid* is the group ID code to assign the calling process (0-255).

**BINDING**

move.w gid,-(sp)
move.w #$115,-(sp)
trap #1
addq.l #4,sp

**RETURN VALUE**

Psetgid() returns gid if successful or EACCDN (-36) if the process did not have the authority to change the group ID.

**COMMENTS**

The group ID of a process may only be changed when it is currently 0. Therefore, once the group ID has been set, it is fixed and unchangeable. Further attempts to modify it will result in an EACCDN error.
**Psetlimit()**

LONG Psetlimit( limit, value )
WORD limit;
LONG value;

Psetlimit() reads/modifies resource allocation limits for the calling process and all of its children.

**OPCODE**

287 (0x11F)

**AVAILABILITY**

This function is available under all MiNT versions integrated with MultiTOS.

**PARAMETERS**

*limit* defines the resource to read or modify as follows:

<table>
<thead>
<tr>
<th>Name</th>
<th>limit</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>LIM_MAXTIME</td>
<td>1</td>
<td>Maximum CPU time in milliseconds. If <em>value</em> is positive, <em>value</em> determines the new maximum. If <em>value</em> is 0, then the limit is set at 'unlimited'. If <em>value</em> is negative, the current value is returned but not modified.</td>
</tr>
<tr>
<td>LIM_MAXMEM</td>
<td>2</td>
<td>Maximum total memory allowed for process. If <em>value</em> is positive, <em>value</em> determines the new maximum. If <em>value</em> is 0, then the limit is set at 'unlimited'. If <em>value</em> is negative, the current value is returned but not modified.</td>
</tr>
<tr>
<td>LIM_MAXMALLOC</td>
<td>3</td>
<td>Maximum total size of each Malloc (Mxalloc). If <em>value</em> is positive, <em>value</em> determines the new maximum. If <em>value</em> is 0, then the limit is set at 'unlimited'. If <em>value</em> is negative, the current value is returned but not modified.</td>
</tr>
</tbody>
</table>

**BINDING**

move.l value,-(sp)
move.w limit,-(sp)
move.w #$11F,-(sp)
trap #1
addq.l #8,sp

**RETURN VALUE**

Psetlimit() returns the previous value or ERANGE (-64) if the value for *limit* was out of range.

**COMMENTS**

The limits imposed by Psetlimit() are inherited from the parent by child processes.

**SEE ALSO**

Prusage()
Psetpgrp()

LONG Psetpgrp( pid, newgrp )
WORD pid, newgrp;

Psetpgrp() sets the process group ID of the specified process.

OPCODE 270 (0x10E)

AVAILABILITY This function is available under all MiNT versions integrated with MultiTOS.

PARAMETERS The process group ID of the process with process ID pid will have its process group ID changed to newgrp if the calling process has the same user ID or is the parent of the specified process. If pid is 0, the process group ID of the current process is sent. If newgrp is 0, the process group ID is set to equal the processes’ (not the callers’ unless pid is also set to 0) process ID.

BINDING

move.w newgrp,-(sp)
move.w pid,-(sp)
move.w #$10E,-(sp)
trap #1
addq.l #6,sp

RETURN VALUE Psetpgrp() returns newgrp if successful or a negative GEMDOS error code otherwise.

SEE ALSO Pgetpgrp()

Psetuid()

WORD Psetuid( uid )
WORD uid;

Psetuid() sets the user ID of the calling process.

OPCODE 272 (0x110)

AVAILABILITY This function is available under all MiNT versions integrated with MultiTOS.

PARAMETERS uid is the user ID to assign to the calling process.

BINDING

move.w uid,-(sp)
move.w #$110,-(sp)
trap #1
addq.l #4,sp

THE ATARI COMPENDIUM
**RETURN VALUE**

Psetuid() returns *uid* if successful or a negative GEMDOS error code otherwise.

**COMMENTS**

As with the process group ID, the user ID of a process may only be set if it is currently 0. This means that once the user ID is set, it may not be changed.

**SEE ALSO**
Pgetuid()

---

**Psigaction()**

LONG Psigaction( *sig*, *act*, *oact* )

WORD *sig;*

SIGACTION **act, oact;**

Psigaction() specifies a default action for the specified signal.

**OPCODE**

311 (0x137)

**AVAILABILITY**

Available when a ‘MiNT’ cookie with a version of at least 0.95 exists.

**PARAMETERS**

*sig* specifies the signal whose action you wish to change. *act* points to a SIGACTION structure (as defined below) which defines the handling of future signals of type *sig*. *oact* points to a SIGACTION structure which defines the handling of pending signals of type *sig*.

```c
typedef struct {
    LONG sa_handler;
    WORD sa_mask;
    WORD sa_flags;
} SIGACTION;
```

Setting *sa_handler* to SIG_DFL (0) will cause the default action to take place for the signal. A value of SIG_IGN (1) will cause the signal to be ignored. Any other value specifies the address of a signal handler.

The signal handler should expect one LONG argument on its stack which contains the signal number being delivered. During execution of the handler, all signals specified in *sa_mask* are blocked.

*sa_flags* is a signal-specific flag. When *sig* is SIGCHLD, setting Bit #0 (SA_NOCLDSTOP) will cause the SIGCHLD signal to be delivered only when the child process terminated (not when stopped).

**BINDING**

move.w    sig,-(sp)
pea       act

---

**THE ATARI COMpendium**
Psigblock() - 2.117

Psigblock() blocks selected signals from delivery.

LONG Psigblock( mask )
LONG mask;

Psigblock() returns the original set of blocked signals in effect prior to the call.

OPCODE 278 (0x116)

availability This function is available under all MiNT versions integrated with MultiTOS.

PARAMETERS mask is a bit mask of signals block. For each bit n set, signal n is added to the ‘blocked’ list.

BINDING
move.l mask,-(sp)
move.w #$116,-(sp)
trap #1
addq.l #6,sp

RETURN VALUE

Psigblock() returns 0 if successful or a negative GEMDOS error code otherwise.

 COMMENTS
Calling Psigaction() automatically unmaps the specified signal for delivery.

SEE ALSO Psignal

Psigblock()

LONG Psigblock( mask )
LONG mask;

Psigblock() blocks selected signals from delivery.

OPCODE 278 (0x116)

availability This function is available under all MiNT versions integrated with MultiTOS.

PARAMETERS mask is a bit mask of signals block. For each bit n set, signal n is added to the ‘blocked’ list.

BINDING
move.l mask,-(sp)
move.w #$116,-(sp)
trap #1
addq.l #6,sp

RETURN VALUE

Psigblock() returns 0 if successful or a negative GEMDOS error code otherwise.

 COMMENTS
Calling Psigaction() automatically unmaps the specified signal for delivery.

SEE ALSO Psignal

Psigblock()
Psignal()

LONG Psignal( sig, handler )
WORD sig;
VOID (*handler)( LONG );

Psignal() determines the action taken when a signal is received by the process.

OPCODE 274 (0x112)

AVAILABILITY This function is available under all MiNT versions integrated with MultiTOS.

PARAMETERS sig specifies the signal whose response you wish to modify. If handler is cast to SIG_DFL (0) then the default action for the signal will occur when received. If handler is cast to SIG_IGN (1) then the signal will be ignored by the process. Otherwise, handler points to a user function which is designed to take action on a signal. This function is called when a signal is received with a LONG signal number on the stack.

BINDING

pea handler
move.w sig,-(sp)
move.w #$112,-(sp)
trap #1
addq.l #8,sp

RETURN VALUE Psignal() returns the old value of the signal handler if successful or a negative GEMDOS error code otherwise.

COMMENTS Signal handler functions may make any GEMDOS, BIOS, or XBIOS calls desired but must not make any AES or VDI calls. Signal handlers must either return with a 680x0 RTS instruction to resume program execution or call Psigreturn() to clean the stack if it intends to do a “C” longjmp().

Signal handling is preserved across Pfork() and Pvfork() calls. Child processes started with Pexec() ignore and follow the default action the same as their parents. Signals which have user functions assigned to them are reset to the default action for child processes.

SEE ALSO Psigreturn(), Psigblock(), Pkill()
Psigpause()

LONG Psigpause( mask )
LONG mask;

Psigpause() sets a new signal mask and then suspends the process until a signal is received.

Opcode 310 (0x136)

Availability Available when a ‘MiNT’ cookie with a version of at least 0.95 exists.

Parameters mask specifies the signal mask to wait for.

Binding
move.l mask, -(sp)
move.w #$136, -(sp)
trap #1
addq.l #6, sp

Return Value Psigpause() returns 0 if successful or non-zero otherwise.

Comments Depending on the state of the signal handler, this call may never return.

See Also Psigaction(), Pause()

Psigpending()

LONG Psigpending( VOID )

Psigpending() indicates which signals have been sent but not yet delivered to the calling process.

Opcode 291 (0x123)

Availability This function is available under all MiNT versions integrated with MultiTOS.

Binding
move.w #123, -(sp)
trap #1
addq.l #2, sp

Return Value Psigpending() returns a bit mask of which signals have been sent but not yet delivered to the calling process because they are being blocked. For each bit n set in the returned LONG, signal n is waiting for reception.
Psigreturn()

VOID Psigreturn( VOID )

**Psigreturn()** prepares exit from a signal handler not planning to return via a 680x0 RTS.

**OPCODE** 282 (0x11A)

**AVAILABILITY** This function is available under all MiNT versions integrated with MultiTOS.

**BINDING**
- move.w #$11A,-(sp)
- trap #1
- addq.l #2,sp

**CAVEATS** Calling this function and then calling the 680x0 RTS opcode to return will produce undesired results.

**COMMENTS** **Psigreturn()** is only needed by ‘C’ programs which intend to exit the signal handler by doing a ‘C’ `longjmp()` rather than simply using the 680x0 RTS.

**SEE ALSO** Psignal()

Psigsetmask()

LONG Psigsetmask( mask )

LONG mask;

**Psigsetmask()** defines which signals are to be blocked before being delivered to the calling application.

**OPCODE** 279 (0x117)

**AVAILABILITY** This function is available under all MiNT versions integrated with MultiTOS.

**PARAMETERS** *mask* is a LONG bit mask which defines which signals to block and which signals to allow. For each bit *n* set, signal *n* will be blocked. For each bit *n* clear, signal *n* will be delivered.

**BINDING**
- move.l mask,-(sp)
- move.w #$117,-(sp)
- trap #1
addq.l #6,sp

RETURN VALUE
Psigsetmask() returns the original mask of blocked/unblocked signals prior to the call or a negative GEMDOS error code.

COMMENTS
Unlike Psigblock(), mask completely replaces the old mask rather than simply OR’ing it.

SEE ALSO
Pkill(), Psignal(), Psigpending()

Pterm()

VOID Pterm( retcode )
WORD retcode;

Pterm() terminates an application returning the specified error code.

OPCODE
76 (0x4C)

AVAILABILITY
All GEMDOS versions.

PARAMETERS
retcode indicates the error status upon termination. Some recommended return values are:

<table>
<thead>
<tr>
<th>Name</th>
<th>retcode</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>TERM_OK</td>
<td>0</td>
<td>Program completion without errors</td>
</tr>
<tr>
<td>TERM_ERROR</td>
<td>1</td>
<td>Generic Error</td>
</tr>
<tr>
<td>TERM_BADPARAMS</td>
<td>2</td>
<td>Bad parameters</td>
</tr>
<tr>
<td>TERM_CRASH</td>
<td>-1</td>
<td>Process crashed (returned by GEMDOS versions from 0.15.)</td>
</tr>
<tr>
<td>TERM_CTRL-C</td>
<td>-32</td>
<td>Process terminated by CTRL-C</td>
</tr>
</tbody>
</table>

BINDING
move.w retcode,-(sp)
move.w #$4C,-(sp)
trap #1
addq.l #4,sp

RETURN VALUE
Pterm() never returns.

COMMENTS
GEMDOS jumps through the etv_term (0x102) vector when this call is made prior to process termination to allow the process one last chance to clean up. In addition, all files opened by the process are closed and all memory blocks allocated by the process are freed.
SEE ALSO Pexec(), Pterm0()

### PtermØ()

VOID PtermØ( VOID )

**PtermØ()** terminates the application returning an exit code of 0 indicating no errors.

**OPCODE** 0 (0x00)

**AVAILABILITY** All GEMDOS versions.

**BINDING**

```
clr.w -(sp)
trap #1
```

**RETURN VALUE** PtermØ() never returns.

**COMMENTS** Same as Pterm().

**SEE ALSO** Pterm()

### Ptermres()

VOID Ptermres( keep, retcode )

LONG keep;

WORD retcode;

**Ptermres()** terminates a process leaving a portion of the program’s TPA intact and removing the memory left from GEMDOS’s memory list.

**OPCODE** 49 (0x31)

**AVAILABILITY** All GEMDOS versions.

**PARAMETERS**

- *keep* is the length (in bytes) of the processes’ TPA to retain in memory after exit.
- *retcode* is the code returned on exit.

**BINDING**

```
move.w retcode,-(sp)
movl keep,-(sp)
move.w #$31,-(sp)
trap #1
addq.l #8,sp
```
Pumask() never returns.

This function is normally used by TSR’s to stay resident in memory. Any files opened by the process are closed. Any memory allocated is, however, retained.

The value for *keep* is usually the sum of the length of the basepage (0x100), the length of the text, data, and bss segments of the application, and the length of the stack. It is important to note that the memory retained by this call may not be freed at a later point as it is removed from the GEMDOS memory list altogether.

See Also  Pterm0(), Pterm()

---

**Pumask()**

WORD Pumask( mode )

WORD mode;

Pumask() defines an initial file and directory creation mask.

**Opcode**  307 (0x133)

**Availability**  Available when a ‘MiNT’ cookie with a version of at least 0.92 exists.

**Parameters**  *mode* specifies the new file access permission mask to apply to all future files created with Fcreate() and Dcreate(). *mode* is a WORD bit mask of various access permission flags as defined in Fchmod().

**Binding**

<table>
<thead>
<tr>
<th>Binding</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>move.w</td>
<td><em>mode</em>, -(sp)</td>
</tr>
<tr>
<td>move.w</td>
<td>#$133, -(sp)</td>
</tr>
<tr>
<td>trap</td>
<td>#1</td>
</tr>
<tr>
<td>addq.l</td>
<td>#4, sp</td>
</tr>
</tbody>
</table>

**Return Value**  Pumask() returns the original mask in effect prior to the call.

**See Also**  Dcreate(), Fcreate(), Fchmod()

---

**Pusrval()**

LONG Pusrval( val )

LONG val;

Pusrval() reads/modifies a user defined value associated with a process.
2.124 – GEMDOS Function Reference

**OPCODE**

280 (0x118)

**AVAILABILITY**

This function is available under all MiNT versions integrated with MultiTOS.

**PARAMETERS**

val specifies the new value of the LONG associated with this process. If val is -1 then this value is not changed but still returned.

**BINDING**

move.w #$118,-(sp)
trap #1
addq.l #2,sp

**RETURN VALUE**

Pusrval() returns the original value of the user LONG prior to the call.

**COMMENTS**

The user-defined longword set by this call is inherited by child processes and may be utilized as desired.

---

**Pvfork()**

WORD Pvfork( VOID )

Pvfork() creates a duplicate of the current process which shares address and data space with the parent.

**OPCODE**

275 (0x113)

**AVAILABILITY**

This function is available under all MiNT versions integrated with MultiTOS.

**BINDING**

move.w #$113,-(sp)
trap #1
addq.l #2,sp

**RETURN VALUE**

Pvfork() returns the new process ID to the parent and 0 to the child. If an error occurs the parent receives a negative GEMDOS error code.

**CAVEATS**

If the parent is in supervisor mode when this call is made the child is placed in user mode anyway.

**COMMENTS**

The child process spawned by this function shares all address and data space with the parent. In other words, any variables altered by the parent will also be altered by the child and vice versa. The child process should not call Mshrink() as its TPA is already correctly sized.

The two processes do not execute concurrently. The parent is blocked until either the child terminates or calls Pexec()’s mode 200.
Pwait()

LONG Pwait( VOID )

Pwait() attempts to determine the exit code of a stopped or terminated child process.

OPCODE 265 (0x109)

AVAILABILITY This function is available under all MiNT versions integrated with MultiTOS.

BINDING

move.w #$109,-(sp)
trap #1
addq.l #2,sp

RETURN VALUE Pwait() returns 0 if no child processes have terminated or a 32-bit return code for a child process which has been terminated or stopped.

The process ID of the child process is placed in the upper 16 bits. A process which returned an exit status (via Pterm(), Ptermres(), or Pterm0() ) returns the exit code in the lower 16 bits.

A process which was stopped as the result of a signal returns 0xnn7F where nn is the signal number which stopped it. A process which was terminated as the result of a signal returns 0xnn00 where nn is the signal number which killed the process.

COMMENTS Pwait() will block the calling process until at least one child has been stopped or terminated. Once the exit code of a process has been returned with this call it will be not be returned again with this call (unless it had been stopped and is restarted and stopped again). This call is identical to Pwait3( 2, NULL );

SEE ALSO Pexec(), Pterm(), Ptermres(), Pterm0()
OPCODE

284 (0x11C)

AVAILABILITY

This function is available under all MiNT versions integrated with MultiTOS.

PARAMETERS

flag is a bit mask which specifies the specifics of this call as follows:

<table>
<thead>
<tr>
<th>Name</th>
<th>Mask</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>PW_NOBLOCK</td>
<td>0x01</td>
<td>If set, the function will not block the calling process if no child has been stopped or terminated, rather it will simply return 0. If clear, the process will be blocked until a child of the process has terminated or is stopped.</td>
</tr>
<tr>
<td>PW_STOPPED</td>
<td>0x02</td>
<td>If set, return exit codes for processes which have been terminated as well as stopped. If clear, only return exit codes for processes which have actually terminated.</td>
</tr>
</tbody>
</table>

rusage points to an array of two LONGs which are filled in with resource usage information of the stopped or terminated process. The first LONG contains the number of milliseconds used by the child in user code. The second LONG indicates the number of milliseconds spent by the process in the kernel. rusage may be set to NULL if this information is undesired.

BINDING

```
pea       rusage
move.w    flag,-(sp)
trap      #1
addq.l    #6,sp
```

RETURN VALUE

Pwait3() returns 0 if no child processes have been stopped and/or terminated (depending on flag) or a 32-bit return code for a child process which has been terminated or stopped.

The process ID of the child process is placed in the upper 16 bits. A process which returned an exit status (via Pterm(), Ptermres(), or Pterm0()) returns the exit code in the lower 16 bits.

A process which was stopped as the result of a signal returns 0xnn7F where nn is the signal number which stopped it. A process which was terminated as the result of a signal returns 0xnn00 where nn is the signal number which killed the process.

SEE ALSO

Pwait(), Pexec(), Pterm(), Pterm0(), Ptermres(), Prusage()
Pwaitpid()

LONG Pwaitpid(pid, flag, rusage)
WORD pid, flag;
LONG *rusage;

Pwaitpid() returns exit code information about one or more child processes.

OPCODE 314 (0x13A)

AVAILABILITY Available when a ‘MiNT’ cookie with a version of at least 0.96 exists.

PARAMETERS pid specifies the children whose exit codes are of interest as follows.

A pid of PWP_ALL (-1) indicates that all children are of interest. A pid of less than -1 indicates that any child whose process group is -pid is of interest. A pid of PWP_GROUP (0) indicates that any child with the same process group ID of the parent is of interest. A pid greater than 0 indicates that the child with the given process ID is of interest.

For the usage of flag and rusage see Pwait3().

BINDING

pea          rusage
move.w      flag,-(sp)
move.w      #$13A,-(sp)
trap        #1
addq.l      #8,sp

RETURN VALUE See Pwait3().

SEE ALSO Pwait(), Pwait3()

Salert()

VOID Salert(str)
char *str;

Salert() sends an alert string to the alert pipe ‘U:\PIPE\ALERT’. 

OPCODE 316 (0x13C)

AVAILABILITY Available when a ‘MiNT’ cookie with a version of at least 0.98 exists.

PARAMETERS str should point to a NULL terminated character string containing the alert

THE ATARI COMPENDIUM
message to display. The message should not contain any carriage returns or escape characters. The string should not be formatted as in `form_alert()`.

**Binding**

```assembly
pea str
move.w #$13C,-(sp)
trap #1
addq.l #6,sp
```

**Caveats**

Messages sent by `Salert()` are only delivered if a separate application is present which was designed to listen to the alert pipe and post its contents.

**See Also**

`form_alert()`

---

**Super()**

```c
VOIDP Super( stack )
VOIDP stack;
```

`Super()` allows you to interrogate or alter the state of the 680x0.

**Opcode**

32 (0x20)

**Availability**

All GEMDOS versions.

**Parameters**

`stack` defines the meaning of the call as follows:

<table>
<thead>
<tr>
<th>Name</th>
<th><code>stack</code></th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>SUP_SET</td>
<td>(VOIDP)0</td>
<td>The processor is placed in supervisor mode and the old supervisor stack is returned.</td>
</tr>
<tr>
<td>SUP_INQUIRE</td>
<td>(VOIDP)1</td>
<td>This interrogates the current mode of the processor. If the processor is in user mode a <code>SUP_USER</code> (0) is returned, otherwise a <code>SUP_SUPER</code> (1) is returned.</td>
</tr>
<tr>
<td>—</td>
<td>&gt;1</td>
<td>The processor is placed in user mode and the supervisor stack is reset to <code>stack</code>.</td>
</tr>
</tbody>
</table>

**Binding**

```assembly
pea stack
move.w #$20,-(sp)
trap #1
addq.l #6,sp
```

**Return Value**

`Super()` returns a different value based on the `stack` parameter. The various return values are explained above.

**Caveats**

You should never call the AES in supervisor mode. In addition, supervisor mode should be entered and left in the same stack context (same `C` function) or stack corruption can result.
**Comments**

To execute portion of a program in supervisor mode you normally call `Super()` with a parameter of 0 and save the return value. When ready to return to user mode you call `Super()` again with the saved return value as a parameter.

Supervisor mode should be used sparingly under MiNT as no task switching can occur.

**See Also**

`Supexec()`

---

**Sversion()**

**UWORD** `Sversion( VOID )`

`Sversion()` returns the current GEMDOS version number.

**Opcode**

48 (0x30)

**Availability**

All GEMDOS versions.

**Binding**

```
move.w #$30,-(sp)
trap #1
addq.l #2,sp
```

**Return Value**

`Sversion()` returns a `UWORD` containing the GEMDOS minor version number in the upper word and the major version number in the lower word. Current values returned by Atari TOS’s are:

<table>
<thead>
<tr>
<th>Return Value</th>
<th>TOS versions (normally) found in:</th>
</tr>
</thead>
<tbody>
<tr>
<td>0x1300 (0.13)</td>
<td>TOS 1.0, TOS 1.02</td>
</tr>
<tr>
<td>0x1500 (0.15)</td>
<td>TOS 1.04, TOS 1.06</td>
</tr>
<tr>
<td>0x1700 (0.17)</td>
<td>TOS 1.62</td>
</tr>
<tr>
<td>0x1900 (0.19)</td>
<td>TOS 2.01, TOS 2.05, TOS 2.06, TOS 3.01, TOS 3.05, TOS 3.06</td>
</tr>
<tr>
<td>0x3000 (0.30)</td>
<td>TOS 4.00, TOS 4.01, TOS 4.02, TOS 4.03, TOS 4.04, MultiTOS 1.00, MultiTOS 1.08</td>
</tr>
</tbody>
</table>

**Comments**

The GEMDOS number is not associated with the TOS or AES version number. You should check for GEMDOS or MiNT version numbers when trying to determine the presence or properties of a GEMDOS function.
Syield()

VOID Syield( VOID )

Syield() surrenders the remainder of the callers’ current process timeslice.

OPCODE 255 (0xFF)

AVAILABILITY This function is available under all MiNT versions integrated with MultiTOS.

BINDING

```assembly
move.w #$FF,-(sp)
trap #1
addq.l #2,sp
```

SEE ALSO Pause(), Fselect()

Sysconf()

LONG Sysconf( inq )
WORD inq;

Sysconf() returns information about the limits or capabilities of the currently running version of MiNT.

OPCODE 290 (0x122)

AVAILABILITY This function is available under all MiNT versions integrated with MultiTOS.

PARAMETERS

<table>
<thead>
<tr>
<th>Name</th>
<th>inq</th>
<th>Return Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>SYS_MAXINQ</td>
<td>-1</td>
<td>Maximum legal value for inq.</td>
</tr>
<tr>
<td>SYS_MAXREGIONS</td>
<td>0</td>
<td>Maximum memory regions per process.</td>
</tr>
<tr>
<td>SYS_MAXCOMMAND</td>
<td>1</td>
<td>Maximum length of Pexec() command string.</td>
</tr>
<tr>
<td>SYS_MAXFILES</td>
<td>2</td>
<td>Maximum number of open files per process.</td>
</tr>
<tr>
<td>SYS_MAXGROUPS</td>
<td>3</td>
<td>Maximum number of supplementary group ID’s.</td>
</tr>
<tr>
<td>SYS_MAXPROCS</td>
<td>4</td>
<td>Maximum number of processes per user.</td>
</tr>
</tbody>
</table>

BINDING

```assembly
move.w inq,-(sp)
move.w #$122,-(sp)
trap #1
addq.l #4,sp
```
RETURN VALUE
See above.

COMMENTS
If the requested item returns UNLIMITED (0x7FFFFFFF) then that item is unlimited.

SEE ALSO
Dpathconf()

Talarm()

LONG Talarm( time )
LONG time;

Talarm() reads/sets a process alarm for the current process.

OPCODE
288 (0x120)

AVAILABILITY
This function is available under all MiNT versions integrated with MultiTOS.

PARAMETERS
 time specifies the length of time (in milliseconds) to wait before a SIGALRM signal is delivered. If time is 0 then any previously set alarm is cancelled. If time is negative the function does not modify any alarm currently set.

BINDING
move.l time,-(sp)
move.w #$120,-(sp)
trap #1
addq.l #6,sp

RETURN VALUE
Talarm() returns 0 if no alarm was scheduled prior to this call or the amount of time remaining (in milliseconds) before the alarm is triggered.

CAVEATS
An alarm with less than 1000 remaining milliseconds will return a value of 0.

COMMENTS
If no SIGALRM signal handler has been set up when the alarm is triggered, the process will be killed.

SEE ALSO
Pause(), Psignal()

Tgetdate()

UWORD Tgetdate( VOID )

Tgetdate() returns the current GEMDOS date.
2.132 – GEMDOS Function Reference

**OPCODE**
42 (0x2A)

**AVAILABILITY**
All GEMDOS versions.

**BINDING**
```assembly
move.w #$2A,-(sp)
trap #1
addq.l #2,sp
```

**RETURN VALUE**
Tgetdate() returns a bit array **UWORD** arranged as follows:

<table>
<thead>
<tr>
<th>Bits 15-9</th>
<th>Bits 8-5</th>
<th>Bits 4-0</th>
</tr>
</thead>
<tbody>
<tr>
<td>Years since 1980</td>
<td>Month (1-12)</td>
<td>Date (0-31)</td>
</tr>
</tbody>
</table>

**SEE ALSO**
Tgettime(), Tsetdate(), Gettime()

---

### Tgettime()

**UWORD** Tgettime( **VOID** )

Tgettime() returns the GEMDOS system time.

**OPCODE**
44 (0x2C)

**AVAILABILITY**
All GEMDOS versions.

**BINDING**
```assembly
move.w #$2C,-(sp)
trap #1
addq.l #2,sp
```

**RETURN VALUE**
Tgettime() returns a bit array arranged as follows:

<table>
<thead>
<tr>
<th>Bits 15-11</th>
<th>Bits 10-5</th>
<th>Bits 4-0</th>
</tr>
</thead>
<tbody>
<tr>
<td>Hour (0-23)</td>
<td>Minute (0 to 59)</td>
<td>Secs/2 (0 to 29)</td>
</tr>
</tbody>
</table>

**SEE ALSO**
Tgetdate(), Tsettime(), Gettime()

---

### Tsetdate()

**WORD** Tsetdate( **date** )

**UWORD** date;

Tsetdate() sets the current GEMDOS date.

THE ATARI COMPENDIUM
Tsettime()

WORD Tsettime(time)
UWORD time;

Tsettime() sets the current GEMDOS time.

OPCODE 45 (0x2D)

AVAILABILITY All GEMDOS versions.

PARAMETERS time is a bit array arranged as illustrated under Tgettime().

BINDING

move.w time,-(sp)
move.w #$2D,-(sp)
trap #1
addq.l #4,sp

RETURN VALUE Tsettime() returns 0 if the time was set or non-zero if the time given was invalid.

CAVEATS GEMDOS version 0.13 did not inform the BIOS of the date change and hence would not change the IKBD date or the date of a battery backed-up clock.

SEE ALSO Tgettime(), Tsetdate(), Settime()
Overview

The Basic Input/Output System (BIOS) is responsible for the lowest level of communications between the operating system and hardware devices. This chapter will document the operating system functions of the BIOS and other system level operations.

System Startup

Upon a cold or warm boot\(^1\), microprocessors in the 680x0 series load the initial supervisor stack pointer from the first longword in memory (\$0) and begin execution at the PC found in the second longword (\$4). The location this points to is the base initialization point for Atari computers.

Every Atari computer follows a predefined set of steps to accomplish system initialization. The following illustrates these steps leaving out some hardware initialization which is specific to the particular computer line (ST, TT, Falcon, etc.).

- The Interrupt Priority Level (IPL) is set to 7 and the OS switches to supervisor mode.
- A RESET instruction is executed to reset external hardware devices.
- The presence of a diagnostic cartridge is determined. If one is inserted, it is JMP’ed to with a return address in register A6.
- If running on a 68030, the CACR, VBR, TC, TT0, and TT1 registers are initialized.
- If a floating-point coprocessor is present it is initialized.
- If the memvalid ($420), memval2 ($43A), and memval3 ($51A) system variables are all valid, a warm boot is assumed and the memory controller is initialized with the value from memcntrl ($424).
- The initial color palette registers are loaded and the screen base is initialized to \$100000.
- Memory is sized if it wasn’t from a previous reset.
- Magic numbers are stored in low memory to indicate the successful sizing and initialization of memory.
- System variables and the cookie jar are initialized.
- The BIOS initialization point is executed.
- Installed cartridges of type 2 are executed.

\(^1\)A cold boot occurs when the computer system experiences a total loss of power and no memory locations can be considered valid (this can be done artificially by zeroing memory, as is the case with the CTRL-ALT-RSHIFT-DELETE reset). A warm boot is a manual restart of the system which can be accomplished via software (like the CTRL-ALT-DELETE reset) or the external reset button found on some machines.
• The screen resolution is programmed.
• Installed cartridges of type 0 are executed.
• Interrupts are enabled by lowering the IPL to 3.
• Installed cartridges of type 1 are executed.
• The **GEMDOS** initialization point is executed.
• On systems running **TOS** 2.06 or **TOS** 3.06 and above, the Fuji logo is displayed and a memory test and hard disk spin-up sequence is executed.
• If at least one floppy drive is attached to the system, the first sector of the first floppy drive is loaded, and if executable, it is called.
• If at least one hard disk or other media is attached to the system, the first sector of each is loaded in succession until one with an executable sector is found or each has been tried.
• If a hard disk sector was found that was executable, it is executed.
• The text cursor is enabled.
• All “\AUTO\*.*PRG” files found on the boot disk are executed.
• If _cmdload ($482) is 0 then an environment string is created and the **AES** is launched, otherwise “COMMAND.PRG” is loaded.
• If the **AES** ever terminates, the system is reset and system initialization begins again.

**OS Header**

The address of the start of operating system is stored in the system variable `_sysbase` ($4F2). The beginning of the operating system contains a table with contents as follows:

<table>
<thead>
<tr>
<th>Offset (<code>_sysbase</code> + $x)</th>
<th>Size</th>
<th>Contents</th>
</tr>
</thead>
<tbody>
<tr>
<td>$0</td>
<td>WORD</td>
<td><code>os_entry</code>: BRA to reset handler (shadowed at $0).</td>
</tr>
<tr>
<td>$2</td>
<td>WORD</td>
<td><code>os_version</code>: <strong>TOS</strong> version number. The high byte is the major revision number, and the low byte is the minor revision number.</td>
</tr>
<tr>
<td>$4</td>
<td>LONG</td>
<td><code>reseth</code>: Pointer to the system reset handler.</td>
</tr>
<tr>
<td>$8</td>
<td>LONG</td>
<td><code>os_beg</code>: Base address of the OS (same as <code>_sysbase</code>).</td>
</tr>
<tr>
<td>$C</td>
<td>LONG</td>
<td><code>os_end</code>: Address of the first byte of RAM not used by the operating system.</td>
</tr>
<tr>
<td>$10</td>
<td>LONG</td>
<td><code>os_rsv1</code>: Reserved</td>
</tr>
<tr>
<td>$14</td>
<td>LONG</td>
<td><code>os_magic</code>: Pointer to the <strong>GEM</strong> Memory Usage Parameter Block (MUPB). See below for more information.</td>
</tr>
<tr>
<td>$18</td>
<td>LONG</td>
<td><code>os_date</code>: Date of system build ($YYYYMMDD).</td>
</tr>
<tr>
<td>$1C</td>
<td>WORD</td>
<td><code>os_conf</code>: OS Configuration Bits. See below for more information.</td>
</tr>
<tr>
<td>$1E</td>
<td>LONG</td>
<td><code>os_dosdate</code>: <strong>GEMDOS</strong> format date of system build.</td>
</tr>
</tbody>
</table>
$20 \text{ LONG} \quad p\_root: \text{ Pointer to a system variable containing the address of the GEMDOS memory pool structure. This entry is available as of TOS 1.2. The location pointed to by this value should never be modified by an application.}

$24 \text{ LONG} \quad p\_kbshift: \text{ Pointer to a system variable which contains the address of the system keyboard shift state variable. See below for more information. This entry is available as of TOS 1.02. This location should never be modified by an application.}

$28 \text{ LONG} \quad p\_run: \text{ Pointer to a system variable which contains the address of the currently executing GEMDOS process. See below for more information. This entry is available as of TOS 1.02. The information pointed to by this variable should never be modified by an application.}

$2C \text{ LONG} \quad p\_rsv2: \text{ Reserved}

Some versions of AHDI (the Atari Hard Disk Interface) contain a bug which copies the system header to RAM and then corrupts some portions of it. The following ‘C’ structure definition defines the OSHEADER structure. The function GetROMSysbase() can be used to return an OSHEADER pointer to the code in ROM. GetROMSysbase() will execute properly in either user or supervisor mode.

```c
typedef struct _osheader
{
    UWORD os_entry;
    UWORD os_version;
    VOID *reseth;
    struct _osheader *os_beg;
    char *os_end;
    char *os_rsv1;
    char *os_magic;
    LONG os_date;
    UWORD os_conf;
    UWORD os_dosdate;

    /* Available as of TOS 1.02 */
    char **p_root;
    char **p_kbshift;
    char **p_run;
    char *p_rsv2;
} OSHEADER;

#define _sysbase ((OSHEADER **)0x4F2)

OSHEADER *
GetROMSysbase( VOID )
{
    OSHEADER *osret;
    char *savesp = (Super(SUP_INQUIRE) ? NULL : Super(SUP_SET));
    osret = (*_sysbase)->os_beg;
    if( savesp )
        Super( savesp );
    return osret;
}
```

---

**THE ATARI COMPENDIUM**
OS Configuration Bits

`os_conf` contains the country code and video sync mode that the operating system was compiled for. Bit #0 of this variable is 0 to indicate NTSC video mode or 1 to indicate PAL. The remaining bits, when shifted right by one bit, yield the country code as follows:

<table>
<thead>
<tr>
<th><code>os_conf &gt;&gt; 1</code></th>
<th>Country</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>USA</td>
</tr>
<tr>
<td>1</td>
<td>Germany</td>
</tr>
<tr>
<td>2</td>
<td>France</td>
</tr>
<tr>
<td>3</td>
<td>United Kingdom</td>
</tr>
<tr>
<td>4</td>
<td>Spain</td>
</tr>
<tr>
<td>5</td>
<td>Italy</td>
</tr>
<tr>
<td>6</td>
<td>Sweden</td>
</tr>
<tr>
<td>7</td>
<td>Switzerland (French)</td>
</tr>
<tr>
<td>8</td>
<td>Switzerland (German)</td>
</tr>
<tr>
<td>9</td>
<td>Turkey</td>
</tr>
<tr>
<td>10</td>
<td>Finland</td>
</tr>
<tr>
<td>11</td>
<td>Norway</td>
</tr>
<tr>
<td>12</td>
<td>Denmark</td>
</tr>
<tr>
<td>13</td>
<td>Saudi Arabia</td>
</tr>
<tr>
<td>14</td>
<td>Holland</td>
</tr>
<tr>
<td>15</td>
<td>Czechoslovakia</td>
</tr>
<tr>
<td>16</td>
<td>Hungary</td>
</tr>
<tr>
<td>127</td>
<td>All countries are supported. As of TOS 4.0 the OS is compiled with text for all languages and switches between them based on the country code stored in non-volatile RAM. Use the `.AKP' cookie to determine the actual language in use.</td>
</tr>
</tbody>
</table>

GEM Memory Usage Parameter Block

The pointer at offset $14$ in the OS header points to the GEM Memory Usage Parameter Block which is defined as follows:

```c
typedef struct
{
   /* $87654321 if GEM present */
   LONG gem_magic;

   /* End address of OS RAM usage */
   LONG gem_end;

   /* Execution address of GEM */
   LONG gem_entry;
} MUPB;
```

GEM is only launched at system startup if `gem_magic` is $87654321$. The XBIOS call `Puntaes()` also uses this information to restart the operating system after clearing GEM (only if disk-based). It verifies that `gem_magic` was valid and that GEM was in RAM, then it modifies `gem_magic` and restarts the operating system.
Keyboard Shift State Variable

The OS header entry \texttt{p\_kbshift} provides a method of reading the state of the keyboard shift state variables more quickly than with \texttt{Kbshift()}. This header entry did not exist in TOS 1.0. The following code provides an acceptable method for accessing this variable in all TOS versions:

\begin{verbatim}
#define Kbstate *p_kbshift
char *p_kbshift;

VOID
init_kbshift( VOID )
{
    /* See above for GetROMSysbase() definition. */
    OSHEADER *os = GetROMSysbase();
    if ( os->os_version == 0x0100)
        p_kbshift = (char *)0xE1BL;
    else
        p_kbshift = *(char **)os->p_kbshift;
}
\end{verbatim}

Currently Running Process

The OS header entry \texttt{p\_run} is used to locate the address of the basepage of the currently running process. This entry has only existed as of TOS 1.02 and should never be modified. The following routine returns the address of the basepage of the currently running process in all versions of TOS:

\begin{verbatim}
#define SPAIN 4
typedef long PID

PID *
get_run()
{
    OSHEADER *os = GetROMSysbase();
    if(os->os_version < 0x0102)
    {
        if(( os->os_conf >> 1 ) == SPAIN)
            return (PID *)0x873C;
        else
            return (PID *)0x602C;
    }
    else
        return (PID *)(os->p_run);
}
\end{verbatim}
The Cookie Jar

Overview

The ‘Cookie Jar’ is a structure in memory containing entries called ‘cookies’ which are placed in the ‘jar’ by the operating system or Terminate and Stay Resident (TSR) applications. Applications can test for the presence of a cookie to determine the presence of a hardware device or system feature.

The location of the cookie jar is determined by the address contained in the system variable _p_cookies ($5A0). If no cookie jar has been allocated yet, this entry will contain NULL (0).

Structure

The variable _p_cookies points to multiple COOKIE structures as defined below:

```c
typedef struct
{
    LONG cookie;
    LONG value;
} COOKIE;
```

The structure member cookie contains a value that hopefully uniquely identifies the cookie. cookie values are 4-byte packed longword identifiers (often a 4 letter ASCII code word). Entries with the high byte equal to $5F, the underscore character, are reserved for use by Atari.

The structure member value may contain any value meaningful to an application or no value at all. In some cases a cookie won’t have a meaningful value and its presence simply signals the existence of another process or system feature. TSR’s often use value to store a pointer to an internal structure. The operating system uses cookies to signal the availability of hardware devices or system features.

The end of the cookie jar is signaled with a final entry with the value for cookie equaling NULL. The value entry for this final cookie contains the number of entries possible without reallocating the jar.

Searching for a Cookie

The following code may be used to find a cookie in the cookie jar. It returns 0 if an error occurred or 1 if successful. If p_value is non-NULL on entry, the address it points to will be filled in with the value of the cookie.

```c
WORD getcookie( target, p_value )
LONG target;
LONG *p_value;
{
    char *oldssp;
    COOKIE *cookie_ptr;

    oldssp = (Super(SUP_INQUIRE) ? NULL : Super(1L));
```
cookie_ptr = *(COOKIE **)0x5A0;

if(oldssp)
    Super( oldssp );

if(cookie_ptr != NULL)
{
    do
    {
        if(cookie_ptr->cookie == target)
        {
            if(p_value != NULL)
                *p_value = cookie_ptr->value;
            return 1;
        }
    } while((cookie_ptr++)->cookie != 0L);

    return 0;
}

Placing a Cookie

Only TSR programs should place cookies in the cookie jar. The cookie these programs place should either signal a function provided by the TSR or the presence of an expansion device. A CPX, desk accessory, or standard application should not place cookies in the jar.

To place a cookie, the TSR must first locate the current location of the cookie jar. It is possible that a cookie jar does not exist ( _p_cookies == 0 ). In that case, a new jar should be allocated.

In most instances, the cookie jar should be allocated in increments of 8 slots (though it is not a requirement). In addition, if the process installs a new cookie jar in a TOS version lower than 1.06 it is also the processes responsibility to remove it upon a warm reset. Calling the following code after installing the cookie jar for the first time will ensure that the cookie jar pointer is properly reset on a warm boot.

RESMAGIC equ $31415926
_resvalid equ $426
_resvector equ $42A
_p_cookies equ $5A0
.globl _unjar

_unjar:
    move.l _resvalid,valsav
    move.l _resvector,vecsav
    move.l #reshand,_resvector
    move.l #RESMAGIC,_resvalid
    rts

reshand:
    clr.l _p_cookies
    move.l vecsav,_resvector
    move.l valsav,_resvalid
    jmp (a6)
.bss
After determining the location of the cookie jar, the application should search for the first empty slot in the jar by looking for a NULL in the cookie field of a slot. Next, the application must determine if this is the last slot in the jar by comparing the entry in the value field of the current cookie to the number of the actual slot you are comparing. For instance, if you have found NULL as the value for cookie in slot 16 and value is equal to 16, the jar is full and must be reallocated.

If the slot found is not the last one, the application can simply copy the current slot to the next slot and insert its own cookie.

If the jar must be reallocated, you should allocate enough memory to increase the size of the cookie jar, copy the old entries to the new jar, insert your entry as the last cookie in the jar, and finally terminate the jar with a cookie containing a NULL and the new number of slots you have allocated.

Though not mentioned previously, it is also advisable to ensure that your cookie isn’t already in the jar before placing it to avoid two cookies for multiple executions of the same application to appear.

System Cookies

As of TOS 1.06, the operating system will place several cookies in the cookie jar to inform applications of certain operating system and hardware capabilities as follows:

<table>
<thead>
<tr>
<th>cookie</th>
<th>value</th>
</tr>
</thead>
<tbody>
<tr>
<td>_CPU</td>
<td>The low WORD of the CPU cookie contains a number representing the processor installed in the system as follows:</td>
</tr>
<tr>
<td></td>
<td>Value</td>
</tr>
<tr>
<td>0</td>
<td>68000</td>
</tr>
<tr>
<td>10</td>
<td>68010</td>
</tr>
<tr>
<td>20</td>
<td>68020</td>
</tr>
<tr>
<td>30</td>
<td>68030</td>
</tr>
</tbody>
</table>

| _VDO   | This cookie represents the revision of the video shifter present. The low WORD represents the minor revision number and the high WORD represents the major revision number. Currently valid values are: |
|        | Major | Minor | Shifter |
| 0      | ST    |
| 1      | STe   |
| 2      | TT030 |
| 3      | Falcon030 |
### _FPU

This cookie identifies the presence of floating-point math capabilities in the system. A non-zero low **WORD** indicates the presence of software floating point support (no specific values have yet been assigned). The high **WORD** indicates the type of coprocessor currently connected to the system as follows:

<table>
<thead>
<tr>
<th>Value</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>No FPU is installed.</td>
</tr>
<tr>
<td>1</td>
<td>SFP004</td>
</tr>
<tr>
<td>2</td>
<td>68881 or 68882</td>
</tr>
<tr>
<td>3</td>
<td>68881 or 68882 and SFP004</td>
</tr>
<tr>
<td>4</td>
<td>68881</td>
</tr>
<tr>
<td>5</td>
<td>68881 and SFP004</td>
</tr>
<tr>
<td>6</td>
<td>68882</td>
</tr>
<tr>
<td>7</td>
<td>68882 and SFP004</td>
</tr>
<tr>
<td>8</td>
<td>68040 Internal</td>
</tr>
<tr>
<td>9</td>
<td>68040 Internal and SFP004</td>
</tr>
</tbody>
</table>

### _FDC

This cookie indicates the capability of the currently connected floppy drive. The lowest three bytes is a code indicating the origin of the unit ('ATC' is an Atari unit). The upper byte is a value indicating the highest density floppy present as follows:

<table>
<thead>
<tr>
<th>Value</th>
<th>Density</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>360 Kb/ 720 Kb</td>
</tr>
<tr>
<td>1</td>
<td>1.44 Mb</td>
</tr>
<tr>
<td>2</td>
<td>2.88 Mb</td>
</tr>
</tbody>
</table>

### _SND

This cookie contains a bitmap of sound features available to the system as follows:

<table>
<thead>
<tr>
<th>Bit</th>
<th>Feature</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>GI Sound Chip (PSG)</td>
</tr>
<tr>
<td>1</td>
<td>Stereo 8-bit Playback</td>
</tr>
<tr>
<td>2</td>
<td>DMA Record (w/ XBIOS)</td>
</tr>
<tr>
<td>3</td>
<td>16-bit CODEC</td>
</tr>
<tr>
<td>4</td>
<td>DSP</td>
</tr>
</tbody>
</table>

### _MCH

This cookie indicates the machine type with the major revision number in the high **WORD** and the minor revision number in the low **WORD** as follows:

<table>
<thead>
<tr>
<th>Major</th>
<th>Minor</th>
<th>Shifter</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0</td>
<td>ST</td>
</tr>
<tr>
<td>1</td>
<td>0</td>
<td>STe</td>
</tr>
<tr>
<td>1</td>
<td>8</td>
<td>ST Book</td>
</tr>
<tr>
<td>1</td>
<td>16</td>
<td>Mega STe</td>
</tr>
<tr>
<td>2</td>
<td>0</td>
<td>TT030</td>
</tr>
<tr>
<td>3</td>
<td>0</td>
<td>Falcon030</td>
</tr>
</tbody>
</table>

### _SWI

On machines that contain internal configuration dip switches, this value specifies their positions as a bitmap. Dip switches are generally used to indicate the presence of additional hardware which will be represented by other cookies.

### _FRB

This cookie is present when alternative RAM is present. It points to a 64k buffer that may be used by DMA device drivers to transfer memory between alternative RAM and ST RAM for DMA operations.

### _FLK

The presence of this cookie indicates that file and record locking extensions to **GEMDOS** exist. The **value** field is a version number currently undefined.
<table>
<thead>
<tr>
<th>Cookie</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>_NET</td>
<td>This cookie indicates the presence of networking software. The cookie value points to a structure which gives manufacturer and version information as follows:</td>
</tr>
<tr>
<td></td>
<td>```c</td>
</tr>
<tr>
<td></td>
<td>struct netinfo</td>
</tr>
<tr>
<td></td>
<td>{</td>
</tr>
<tr>
<td></td>
<td>LONG publisher;</td>
</tr>
<tr>
<td></td>
<td>LONG version;</td>
</tr>
<tr>
<td></td>
<td>}</td>
</tr>
<tr>
<td></td>
<td>```</td>
</tr>
<tr>
<td></td>
<td>Bits #0-7 contain the ASCII code of the date separator. Bits #8-11 contain a value indicating the date display format as follows:</td>
</tr>
<tr>
<td></td>
<td><img src="Value.png" alt="Value" /> <img src="Meaning.png" alt="Meaning" /></td>
</tr>
<tr>
<td></td>
<td>Bits #12-15 contain a value indicating the time format as follows:</td>
</tr>
<tr>
<td></td>
<td><img src="Value.png" alt="Value" /> <img src="Meaning.png" alt="Meaning" /></td>
</tr>
<tr>
<td></td>
<td>Note: The value of this cookie does not affect any of the internal time functions. It is intended for informational use by applications only.</td>
</tr>
<tr>
<td>_IDT</td>
<td>This cookie defines the currently configured date and time format.</td>
</tr>
<tr>
<td></td>
<td>Bits #0-7 contain the ASCII code of the date separator. Bits #8-11 contain a value indicating the date display format as follows:</td>
</tr>
<tr>
<td></td>
<td><img src="Value.png" alt="Value" /> <img src="Meaning.png" alt="Meaning" /></td>
</tr>
<tr>
<td></td>
<td>Bits #12-15 contain a value indicating the time format as follows:</td>
</tr>
<tr>
<td></td>
<td><img src="Value.png" alt="Value" /> <img src="Meaning.png" alt="Meaning" /></td>
</tr>
<tr>
<td>_AKP</td>
<td>This cookie indicates the presence of an Advanced Keyboard Processor. The high word of this cookie is currently reserved. The low word indicates the language currently used by TOS for keyboard interpretation and alerts. See the explanation for the country code in the OS header earlier in this chapter for valid values.</td>
</tr>
<tr>
<td></td>
<td>If this cookie is present on TOS 5.0 and higher then the system supports soft-loaded keyboard tables.</td>
</tr>
<tr>
<td>FSMC</td>
<td>This cookie indicates the presence of FSM or SpeedoGDOS. Its value field is a pointer to a structure as follows:</td>
</tr>
<tr>
<td></td>
<td>```c</td>
</tr>
<tr>
<td></td>
<td>typedef struct</td>
</tr>
<tr>
<td></td>
<td>{</td>
</tr>
<tr>
<td></td>
<td>LONG gdos_type;</td>
</tr>
<tr>
<td></td>
<td>UWORD version;</td>
</tr>
<tr>
<td></td>
<td>WORD quality;</td>
</tr>
<tr>
<td></td>
<td>} GDOS_INFO;</td>
</tr>
<tr>
<td></td>
<td>```</td>
</tr>
<tr>
<td></td>
<td>The gdos_type field determines the variety of GDOS. _FSM represents Imagen font-based FSM whereas _SPD represents Bitstream font-based FSM. version specifies the current GDOS version.</td>
</tr>
<tr>
<td></td>
<td>quality determines the output quality of v_updwk(). The default setting is QUAL_DEFAULT (0xFFFF) which causes the driver to use the setting last set in the driver configuration accessory or CPX. This default setting may be overridden by placing a value of QUAL_DRAFT (0x0000) or QUAL_FINAL (0x0001) at this location. The quality setting should be restored to QUAL_DEFAULT at the end of each print job.</td>
</tr>
</tbody>
</table>
The BIOS provides access to six default devices (numbered 0–5). In addition, TOS 2.00 provides the ability to add extra devices with the XBIOS Bconmap() function (see the XBIOS overview for more information). Device assignments higher than device five are dependent upon the machine and any third-party enhancements. The following list indicates the device assignments which remain constant:

<table>
<thead>
<tr>
<th>Name</th>
<th>Device Number</th>
<th>GEMDOS Filename</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>DEV_PRINTER</td>
<td>0</td>
<td>PRN:</td>
<td>Centronics Parallel Port</td>
</tr>
<tr>
<td>DEV_AUX</td>
<td>1</td>
<td>AUX:</td>
<td>Default Serial Device (this device number could actually refer to any serial device connected to the system depending on which was mapped with Bconmap() )</td>
</tr>
<tr>
<td>DEV_CON</td>
<td>2</td>
<td>CON:</td>
<td>Console (screen device)</td>
</tr>
<tr>
<td>DEV_MIDI</td>
<td>3</td>
<td>N/A</td>
<td>MIDI Ports</td>
</tr>
<tr>
<td>DEV_IKBD</td>
<td>4</td>
<td>N/A</td>
<td>Intelligent Keyboard Controller</td>
</tr>
<tr>
<td>DEV_RAW</td>
<td>5</td>
<td>N/A</td>
<td>Console (no interpretation)</td>
</tr>
</tbody>
</table>

The Console Device

Two methods are provided for outputting characters to the screen. Output via BIOS device #2 subjects character codes to interpretation. Codes such as a carriage return (ASCII 13), line feed (ASCII 10), TAB (ASCII 9), CTRL-G (ASCII 7), and ESCAPE (ASCII 27) are interpreted as special cases and handled specially.

Output via BIOS device #5 causes all characters to be output literally to the screen without interpretation.

The VT-52 Emulator

The Atari console device contains emulation code compatible with the VT-52 standard. Special escapes may be used to manipulate the cursor and create text effects.

To send an escape sequence, one of the following codes (and possibly additional characters) must be sent following the ESCAPE character (ASCII 27):

<table>
<thead>
<tr>
<th>Escape</th>
<th>Code</th>
<th>Effect</th>
</tr>
</thead>
<tbody>
<tr>
<td>A</td>
<td>65</td>
<td>Move the cursor up one line. If the cursor is on the top line this does nothing.</td>
</tr>
<tr>
<td>B</td>
<td>66</td>
<td>Move the cursor down one line. If the cursor is on the bottom line this does nothing.</td>
</tr>
<tr>
<td>Key</td>
<td>Code</td>
<td>Description</td>
</tr>
<tr>
<td>-----</td>
<td>------</td>
<td>-------------</td>
</tr>
<tr>
<td>C</td>
<td>67</td>
<td>Move the cursor right one line. If the cursor is on the far right of the screen this does nothing.</td>
</tr>
<tr>
<td>D</td>
<td>68</td>
<td>Move the cursor left one line. If the cursor is on the far left of the screen this does nothing.</td>
</tr>
<tr>
<td>E</td>
<td>69</td>
<td>Clear the screen and place the cursor at the upper-left corner.</td>
</tr>
<tr>
<td>H</td>
<td>72</td>
<td>Move the cursor to the upper-left corner of the screen.</td>
</tr>
<tr>
<td>I</td>
<td>73</td>
<td>Move the cursor up one line. If the cursor is on the top line, the screen scrolls down one line.</td>
</tr>
<tr>
<td>J</td>
<td>74</td>
<td>Erase the screen downwards from the current position of the cursor.</td>
</tr>
<tr>
<td>K</td>
<td>75</td>
<td>Clear the current line to the right from the cursor position.</td>
</tr>
<tr>
<td>L</td>
<td>76</td>
<td>Insert a line by scrolling all lines at the cursor position down one line.</td>
</tr>
<tr>
<td>M</td>
<td>77</td>
<td>Delete the current line and scroll lines below the cursor position up one line.</td>
</tr>
<tr>
<td>Y</td>
<td>89</td>
<td>Position the cursor at the coordinates given by the following two codes. The screen starts with coordinates (32, 32) at the upper-left of the screen. Coordinates should be presented in reverse order, Y and then X.</td>
</tr>
<tr>
<td>b</td>
<td>98</td>
<td>This code is followed by a character from which the lowest four bits determine a new text foreground color.</td>
</tr>
<tr>
<td>c</td>
<td>99</td>
<td>This code is followed by a character from which the lowest four bits determine a new text background color.</td>
</tr>
<tr>
<td>d</td>
<td>100</td>
<td>Erase the screen from the upper-left to the current cursor position.</td>
</tr>
<tr>
<td>e</td>
<td>101</td>
<td>Enable the cursor.</td>
</tr>
<tr>
<td>f</td>
<td>102</td>
<td>Disable the cursor.</td>
</tr>
<tr>
<td>j</td>
<td>106</td>
<td>Save the current cursor position. (Only implemented as of TOS 1.02)</td>
</tr>
<tr>
<td>k</td>
<td>107</td>
<td>Restore the current cursor position. (Only implemented as of TOS 1.02)</td>
</tr>
<tr>
<td>l</td>
<td>108</td>
<td>Erase the current line and place the cursor at the far left.</td>
</tr>
<tr>
<td>o</td>
<td>111</td>
<td>Erase the current line from the far left to the current cursor position.</td>
</tr>
<tr>
<td>p</td>
<td>112</td>
<td>Enable inverse video.</td>
</tr>
<tr>
<td>q</td>
<td>113</td>
<td>Disable inverse video.</td>
</tr>
<tr>
<td>v</td>
<td>118</td>
<td>Enable line wrap.</td>
</tr>
<tr>
<td>w</td>
<td>119</td>
<td>Disable line wrap.</td>
</tr>
</tbody>
</table>

## Media Change

The BIOS function `Mediach()` returns the current media-change status of the drive specified. This state is used to determine if a disk has been changed in removable media drives (floppies, removable hard drives, etc.

The `Getbpb()` incorrectly resets the media change state. Failure to properly reset this state after calling `Getbpb()` can cause data loss. The function `mediach()`, shown below, forces the `Mediach()` function to return a ‘definitely changed’ state and should always be called after calling `Getbpb()` on removable media drives.

```c
/*
 * _mediach(): force the media 'changed' state on a removable drive.
 *
 * Usage: errcode = _mediach( devno )    - returns 1 if an error occurs
 *
 * Inputs: devno - (0 = 'A:', 1 = 'B:', etc...)
 */
```

**The Atari Compendium**
`*/
.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

loop:
  clr.l -(sp) ; Get supervisor mode, leave old SSP
  move.w #$20,-(sp) ; and "Super" function code on stack.
  trap #1
  addq.l #6,sp
  move.l d0,-(sp)
  move.w #$20,-(sp)

  move.l $472,oldgetbp
  move.l $47e,oldmediach
  move.l $476,oldrwabs
  move.l #newgetbp,$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.l #8,sp

; Fclose the handle
  tst.l d0
  bmi.s noclose

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

noclose:
  moveq #0,d7
  cmp.l #newgetbp,$472 ; still installed?
  bne.s done
  move.l oldgetbp,$472 ; Error, restore vectors.
  move.l oldmediach,$47e
  move.l oldrwabs,$476
  trap #1 ; go back to user mode
  addq.l #6,sp ; restore sp
  moveq.l #1,d0 ; 1 = Error
  rts

done:
  trap #1 ; go back to user mode
  addq.l #6,sp ; from stack left above
  clr.l d0 ; No Error

THE AtARI Compendium
rts

/*
 * New Getbpb()...if it’s the target device, uninstall vectors.
 * In any case, call normal Getbpb().
 */
newgetbpb:
move.w mydev,d0
cmp.w 4(sp),d0
bne.s dooldg
move.l oldgetbpb,$472 ; Got target device so uninstall.
move.l oldmediach,$47e
move.l oldrwabs,$476
dooldg:
move.l oldgetbpb,a0 ; Go to real Getbpb()
jmp (a0)

/*
 * New Mediach()...if it’s the target device, return 2. Else call old.
 */
newmediach:
move.w mydev,d0
cmp.w 4(sp),d0
bne.s dooldm
moveq.l #2,d0 ; Target device, return 2
rts
dooldm:
move.l oldmediach,a0 ; Call old
jmp (a0)

/*
 * New Rwabs()...if it’s the target device, return E_CHG (-14)
 */
newrwabs:
move.w mydev,d0
cmp.w 4(sp),d0
bne.s dooldr
moveq.l #-14,d0
rts
dooldr:
move.l oldrwabs,a0
jmp (a0)
.data
fspec: dc.b "X:\X",0
mydev: ds.w 1
oldgetbpb: ds.l 1
oldmediach: ds.l 1
oldrwabs: ds.l 1
.end
**BIOS Vectors**

**Reset Vector**

Shortly after a warm boot the OS will jump to the address contained in the system variable `resvector` ($42A) if the value in the system variable `resvalid` ($426) contains the magic number $31415926. The OS will supply a return address to this code segment in register A6 but the subroutine must not utilize the stack as neither stack pointer will be valid.

If your process needs to do cleanup in the event of a warm reset (see “Placing a Cookie” earlier in this chapter) the following code installs a user routine to accomplish this.

```assembly
_resvalid equ $426
_resvector equ $42A
RESMAGIC equ $31415926

.text
installres:
  move.l _resvalid,oldvalid
  move.l _resvector,oldvector
  move.l #myresvec,_resvector
  move.l #RESMAGIC,_resvalid
  rts

myresvec:
  *
  * Insert user code here
  *
  move.l oldvector,_resvector
  move.l oldvalid,_resvalid
  jmp (a6)

.bss
oldvector: ds.l 1
oldvalid: ds.l 1
```

---

T H E  A T A R I  C O M P E N D I U M
System Bell Vector
As of TOS 1.06, the OS jumps through the address contained in the system variable `bell_hook` ($5AC) to ring the system bell. It is possible for a custom routine to hook into this vector to alter the bell sound. The user routine may modify registers D0-D2/A0-A2 and may chain to the old bell handler if desired. It is also safe to make BIOS and XBIOS calls following the procedure for calling from an interrupt (when not running under MultiTOS). The routine should either jump to the old handler or execute an RTS statement.

System Keyclick Vector
Similar to the system bell vector, another vector is called each time a keyclick sound is generated. This vector is stored in system variable `kcl_hook` ($5B0) and is entered with the keycode (not the ASCII code) of the key struck in the low byte of D0. Registers D1-D2/A0-A2 may be modified, however, all other registers including D0 must be maintained. The replacement handler may either chain to a new handler or RTS.

Deferred Vertical Blank Handlers
Applications may install custom routines which are called during every vertical blank (approx. 50-72 times per second). The OS performs several operations during the vertical blank as follows:

- The system variable `_frclock` is incremented.
- The system variable `_vblsem` is tested. If 0, the vertical blank handler exits immediately.
- All registers are saved.
- The system variable `_vbclock` is incremented.
- If the system is currently in a high resolution video mode and a low-resolution monitor is detected, the video resolution is adjusted and the vector found at system variable `swv_vec` is called.
- The text cursor blink routine is called.
- If a new palette has been selected since the last vertical blank, it is loaded.
- If a new screen base address has been selected since the last vertical blank, it is selected.
- Each of the “deferred” vertical blank routine handlers is called.
- If the system variable `prt_cnt` is greater than -1, the vector at system variable `scr_dump` is called.
- Saved registers are restored and processing continues.

To install a routine to be called as a “deferred” vertical blank handler, you must inspect the list of handler vectors at `vblqueue` for a NULL slot, replace it with your vector and initialize the next slot to NULL. The system variable `nvbls` indicates the number of slots pointed to by
vblqueue. If the vertical blank handler list is filled, you may allocate a new area, copy the old list of handlers with your handler, and update the pointer vblqueue and nvbls.

The XBRA Protocol

Many applications that add functionality to the system do so by ‘hooking’ themselves into one or more interrupt or pass-through vectors (usually with Setexc()). Most vector handlers work by executing the relevant code when the interrupt is called and then calling the original vector handler. When several applications handle one vector, a vector ‘chain’ is created. This chain makes it difficult for debuggers or the process itself to ‘unhook’ itself from the chain.

The XBRA protocol was designed so that processes that wish to be able to unhook themselves may and so that debuggers can trace the ‘chain’ of vector handlers. Following the protocol is simple. Prior to the first instruction of the vector handler, insert three longwords into the application as follows:

- The longword ‘XBRA’ 0x58425241.
- Another longword containing the application ‘cookie’ ID (this is the same as that put into the cookie jar if applicable).
- A longword into which should be placed the address of the original handler.

The following code example shows how to correctly use the XBRA protocol in a routine designed to supplement the 680x0 TRAP #1 vector (GEMDOS):

```assembly
instl_trap1:
move.l #my_trap1,-(sp)
move.w #VEC_GEMDOS,-(sp)
move.w #Setexc,-(sp)
trap #13
addq.l #8,sp
move.l old_handler
rts

DC.L 'XBRA'
DC.L 'SDS1' ; Put your cookie here

old_handler DC.L 0

my_trap1:
movem.l d2-d7/a2-a6,-(sp)

; ; Your TRAP #1 handler goes here.
;
movem.l (sp)+,d2-d7/a2-a6
move.l old_handler,-(sp) ; Fake a
return
rts ; to old code.
```
The following ‘C’ function is an example of how to use the XBRA protocol to unhook a vector handler from the XBRA chain. This function will only work if all installed vector handlers follow the XBRA protocol. It takes a `Setexc()` vector number and an XBRA application id cookie as a parameter. It returns the address of the routine that was unhooked or 0L if unsuccessful.

```c
typedef struct xbra
{
    LONG xbra_id;
    LONG app_id;
    VOID (*oldvec)();
} XBRA;

LONG
unhook_xbra( WORD vecnum, LONG app_id )
{
    XBRA *rx;
    LONG vecadr, *stepadr, lret = 0L;
    char *savessp;

    vecadr = Setexc( vecnum, VEC_INQUIRE );
    rx = (XBRA *)(vecadr - sizeof( XBRA ));

    /* Set supervisor mode for search just in case. */
    savessp = Super( SUP_SET );

    /* Special Case: Vector to remove is first in chain. */
    if( rx->xbra_id == 'XBRA' && rx->app_id == app_id )
    {
        Setexc( vecnum, rx->oldvec );
        return vecadr;
    }

    stepadr = (LONG *)&rx->oldvec;
    rx = (XBRA *)((LONG)rx->oldvec - sizeof( XBRA ));
    while( rx->xbra_id == 'XBRA' )
    {
        if( rx->app_id == app_id )
        {
            *stepadr = lret = (LONG)rx->oldvec;
            break;
        }

        stepadr = (LONG *)&rx->oldvec;
        rx = (XBRA *)((LONG)rx->oldvec - sizeof( XBRA ));
    }

    Super( savessp );
    return lret;
}
```
**BIOS Function Calling Procedure**

**BIOS** system functions are called via the TRAP #13 exception. Function arguments are pushed onto the current stack (user or supervisor) in reverse order followed by the function opcode. The calling application is responsible for correctly resetting the stack pointer after the call.

The **BIOS** may utilize registers D0-D2 and A0-A2 as scratch registers and their contents should not be depended upon at the completion of a call. In addition, the function opcode placed on the stack will be modified.

The following example for `Bconout()` illustrates calling the **BIOS** from assembly language:

```assembly
move.w #char,-(sp)
move.w #dev,-(sp)
move.w #$03,-(sp)
trap #13
addq.l #6,sp
```

A ‘C’ binding for a generic **BIOS** handler would be as follows:

```c

#define Bconout( a ) bios( 0x02, a )

```

With the above code, you could easily design a ‘C’ macro to add **BIOS** calls to your compiler as in the following example for `Bconout()`:

```c

#define Bconout( a ) bios( 0x02, a )

```

The **BIOS** is re-entrant to three levels, however there is no error checking performed so interrupt handlers should avoid intense **BIOS** usage. In addition, no disk or printer usage should be attempted from the system timer interrupt, critical error, or process-terminate handlers.

**Calling the BIOS from an Interrupt**

The **BIOS** and **XBios** are the only two OS sub-systems which can be called from an interrupt handler. Precisely one interrupt handler at a time may use the **BIOS** as shown in the following code segment:

```assembly
savptr equ $4A2
savamt equ $23*2
myhandler:
sub.l #savamt,savptr
```

**THE ATARI COMPENDIUM**
; BIOS calls may be performed here
add.l #savamt, savptr
rte ; (or rts?)

This method is not valid under MultiTOS.
Bconin()

LONG Bconin(dev)
WORD dev;

Bconin() retrieves a character (if one is waiting) from the specified device.

**Opcode**
2 (0x02)

**Availability**
All TOS versions.

**Parameters**
`dev` specifies the device to read from as follows:

<table>
<thead>
<tr>
<th>Name</th>
<th><code>dev</code></th>
<th>Device</th>
</tr>
</thead>
<tbody>
<tr>
<td>DEV_PRINTER</td>
<td>0</td>
<td>Parallel port</td>
</tr>
<tr>
<td>DEV_AUX</td>
<td>1</td>
<td>Auxillary device (normally the RS-232 port, however, TOS versions with Bconmap() can map in other devices to this handle)</td>
</tr>
<tr>
<td>DEV_CONSOLE</td>
<td>2</td>
<td>Console device (keyboard)</td>
</tr>
<tr>
<td>DEV_MIDI</td>
<td>3</td>
<td>MIDI Port</td>
</tr>
<tr>
<td>DEV_IKBD</td>
<td>4</td>
<td>IKBD Controller (not available as an input device)</td>
</tr>
<tr>
<td>DEV_RAW</td>
<td>5</td>
<td>Console device (keyboard)</td>
</tr>
<tr>
<td>See Overview</td>
<td>6 –</td>
<td>Additional devices (as available)</td>
</tr>
</tbody>
</table>

**Binding**
move.w dev, -(sp)
move.w #$02,-(sp)
trap #13
addq.l #4,sp

**Return Value**
Bconin() returns a bit array arranged as follows:

<table>
<thead>
<tr>
<th>Bits 31-24</th>
<th>Bits 23-16</th>
<th>Bits 15-8</th>
<th>Bits 7-0</th>
</tr>
</thead>
<tbody>
<tr>
<td>Shift key status (see Kbshift())</td>
<td>Keyboard Scan Code</td>
<td>Reserved (0)</td>
<td>ASCII value</td>
</tr>
</tbody>
</table>

**Comments**
The shift key status is only returned if the system variable conterm (char *(0x484) ) has bit 3 set. This is normally disabled.

Non-ASCII keys return 0 in bits 7-0.

**See Also**
Bconstat(), Cconin(), Cauxin()
Bconout()

LONG Bconout( dev, ch )
WORD dev, ch;

Bconout() outputs a character to a named device.

OPCODE 3 (0x03)

AVAILABILITY All TOS versions.

PARAMETERS dev specifies the output device as follows:

<table>
<thead>
<tr>
<th>Name</th>
<th>dev</th>
<th>Device</th>
</tr>
</thead>
<tbody>
<tr>
<td>DEV_PRINTER</td>
<td>0</td>
<td>Parallel port</td>
</tr>
<tr>
<td>DEV_AUX</td>
<td>1</td>
<td>Auxillary device (see note under Bconin() )</td>
</tr>
<tr>
<td>DEV_CONSOLE</td>
<td>2</td>
<td>Console device (screen)</td>
</tr>
<tr>
<td>DEV_MIDI</td>
<td>3</td>
<td>MIDI port</td>
</tr>
<tr>
<td>DEV_IKBD</td>
<td>4</td>
<td>Keyboard (IKBD)</td>
</tr>
<tr>
<td>DEV_RAW</td>
<td>5</td>
<td>Raw screen device (control characters and escapes are not processed)</td>
</tr>
<tr>
<td>See Overview</td>
<td>6</td>
<td>Additional devices (as available)</td>
</tr>
</tbody>
</table>

BINDING
move.w ch,-(sp)
movw w dev,-(sp)
mov.w #$03,-(sp)
trap #13
addq.l #6,sp

RETURN VALUE Bconout() returns 0 if the character was sent successfully or non-zero otherwise.

SEE ALSO Bconin(), Cconout(), Cauxout(), Cprnout(), Bconstat()

Bconstat()

LONG Bconstat( dev )
WORD dev;

Bconstat() determines whether the specified device is prepared to transmit at least one character.

OPCODE 1 (0x01)
**Availability**

All TOS versions.

**Parameters**

*dev* specifies the device to check as listed under **Bconin()**.

**Binding**

```
move.w dev,-(sp)
mov.w #$01,-(sp)
trap #13
addq.l #4,sp
```

**Return Value**

**Bcostat()** returns 0 if no characters are waiting or -1 if characters are waiting to be received.

**See Also**

**Bconin(), Cconis(), Cauxis()**

---

**Bcostat()**

**LONG Bcostat( dev )**

**WORD dev;**

**Bcostat()** determines if the specified device is prepared to receive a character.

**Opcode**

8 (0x08)

**Availability**

All TOS versions.

**Parameters**

*dev* specifies the device to poll as listed under **Bconout()**.

**Binding**

```
move.w dev,-(sp)
mov.w #$08,-(sp)
trap #13
addq.l #4,sp
```

**Return Value**

**Bcostat()** returns 0 if the device is not ready to receive characters or -1 otherwise.

**Caveats**

A bug in TOS 1.0 existed that caused the IKBD and MIDI device numbers to become swapped when being handled by the **Bcostat()** call, subsequently returning data for the wrong device. To allow previously written programs to continue operating correctly, this bug has been maintained on purpose in all current versions of TOS. You should therefore specify a value of 3 for the IKBD and 4 for MIDI for this call only.

**See Also**

**Bconout(), Cauxos(), Cconos(), Cprnos()**
**Drvmap()**

ULONG Drvmap( VOID )

`Drvmap()` returns a list of mounted drives.

**OPCODE**
10 (0x0A)

**AVAILABILITY**
All TOS versions.

**PARAMETERS**
None.

**BINDING**
move.w #$0A,-(sp)
trap #13
addq.l #2,sp

**RETURN VALUE**
`Drvmap()` returns a ULONG bitmap of mounted drives. For each drive present, its bit is enabled. Drive ‘A:’ is bit 0, drive ‘B:’ is bit 1, and so on.

**COMMENTS**
Single floppy systems will indicate that two drives are available since both drives can actually be addressed. A request for drive ‘B:’ will simply cause TOS to ask the user to insert ‘Disk B’ and provide automatic handling routines for all disk swapping.

**SEE ALSO**
Dsetdrv()

---

**Getbpb()**

BPB *Getbpb( dev )

WORD dev;

`Getbpb()` returns the address of the current BPB (Bios Parameter Block) for a mounted device.

**OPCODE**
7 (0x07)

**AVAILABILITY**
All TOS versions.

**PARAMETERS**
`dev` specifies the mounted device (‘A:’ = 0, ‘B:’ = 1).

**BINDING**
move.w dev,-(sp)
move.w #$07,-(sp)
trap #13
addq.l #4,sp
**RETURN VALUE**

`Getmpb()` returns a pointer to the device’s BPB. The BPB is defined as follows:

```c
typedef struct
{
    WORD recsiz; /* bytes per sector */
    WORD clsiz; /* sectors per cluster */
    WORD clsizb; /* bytes per cluster */
    WORD rdlen; /* sector length of root directory */
    WORD fsiz; /* sectors per FAT */
    WORD fatrec; /* starting sector of second FAT */
    WORD datrec; /* starting sector of data */
    WORD numcl; /* clusters per disk */
    WORD bflags; /* bit 0=1 - 16 bit FAT, else 12 bit */
} BPB;
```

**CAVEATS**

A media change *must* be forced after calling this function prior to making any GEMDOS calls. Failure to do so may cause GEMDOS to become unaware of a disk change causing data loss. Refer to the discussion of forcing a media change earlier in this chapter.

**Getmpb()**

VOID Getmpb (mpb)

`Getmpb()` returns information regarding GEMDOS free and allocated memory blocks.

**OPCODE**

0 (0x00)

**AVAILABILITY**

All TOS versions.

**PARAMETERS**

`mpb` is a pointer to a MPB structure which is filled in by the function. The related structures are defined as follows:

```c
typedef struct md
{
    struct md *m_link; /* pointer to next block */
    VOIDP m_start; /* pointer to start of block */
    LONG m_length; /* length of block */
    BASEPAGE *m_own; /* pointer to basepage of owner */
} MD;

typedef struct mpb
{
    MD *mp_mfl; /* free list */
    MD *mp_mal; /* allocated list */
    MD *mp_rover; /* roving pointer */
} MPB;
```
**3.32 – BIOS Function Reference**

**BINDING**

```
pea mpb
clr.w -(sp)
trap #13
addq.l #6,sp
```

**Caveats**

MultiTOS uses a very different method of memory management which makes this call useless.

**Comments**

An application should never attempt to modify any of the returned information nor make any assumptions about memory allocation because of this function.

**See Also**

Malloc(), Mfree()

---

**Kbshift()**

**LONG Kbshift( mode )**

**WORD mode;**

Kbshift() allows the user to interrogate or modify the state of the keyboard ‘special’ keys.

**Opcode**

11 (0x0B)

**Availability**

All TOS versions.

**Parameters**

`mode` is -1 to read the state of the keys or a mask of the following values to change the current state:

<table>
<thead>
<tr>
<th>Name</th>
<th>Mask</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>K_RSHIFT</td>
<td>0x01</td>
<td>Right shift key depressed</td>
</tr>
<tr>
<td>K_LSHIFT</td>
<td>0x02</td>
<td>Left shift key depressed</td>
</tr>
<tr>
<td>K_CTRL</td>
<td>0x04</td>
<td>Control key depressed</td>
</tr>
<tr>
<td>K_ALT</td>
<td>0x08</td>
<td>Alternate key depressed</td>
</tr>
<tr>
<td>K_CAPSLOCK</td>
<td>0x10</td>
<td>Caps-lock engaged</td>
</tr>
<tr>
<td>K_CLRHOME</td>
<td>0x20</td>
<td>Clr/Home key depressed</td>
</tr>
<tr>
<td>K_INSERT</td>
<td>0x40</td>
<td>Insert key depressed</td>
</tr>
</tbody>
</table>

**Binding**

```
move.w mode,-(sp)
mv.e #0B,-(sp)
trap #13
addq.l #4,sp
```

**Return Value**

Kbshift() returns the state that the keyboard ‘special’ keys were in prior to the call.
**Comments**

*Kbshift() is not a particularly fast call. If you are only interested in reading the state a documented macro follows that replaces *Kbshift() and is much faster. Call the kb_init() function, as shown below, before using:

```c
char *p_kbshift;
#define Kbstate()  *p_kbshift

VOID kb_init(VOID)
{
    /* GetROMSysbase is defined in the BIOS Overview */
    OSHEADER *osheader = GetROMSysbase();

    if ( osheader->os_version == 0x0100 )
        p_kbshift = (char *)0xe1bL;
    else
        p_kbshift = *(char **)osheader->p_kbshift;
}
```

**See Also**

evnt_keybd(), evnt_multi(), Cconin(), Bconin()

---

**Mediach()**

**LONG Mediach( dev )**

**WORD dev;**

`Mediach()` inquires as to whether the ‘media’ has been changed since the last disk operation on a removable block device (floppy, removable hard drive, floptical, etc...).

**Opcode**

9 (0x09)

**Availability**

All TOS versions.

**Parameters**

`dev` specifies the mounted device number to inquire (‘A’ = 0, ‘B’ = 1, etc.).

**Binding**

```assembly
move.w  dev,-(sp)  
move.w  #$09,-(sp)  
trap  #13  
addq.l  #4,sp  
```

**Return Value**

`Mediach()` returns one of three values:

<table>
<thead>
<tr>
<th>Name</th>
<th>Value</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>MED_NOCHANGE</td>
<td>0</td>
<td>Media has not changed</td>
</tr>
<tr>
<td>MED_UNKNOWN</td>
<td>1</td>
<td>Media may have changed</td>
</tr>
<tr>
<td>MED_CHANGED</td>
<td>2</td>
<td>Media has changed</td>
</tr>
</tbody>
</table>

---

*The Atari Compendium*
See Also

Getbpb()

**Rwabs()**

LONG Rwabs( mode, buf, count, recno, dev, lrecno )

WORD mode;
VOIDP buf;
WORD count,recno,dev;
LONG lrecno;

Rwabs() reads and writes sectors to a mounted device.

**Opcode**

4 (0x04)

**Availability**

All TOS versions. Hard disk access requires the use of a hard disk driver (such as AHDI). The long sector offset version is only available as of AHDI 3.0. AHDI version numbers can be inquired through system variable pun_ptr (see discussion earlier in this chapter).

**Parameters**

*mode* is a bit mask which effects the operation to be performed as follows:

<table>
<thead>
<tr>
<th>Name</th>
<th>Bit</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>RW_READ</td>
<td>0</td>
<td>0 = Read, 1 = Write</td>
</tr>
<tr>
<td>or</td>
<td></td>
<td></td>
</tr>
<tr>
<td>RW_WRITE</td>
<td></td>
<td></td>
</tr>
<tr>
<td>RW_NOMEDIACH</td>
<td>1</td>
<td>Do not read or modify the media change status.</td>
</tr>
<tr>
<td>RW_NORETRIES</td>
<td>2</td>
<td>Disable retries</td>
</tr>
<tr>
<td>RW_NOTRANSLATE</td>
<td>3</td>
<td>Do not translate logical sectors into physical sectors (recno specifies physical instead of logical sectors)</td>
</tr>
</tbody>
</table>

The read or write operation is performed at address *buf*. *buf* must be *count* * bytes per logical sector in logical mode or *count* * 512 bytes in physical mode. *count* specifies how many sectors will be transferred.

*dev* specifies the index of the mounted device. In logical mode, ‘C:’ is 2, ‘D:’ is 3, etc... In physical mode, devices 2-9 are the ACSI devices and 10-17 are SCSI devices.

*recno* specifies the first sector to read from. If you need to specify a long offset, set *recno* to -1 and pass the long value in *lrecno*. When using a version of the AHDI below 3.0, the parameter *lrecno* should not be passed.

**Binding**

/* If running AHDI <3.0 omit first parameter */
move.l  lrecno,-(sp)
move.w  dev,-(sp)
move.w  recno,-(sp)
move.w  count,-(sp)
pea     buf,-(sp)
move.w  mode,-(sp)
move.w  #$04,-(sp)
trap    #13
lea     18(sp),sp

RETURN VALUE

Rwabs() returns E_OK (0) if successful or a negative BIOS error code otherwise.

COMMENTS

Some C compilers (Lattice C in particular) have a secondary binding called Lrwabs() used to pass the additional parameter.

This function may invoke the critical error handler (etv_critic).

Setexc()

(VOIDP)() Setexc( num, newvec )
WORD    num;
VOID (*newvec());

Setexc() reads or modifies system exception vectors.

OPCODE

5 (0x05)

AVAILABILITY

All TOS versions.

PARAMETERS

num indicates the vector number you are interested in. To obtain the vector number divide the address of the vector by 4. Some common vectors are:

<table>
<thead>
<tr>
<th>Name</th>
<th>num</th>
<th>Vector</th>
</tr>
</thead>
<tbody>
<tr>
<td>VEC_BUSERROR</td>
<td>0x02 - 0x04</td>
<td>Bomb errors (Bus, Address, Instruction)</td>
</tr>
<tr>
<td>VEC_ADDRESSERROR</td>
<td>0x21</td>
<td>Trap #1 (GEMDOS)</td>
</tr>
<tr>
<td>VEC_ILLEGALINSTRUCTION</td>
<td>0x22</td>
<td>Trap #2 (AES/VDI)</td>
</tr>
<tr>
<td>VEC_BIOS</td>
<td>0x2D</td>
<td>Trap #13 (BIOS)</td>
</tr>
<tr>
<td>VEC_XBIOS</td>
<td>0x2E</td>
<td>Trap #14 (XBIOS)</td>
</tr>
<tr>
<td>VEC_TIMER</td>
<td>0x100</td>
<td>System timer (etv_timer)</td>
</tr>
<tr>
<td>VEC_CRITICALERROR</td>
<td>0x101</td>
<td>Critical error handler (etv_critic)</td>
</tr>
<tr>
<td>VEC_TERMINATE</td>
<td>0x102</td>
<td>Process terminate handle (etv_term)</td>
</tr>
</tbody>
</table>

newvec should be the address of your new vector handler. Passing a value of
VEC_INQUIRE((VOIDP)-1) will not modify the vector.

**Binding**

```assembly
  pea       newvec
  move.w   num, -(sp)
  move.w   #$05, -(sp)
  trap     #13
  addq.l   #8, sp
```

**Return Value**
The original value of the vector is returned by the call.

**Comments**
You must reinstate old vector handlers you changed prior to your process exiting.

Programs which modify replace system vector code should install themselves following the conventions of the XBRA protocol. For details, consult the overview portion of this chapter.

---

**Tickcal()**

**LONG** Tickcal( VOID )

Tickcal() returns the system timer calibration.

**OpCode**

6 (0x06)

**Availability**

All TOS versions.

**Parameters**

None.

**Binding**

```assembly
  move.w   #$06, -(sp)
  trap     #13
  addq.l   #2, sp
```

**Return Value**

Tickcal() returns a LONG indicating the number of milliseconds between system clock ticks.
Overview

The eXtended Basic Input/Output System (XBIOS) is a software sub-system of TOS which contains functions used to interact with and control Atari computer hardware. The availability of many of these functions is dependent on hardware whose presence can be determined by the current TOS version or by interrogating the system ‘cookie jar’ (see Chapter 3: BIOS for more details).

Some functions (notably video hardware and storage device related functions) should only be used by device drivers and system level software as they represent a non-portable method of hardware interaction which may be unsupported in future Atari computers.

As a general rule, GEMDOS and VDI functions should be used, when possible, rather than XBIOS calls. The GEMDOS and VDI provide a software abstraction layer which will make software applications much more compatible across new computer releases.

Video Control

The video capabilities of Atari computer systems have varied greatly since their introduction. Applications which use the VDI for their video displays will require little if any modifications to run on new systems. The XBIOS is mostly required for device drivers and other applications which require more direct control over the video hardware. When present, the ‘_VDO’ entry in the system cookie jar will reveal information about the video hardware present.

The Physical/Logical Screen

Two separate video display pointers are maintained by the XBIOS at any time. The physical screen address points to the memory location that the video shifter uses to update the display. This memory must not be in fast RAM and must be WORD-aligned (original ST computers expect screen memory to be aligned to a 256-byte boundary).

A second video memory pointer points to the ‘logical’ screen. This memory area is used by the VDI to output graphics. Normally, the physical screen address is equal to the logical screen address meaning that VDI output is shown immediately on screen. Software (most commonly games) can allocate an additional memory block and use these two pointers to page-flip for smooth animations.

Physbase() and Logbase() return these two addresses. Setscreen() can be used to reset these addresses and change screen modes. As of TOS 4.0, Setscreen() reinitializes the VDI screen driver (you must still call vq_extnd() to update your workstations) but will not reinitialize the AES. This means that if you change resolution using Setscreen(), do not use the AES until the screen is restored to its original resolution. On TOS versions prior to 4.0, you should not use any GEM calls while the screen mode is altered.
The Falcon030 function `VGetSize()` is a utility function that will return the number of bytes that must be allocated for the specified video mode. When not running on a Falcon030, you will have to calculate this yourself.

Setting/Determining Screen Resolution

`Getrez()` was originally a safe method for determining the current video hardware configuration. As new video modes became available, though, `Getrez()` became less and less useful. Currently, `Getrez()` should be used for only one purpose. The formula `Getrez() + 2` should be used to select the VDI physical device ID for the screen so that the proper screen fonts can be selected. See the description of `v_opnwvk()` for more details.

In order to provide true screen independence, you should use the values returned by the VDI call `v_opnwvk()` to determining the screen resolution your application is using. The XBIOS provides calls that will determine the current video mode but they are hardware dependent and will probably stop working as expected as new video hardware is released.

The `Getrez()` call can reliably determine the video mode of an ST, STe or Mega ST/e. Three calls have since been added to determine the video mode of the TT030 and Falcon030 computers.

`EgetShift()` and `EsetShift()` can be used to interrogate and set the TT030 video mode. `VsetMode()` can similarly be used to interrogate and set the Falcon030 video mode. The Falcon030 call `VgetMonitor()` can be used to determine the type of attached monitor and, therefore, the available video modes.

TT030 TOS also provides the calls `EsetGray()` and `EsetSmear()`. Together, these calls duplicate some of the functionally contained in `EsetShift()` but can be used individually as desired to configure the special gray-scale and smear modes present in the TT030.

`EsetShift()` and `VsetMode()` are designed to change the video modes of the TT030 and Falcon030 respectively, however, they do not reinitialize the AES or VDI. It is also possible to change TT030 and Falcon030 video modes using `Setscreen()`. TT030 modes are set by supplying the appropriate resolution code (see `Getrez()` for a list of resolution codes). Falcon030 modes are set by adding an extra parameter to the call with a special resolution code of 3. See the explanation for `Setscreen()` later in this chapter for details.

Manipulating the Palette

Prior to the introduction of the TT, `Setcolor()` and `Setpalette()` were used to set the 16 available palette entries. `Setpalette()` sets the entire palette at once whereas `Setcolor()` sets colors at an individual level and can also be used to interrogate palette entries.

The ST has 16 palette entries, each supporting any of 512 available colors. The ST specifies color in components of red, green, and blue. Intensity settings of 0–7 are valid for each color component. The following list contains the red, green, and blue values for the ST’s default 16 color palette.
You might have noticed that these registers are not mapped the same as VDI color indexes. The VDI re-maps color requests to its own needs. For a list of these re-mappings, see the entry for `vr_trnfm()`. It is also possible to build a remapping table on the fly by plotting one pixel for each VDI pen on the screen and using the VDI `v_get_pixel()` call on each to return the VDI and hardware register index.

Each of the sixteen color registers is bitmapped into a WORD as follows (The first row indicates color, the second is bit significance):

```
xxxx  xRRR  xGGG  xBBB
xxxx  x321  x321  x321
```

The STe series expanded the color depth to four bits instead of three which expanded the number of available colors from 512 to 4096. This changed the layout of these color WORDs as follows:

```
xxxx  RRRR  GGGG  BBBB
xxxx  1432  1432  1432
```

This odd bit layout allowed for backward compatibility to the ST series.

The TT030 supports an expanded palette of 256 entries in 16 banks containing any of 4096 colors. The first bank of colors is still supported by `Setcolor()` and `Setpalette()`, however to access the additional 240 colors, 4 additional palette support calls were added.

`Esetpalette()`, `Egetpalette()`, and `Esetcolor()` provide access to these colors in a similar manner to `Setpalette()` and `Setcolor()`. `Esetbank()` switches between the 16 available banks of colors in color modes that support less than 16 colors. You should note that the TT030 color calls returned the color WORDs to normal bit ordering as follows:
When using the TT’s special gray mode, the lower eight bits of each hardware register is used as a gray value from 0–255.

The Falcon030 computer gives up the TT030 calls in favor of a more portable method of setting the hardware palette (ST calls will remain as compatible as possible). \texttt{VsetRGB()} and \texttt{VgetRGB()} set color palette entries based on 24-bit true color values. The \texttt{XBIOS} will scale these values as appropriate for the screen mode.

**Advanced Video**

\texttt{Vsync()} halts all further processing by the application until a vertical blank interrupt occurs. This interrupt signals that the video display gun has reached the bottom of the display and is returning to the top. At this time, a brief period occurs where updates to the screen will not be immediately apparent to the user. This time is usually used to present flicker-free animation and redraws.

\texttt{VsetSync()} is used to enable external hardware video synchronization for devices such as GENLOCK’s. Both the vertical and horizontal synchronizations may be set independent of each other with this call.

\texttt{VsetMask()} provides easy access to the Falcon030’s overlay mode. This call allows you to specify bits which will be added or removed to future color definitions created with the \texttt{VDI \_vs\_color()} call. When a GENLOCK hardware device is connected, pixels with their overlay bit cleared will be replaceable by the device with external video.

**The Falcon030 Sound System**

\texttt{XBIOS} sound system calls are only present as of the Falcon030 computer (though their presence should always be verified by the ‘\_SND’ cookie). If you want to program digitized audio that plays on an STe, TT, and Falcon030, see Chapter 5: \textit{Hardware}.

The Falcon030 sound system consists of four stereo 16-bit DMA playback and record channels\(^1\), an onboard ADC (microphone jack), DAC (speaker and headphone jack), connection matrix, and digital signal processor.

When your application uses the sound system you should first lock it with \texttt{Locksnd()}. This ensures that other system processes don’t try to access the sound system simultaneously. \texttt{UnLocksnd()} should be used as soon as the sound system is free.

\(^1\)Only one output track may be monitored at a time, though the DSP may be programmed as a mixer to combine more tracks while sound is being output.
Each of four possible source devices can be connected to any or all of the four possible destination devices using the connection matrix as follows:

```
<table>
<thead>
<tr>
<th></th>
<th>DAC</th>
<th>DSP Receive</th>
<th>DMA Record</th>
<th>Ext. Output</th>
</tr>
</thead>
<tbody>
<tr>
<td>External Input</td>
<td>⊙</td>
<td>⊙</td>
<td>⊙</td>
<td>⊙</td>
</tr>
<tr>
<td>DSP Transmit</td>
<td>⊙</td>
<td>⊙</td>
<td>⊙</td>
<td>⊙</td>
</tr>
<tr>
<td>DMA Playback</td>
<td>⊙</td>
<td>⊙</td>
<td>⊙</td>
<td>⊙</td>
</tr>
<tr>
<td>ADC (PSG/Mic)</td>
<td>⊙</td>
<td>⊙</td>
<td>⊙</td>
<td>⊙</td>
</tr>
</tbody>
</table>
```

The external input and output are accessible with a specially designed hardware device connected to the DSP connector.

**The Connection Matrix**

The sound system call `Devconnect()` connects sound system components together. You must specify the source device, destination device(s), source clock, prescaler setting, and handshaking protocol.

The source clock can be set to either of two internal clocks (25.175 MHz and 32 MHz) or an external clock. The internal DMA sound routines are only compatible with the 25.175 MHz clock. Other clock sources are used in conjunction with external hardware devices.

The prescaler sets the actual sample playback and recording rate. A value of 0 will cause the sound system to use a STe/TT030 compatible prescaler for outputting sound recorded at STe/TT030 frequencies. One STe/TT030 frequency, 6.258 kHz, is not supported on the Falcon030. You can set the STe/TT030 prescaler with the `Soundcmd()` call. Using values other than 0 will set the Falcon030 prescaler as documented under the `Devconnect()` call.

The last parameter you must pass to `Devconnect()` specifies whether to enable or disable hardware handshaking. Enabling handshaking will produce data that is 100% error free but will result in a variable transfer rate which may negatively affect digital sound. Handshaking is generally only enabled when the data being transferred must be transferred without errors (usually compressed audio or video data).

**Recording/Playing Digital Audio**

To record or playback an audio sample, use `Setbuffer()` to identify the location and length of your playback/recording buffer. Also, any `Devconnect()`, `Setmode()`, and `Soundcmd()` calls should be made prior to starting your playback/recording to set the sound hardware to the proper frequency and mode.
The Falcon030 only supports the recording of 16-bit stereo audio. To generate 8-bit samples you must scale the values in the buffer from WORDs to BYTEs after recording.

When processing either recording or playback through the DSP, the command Dsptristate() must be used to connect the DSP to the matrix.

You may use the function Setinterrupt(), as desired, to cause a MFP or Timer A interrupt at the end of every frame. This is most useful when you are playing or recording in repeat mode and you wish to use multiple buffers.

Buffptr() may be used to determine the current playback or record buffer pointer as sounds are being played/recorded.

Setmontracks() is used to define which track which will be output over the computer speaker/headphones. Settracks() controls which tracks will be used to record/playback data.

Configuring Levels

The function Soundcmd() has four modes which allow the setting and interrogation of the current levels of attenuation and gain. Gain affects input levels. The higher the value for gain, the louder the microphone input will be. Attenuation affects output levels. The higher the attenuation setting, the softer sounds will be output from the computer speaker/headphone jack.

Other Calls

Sndstatus() can be used to tell if a source clock rate was correctly set or if hardware clipping has occurred on either channel.

Gpio() is used to communicate data over the three general purpose pins of the DSP connector.

The DSP

The Falcon030 comes standard with a Motorola 56001 digital signal processor (DSP). Digital signal processors are useful for many different purposes such as audio/video compression, filtering, encryption, modulation, and math functions.

The DSP is able to support both programs and subroutines. Both must be written in 56001 assembly language (or a language which outputs 56001 object code). A full treatment of 56001 assembly language is beyond the scope of this document. Consult the DSP56000/56001 Digital Signal Processor’s User Manual published by Motorola, Inc. for more information.

The DSP is capable of having many subroutines resident in memory, however, only one program may be loaded at any time.

When using the DSP you should call Dsp_Lock() to prevent other processes from modifying your setup and to ensure that you do not modify the work of other processes. Call Dsp_Unlock()
when done (the DSP’s MR and IPR registers should have been returned to their original state) to release the DSP semaphore.

DSP Memory

The Falcon030’s DSP contains 96K bytes of RAM for system programs, user programs, and subroutines. The DSP uses three distinct address spaces, X, Y, and P. Program memory (P) overlaps both X and Y memory spaces. Because of this, DSP programs should be careful when referencing memory. The following is a memory map of the DSP:

![Memory Map Diagram]

DSP Word Size

The 56001 uses a 24-bit WORD. Future Atari computers may use different DSP’s with different WORD sizes. Use the Dsp_GetWordSize() call prior to using the DSP to determine the proper DSP WORD size.

DSP Subroutines

Subroutines are usually short programs (no longer than 1024 DSP WORDs) which transform incoming data. Each subroutine must be written to be fully relocatable. When writing subroutines, start instructions at location $0$. All addresses in the subroutine must be relocatable based on the original PC of $0$ in order to function. An alternative to this is to include a stub program at the start of your subroutine that performs a relocation based upon the start address assigned by the XBIOS (which is available in X:HRX at subroutine start).

Subroutines should store initialized data within its program space. The memory area from $3f00–$3fff is reserved for use as the BSS of subroutines. Subroutines must not rely on the BSS’s data to remain constant between subroutine calls.
Each subroutine must be assigned a unique ability code either by using one predefined by Atari (none have been published yet) or by using the `Dsp_RequestUniqueAbility()` call. Since subroutines are only flushed from the DSP when necessary, an application may be able to use an existing subroutine with the same ability left by another application by using the `Dsp_InqrSubrAbility()` call.

Here is a sample of how to load a DSP subroutine with a non-unique ability code:

```c
if(!DSP_Lock())
{
    ability = DSP_RequestUniqueAbility();
    handle = DSP_LoadSubroutine( subptr, length, ability );
    if(!handle)
    {
        DSP_FlushSubroutines();
        handle = DSP_LoadSubroutine( subptr, length, ability );
        if(!handle)
            error("Unable to load DSP subroutine");
    }
    if(handle)
    {
        if(!Dsp_RunSubroutine( handle ))
            DSP_DoBlock( data_in, size_in, data_out, size_out);
        else
            error("Unable to run DSP subroutine!");
    }
}
```

### DSP Programs

Only one DSP program may be resident in memory at once. Prior to loading a DSP program you should ensure enough memory is available for your program by calling `Dsp_Available()`. If not enough memory is available, you may have to flush resident subroutines to free enough memory.

After you have found that enough memory is available, you must reserve it with `Dsp_Reserve()`. This memory will be reserved until the next `Dsp_Reserve()` call so you should ensure that you have called `Dsp_Lock()` to block other processes from writing over your program.

Programs can be stored in either binary or ASCII (‘.LOD’) format. The function `Dsp_LodToBinary()` can be used to convert this data. DSP programs in binary form load much faster than those in the ‘.LOD’ format.

`Dsp_LoadProg()` is used to execute programs stored on disk in the ‘.LOD’ format. `Dsp_ExecProg()` is used to execute programs stored in memory in binary format.

As with subroutines, programs are assigned a unique ability code that can be determined with `Dsp_GetProgAbility()`.

### Sending Data to the DSP

Several functions transfer data to and from DSP programs and subroutines as follows:
You should read the description of each in the function reference and decide which is best suited for your needs.

**Dsp_SetVectors()** installs special purpose routines that are called when the DSP sends an interrupt indicating it is ready to send or receive data. **Dsp_RemoveInterrupts()** removes these routines from the vector table in memory.

### DSP State

The HFx bits of the HSR register can be read atomically with the four calls **Dsp_Hf0()**, **Dsp_Hf1()**, **Dsp_Hf2()**, and **Dsp_Hf3()**. The current value of the ISR register may be read with **Dsp_Hstat()**.

DSP programs may also define special host commands at DSP vectors $13$ and $14$ to be triggered by the command **DSP_TriggerHC()**.

### DSP Debugging

When full control over the DSP is necessary (such is the case for specialized debuggers), the command **Dsp_ExecBoot()** can be used to download up to 512 DSP WORDs of bootstrap code. The DSP will be reset before this happens. This call should only be used by advanced applications as it will cause other DSP functions to stop working unless those functions are properly supported.

### User/Supervisor Mode

The XBIOS call **Supexec()** provides access to a special mode of the 680x0 processor called supervisor mode. Normal programs always execute in user mode. Programs operating in user mode, however, have less memory access privileges than those operating in supervisor mode.

Some special instructions of the 680x0 may only be executed in supervisor mode. In addition, any memory reads or writes to locations $0$–$7FF$ or memory-mapped I/O must be made in supervisor mode.
To use `supexec()`, simply pass it the address of a function to be called. When writing the function in ‘C’, you should be careful to define the function in a way that is safe for your compiler (see your compiler documentation for details).

While in supervisor mode, the AES should never be called.

### MetaDOS

One special XBIOS opcode, `Metainit()` was reserved for a TOS extension called MetaDOS. MetaDOS was designed to supplement the OS to allow for more than 16 drives and to provide the extra support needed for CD-ROM drives. MetaDOS is no longer officially supported by Atari because of the increased functionality of MultiTOS.

MultiTOS allows the use of all 26 drive letters as well as providing loadable device drivers and file systems. See Chapter 2: GEMDOS for more information.

### Keyboard and Mouse Control

The XBIOS has several functions that provide extended control over the keyboard and mouse. These functions should be used with care, however, as the keyboard and mouse are ‘global’ devices shared by other processes.

`Initmous()` is used to change the way the keyboard controller reports mouse movements to the system. Changing this mode will cause the AES and VDI to be unable to recognize mouse input.

`Keytbl()` allows you to read and manipulate the tables which translate IKBD scan codes into ASCII codes. This is essential when you want your application to run on Atari machines with foreign keyboards. Use `Keytbl()` to return a pointer to the internal table structure and then convert keycodes into ASCII by looking codes up in the appropriate table.

### Loadable XBIOS Keyboard Tables

TOS versions 5.0 and greater support the loading of external keyboard tables when the ‘_AKP’ cookie is present. In this case, if a file called ‘KEYTBL.TBL’ is found in the ‘\MULTITOS’ directory of the boot drive, it will be loaded upon bootup to provide keyboard mapping changes. The format of the file is as follows:

<table>
<thead>
<tr>
<th>Table Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Magic Table Identifier Word</td>
<td>This should be a WORD value of 0x2771.</td>
</tr>
<tr>
<td>Unshifted Keyboard Table</td>
<td>This is a 128 byte table of ASCII codes that are generated when no keyboard shift keys are being held down. There is one entry for each possible scan code.</td>
</tr>
<tr>
<td>Shifted Keyboard Table</td>
<td>This is a 128 byte table of ASCII codes that are generated when the SHIFT key is being held down. There is one entry for each possible scan code.</td>
</tr>
</tbody>
</table>
The configuration functions `Cursconf()` and `Kbrate()` set the cursor blink rate and keyboard repeat rates respectively. These settings should only be changed by a CPX or other configuration utility at the user’s request as they are global and affect all applications.

**IKBD Intelligent Keyboard Controller**

The IKBD Controller is an intelligent hardware device that handles communications between the computer and the keyboard matrix. The XBIOS function `Ikbdws()` can be used to transmit command strings to the IKBD controller. For further information about the IKBD, consult *Chapter 5: Hardware*.

## Disk Functions

### Boot Sectors

Both floppy disks and hard disks share a similar format for boot sectors as follows:

<table>
<thead>
<tr>
<th>Name</th>
<th>Offset</th>
<th>Contents</th>
</tr>
</thead>
<tbody>
<tr>
<td>BRA</td>
<td>0x0000</td>
<td>This WORD contains a 680x0 BRA.S instruction to the boot code in this sector if the disk is executable, otherwise it is unused.</td>
</tr>
<tr>
<td>OEM</td>
<td>0x0002</td>
<td>These six bytes are reserved for use as any necessary filler information. The disk-based TOS loader program places the string ‘Loader’ here.</td>
</tr>
<tr>
<td>SERIAL</td>
<td>0x0008</td>
<td>The low 24-bits of this LONG represent a unique disk serial number.</td>
</tr>
</tbody>
</table>
The boot sector may be found on side 0, track 0, sector 1 of each physical disk.

The Floppy Drive
The XBIOS provides several functions used for reading, writing, verifying, and formatting sectors on the hard disk.

Floprd() and Flopwr() read and write from the floppy drive at the sector level rather than the file level. For example, these functions could be used to create executable boot sectors on a floppy disk. Flopver() can be used to verify written sectors against data still in memory.

Formatting a floppy disk is accomplished with Flopfmt(). After a floppy is completely formatted use the function Protobt() to create a prototype boot sector (as shown above) which can then be written to sector #1 to make the disk usable by TOS.

ASCI and SCSI DMA
The functions DMAread() and DMAwrite() were added as of TOS 2.00. These functions provide a method of accessing ACSI and SCSI devices at the sector level.

<table>
<thead>
<tr>
<th></th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>BPS</strong></td>
<td>0x000B</td>
</tr>
<tr>
<td><strong>SPC</strong></td>
<td>0x000D</td>
</tr>
<tr>
<td><strong>RES</strong></td>
<td>0x000E</td>
</tr>
<tr>
<td><strong>NFATS</strong></td>
<td>0x0010</td>
</tr>
<tr>
<td><strong>NDIRS</strong></td>
<td>0x0011</td>
</tr>
<tr>
<td><strong>NSECTS</strong></td>
<td>0x0013</td>
</tr>
<tr>
<td><strong>MEDIA</strong></td>
<td>0x0015</td>
</tr>
<tr>
<td><strong>SPF</strong></td>
<td>0x0016</td>
</tr>
<tr>
<td><strong>SPT</strong></td>
<td>0x0018</td>
</tr>
<tr>
<td><strong>NSIDES</strong></td>
<td>0x001A</td>
</tr>
<tr>
<td><strong>NHID</strong></td>
<td>0x001C</td>
</tr>
<tr>
<td><strong>BOOTCODE</strong></td>
<td>0x001E</td>
</tr>
<tr>
<td><strong>CHECKSUM</strong></td>
<td>0x01FE</td>
</tr>
</tbody>
</table>
ASCI accesses must not use alternate RAM as a transfer buffer because they are performing DMA. The TT030 uses handshaking for SCSI so alternate RAM transfers are safe. SCSI transfers on the Falcon030 do, however, use DMA so alternate RAM must be avoided.

If you need to transfer data using these functions to an alternate RAM buffer, use the special standard memory block pointed to by the cookie ‘_FRB’ as an intermediary point between the two types of RAM. You must also use the ‘_flock’ system variable (at 0x43E) to lock out other attempted uses of this buffer.

Each physical hard disk drive must contain a boot sector. The boot sector for hard disk drives is the same as floppies except for the following locations:

<table>
<thead>
<tr>
<th>Name</th>
<th>Offset</th>
<th>Contents</th>
</tr>
</thead>
<tbody>
<tr>
<td>hd_siz</td>
<td>0x01C2</td>
<td>This is a Motorola format LONG that indicates the number of physical 512-byte sectors on the device.</td>
</tr>
<tr>
<td>Partition Header #0</td>
<td>0x01C6</td>
<td>This section contains a 12 BYTE partition information block for the first logical partition.</td>
</tr>
<tr>
<td>Partition Header #1</td>
<td>0x01D2</td>
<td>This section contains a 12 BYTE partition information block for the second logical partition.</td>
</tr>
<tr>
<td>Partition Header #2</td>
<td>0x1DE</td>
<td>This section contains a 12 BYTE partition information block for the third logical partition.</td>
</tr>
<tr>
<td>Partition Header #3</td>
<td>0x1EA</td>
<td>This section contains a 12 BYTE partition information block for the fourth logical partition.</td>
</tr>
<tr>
<td>bst_st</td>
<td>0x1F6</td>
<td>This is a Motorola format LONG that indicates the sector offset to the bad sector list (from the beginning of the physical disk).</td>
</tr>
<tr>
<td>bst_cnt</td>
<td>0x01FA</td>
<td>This is a Motorola format LONG that indicates the number of 512-byte sectors reserved for the bad sector list.</td>
</tr>
</tbody>
</table>

The partition information block is defined as follows:

<table>
<thead>
<tr>
<th>Name</th>
<th>Offset</th>
<th>Contents</th>
</tr>
</thead>
<tbody>
<tr>
<td>p_flg</td>
<td>0x00</td>
<td>This is a BYTE size bit field indicating the partition state. If bit 0 is set, the partition exists, otherwise it does not. If bit 7 is set, the partition is bootable, otherwise it is not. Bits 1-6 are unused.</td>
</tr>
<tr>
<td>p_id</td>
<td>0x01</td>
<td>This is a three BYTE field that indicates the partition type as follows:</td>
</tr>
<tr>
<td></td>
<td></td>
<td><strong>Contents</strong></td>
</tr>
<tr>
<td></td>
<td></td>
<td>‘GEM’</td>
</tr>
<tr>
<td></td>
<td></td>
<td>‘BGM’</td>
</tr>
<tr>
<td></td>
<td></td>
<td>‘XGM’</td>
</tr>
<tr>
<td>p_st</td>
<td>0x04</td>
<td>This is a Motorola format LONG that indicates the start of the partition as an offset specified in 512-byte sectors.</td>
</tr>
<tr>
<td>p_size</td>
<td>0x08</td>
<td>This is a Motorola format LONG that indicates the size of the partition in 512-byte sectors.</td>
</tr>
</tbody>
</table>
A hard disk may have up to four standard (GEM or BGM) partitions or three standard and one extended (XGM) partition. The first partition of a hard disk must be a standard one.

### Extended Partitions

The first sector of an extended partition contains a standard boot sector with hard disk information except that the `hd_siz`, `bst_st`, and `bst_cnt` fields are unused. At least one, but no more than two (not necessarily the first two), partition headers are used. The first partition header is the same as described above except that `p_st` describes the offset from the beginning of the extended partition rather than the beginning of the physical disk.

If another partition needs to be linked, the second partition block should contain ‘XGM’ in its `p_id` field and an offset to the next extended partition in `p_st`.

### The Bad Sector List

The bad sector list is a group of three-byte entries describing which physical sectors on the hard disk are unusable. The first three-byte entry contains the number of bad sectors recorded. The second three-byte entry is a checksum and when added to the entire bad sector list byte wise should cause the list to **BYTE** sum to 0xA5. If this is not the case then the bad sector list is considered bad itself.

### The Serial Port

Application writers who develop communication programs will need to use some of the special functions the **XBIOS** provides for control of the serial port(s). Older Atari computers support only one serial port connected by the Multi-Function Peripheral (MFP) chip.

The Atari TT030 contains two MFP chips to provide two serial ports and one Serial Communications Chip (SCC) which controls two more serial ports. One of the SCC ports, however, can be switched over to control a Localtalk compatible network port as follows:

Switch to Serial 2 Connector:

```
Ongibit(0x80);
```

Switch to LAN connector:

```
Offgibit(0x7F);
```

The Mega STe is similar to the TT030, however, it has only one MFP chip to provide one less serial device.

The Atari Falcon030 uses a SCC chip to drive its single serial port and networking port. The Falcon030 does contain a MFP chip but it does not control any of the serial device hardware. The MFP’s ring indicator has, however, been wired across the SCC to provide compatibility with older applications.
Serial Port Mapping

**BIOS** input and output calls to device #1 and **XBIOS** calls which configure the serial port always refer to the currently ‘mapped’ device as set with **Bconmap()**. The Modem CPX allows a user to map any installed device as the default. A program which is aware of the extra ports on newer machines can access them through their own **BIOS** device number as follows:

<table>
<thead>
<tr>
<th>Device Number</th>
<th>Mega ST</th>
<th>TT030</th>
<th>Falcon030</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>Currently mapped device. DEV_AUX</td>
<td>Currently mapped device. DEV_AUX</td>
<td>Currently mapped device. DEV_AUX</td>
</tr>
<tr>
<td>6</td>
<td>Modem 1 (ST MFP) DEV_MEGAMODEM1</td>
<td>Modem 1 (ST MFP) DEV_TTMODEM1</td>
<td>—</td>
</tr>
<tr>
<td>7</td>
<td>Modem 2 (SCC B) DEV_MEGAMODEM2</td>
<td>Modem 2 (SCC B) DEV_TTMODEM2</td>
<td>Modem (SCC B) DEV_FALCONMODEM</td>
</tr>
<tr>
<td>8</td>
<td>Serial/LAN (SCC A) DEV_MEGALAN</td>
<td>Serial 1 (TT MFP) DEV_TTSERIAL1</td>
<td>LAN (SCC A) DEV_FALCONLAN</td>
</tr>
<tr>
<td>9</td>
<td>—</td>
<td>Serial 2/LAN (SCC A) DEV_TTLAN</td>
<td>—</td>
</tr>
</tbody>
</table>

**Configuring the Serial Port**

**Rsconf()** and **Iorec()** set the communication mode and input/output buffers of the currently mapped serial port. You should note that while some ports support transfer rates of greater than 19200 baud, this is the limit of the **Rsconf()** call. Other rates must currently be set in hardware (or with the **Fcntl()** when **MiNT** is present).

**MFP Interrupts**

Each MFP chip supports a number of interrupts used by the serial port and other system needs. The function **Mfpint()** should be used to set define a function in your application that handles one of these interrupts. **Jenabint()** and **Jdisint()** are used to enable/disable these interrupts respectively.

All MFP interrupt calls only work on ST compatible MFP serial ports. The RS-232 ring indicator is the only interrupt that has been wired through the MFP on a Falcon. Because of this, the ring indicator interrupt is the only RS-232 interrupt that may be changed with **Mfpint()** on a Falcon.

**SCC Interrupts**

The **XBIOS** functions used for setting MFP interrupts do not affect the SCC interrupts regardless of the **Bconmap()** mapping. Refer to the memory map for the location of SCC interrupt registers.

**Printer Control**

The **XBIOS** contains two functions used for controlling printers. Both functions are very outdated and should not be relied on in any ST.
**Scrdmp()** triggers the built-in **ALT-HELP** screen dump code. **Prtblk()** enables the built-in screen dump routine of the ST printing only the desired block to an Atari or Epson dot-matrix printer.

**Setprt()** configures the built-in screen dump routine as to the basic configuration of the attached printer.

### Other XBIOS Functions

**NVMaccess()** accesses the non-volatile RAM present in the TT, Mega STe, and Falcon030. You should not read or write to this area as all of its locations are currently reserved.

The functions **Settime()** and **Gettime()** set the **BIOS** time and date. As of **TOS 1.02**, they also update the **GEMDOS** time as well.

Besides the sound capabilities of the **XBIOS** when running on a Falcon, the function **Dosound()** generates music on any Atari computer using the FM sound generator. The function works at the interrupt level processing a ‘sound command list’ you specify. It can be used to reproduce a single tone or a complete song in as many as three parts of harmony.

**Random()** generates a pseudo-random number using a built-in algorithm whose seed comes from the system 60kHz clock.

**Ssbrk()** is used by the operating system to reserve system RAM before **GEMDOS** is initialized. It should not be used by application programmers.

**Puntaes()** is useful only when using a disk-loaded version of **TOS**. It clears the OS from RAM and reboots the computer.

**Midiws()** is a similar function to **Ikbdws()** in that it writes to the MIDI controller. It is more useful at transferring large amounts of MIDI data than **Bconout()**.

The **Dbmsg()** **XBIOS** call is added by supporting debuggers as a method of transferring debugging messages between the application and debugger. The Atari Debugger (DB) currently supports this interface.

### XBIOS Function Calling Procedure

**XBIOS** system functions are called via the TRAP #14 exception. Function arguments are pushed onto the current stack (user or supervisor) in reverse order followed by the function opcode. The calling application is responsible for correctly resetting the stack pointer after the call.

The **XBIOS**, like the **BIOS** may utilize registers D0-D2 and A0-A2 as scratch registers and their contents should not be depended upon at the completion of a call. In addition, the function opcode placed on the stack will be modified.
The following example for Getrez() illustrates calling the XBIOS from assembly language:

```
movlw $04, -(sp)
trap #14
addlw #6, sp
```

A ‘C’ binding for a generic XBIOS handler would be as follows:

```
xbios:

; Save the return code from the stack
movel (sp)+, trpl4ret
trap #14
movel trpl4ret, -(sp)
ret

.bss
trpl4ret:
    .ds.l 1
```

The XBIOS is re-entrant to three levels, however there is no depth checking performed so interrupt handlers should avoid intense XBIOS usage. In addition, no disk or printer usage should be attempted from the system timer interrupt, critical error, or process-terminate handlers.

### Calling the XBIOS from an Interrupt

The BIOS and XBIOS are the only two OS sub-systems which may be called from an interrupt handler. Precisely one interrupt handler at a time may use the XBIOS as shown in the following code segment:

```
savptr equ $4A2
savamt equ $23*2

myhandler:
    subl #savamt, savptr
    ; BIOS calls may be performed here
    addl #savamt, savptr
    retl ; (or rts?)
```

Certain XBIOS calls are not re-entrant because they call GEMDOS routines. The Setscreen() function, and any DSP function which loads data from disk should not be attempted during an interrupt.

It is not possible to use this method to call XBIOS functions during an interrupt when running under MultiTOS.
Bconmap()

LONG Bconmap( devno )
WORD devno;

Bconmap() maps a serial device to BIOS device #1. It is also used to add serial device drivers to the system.

**Opcode**

44 (0x2C)

**Availability**

To reliably check that Bconmap() is supported, the TOS version must be 1.02 or higher and the following function should return a TRUE value.

```c
#define BMAP_EXISTS 0

BOOL IsBconmap( VOID )
{
    return (Bconmap(0) == BMAP_EXISTS);
}
```

**Parameters**

The value of devno has the following effect:

<table>
<thead>
<tr>
<th>Name</th>
<th>devno</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>BMAP_CHECK</td>
<td>0</td>
<td>Verify the existence of the call (systems without Bconmap() will return the function opcode 44).</td>
</tr>
<tr>
<td></td>
<td>1-5</td>
<td>These are illegal values (will return 0).</td>
</tr>
<tr>
<td>See XBIOS Serial Port Mapping for constants.</td>
<td>6-</td>
<td>Redefine BIOS device 1 (the GEMDOS ‘aux:’ device) to map to the named serial device. All Bcon...(1,...), Rsconf(), and Iorec() calls will return information for the named device. Returns the old value.</td>
</tr>
<tr>
<td>BMAP_INQUIRE</td>
<td>-1</td>
<td>Don’t change anything, simply return the old value.</td>
</tr>
<tr>
<td>BMAP_MAPTAB</td>
<td>-2</td>
<td>Return a pointer to the serial device vector table (see below).</td>
</tr>
</tbody>
</table>

**Binding**

```assembly
move.w devno,-(sp)
mov.w #$2C,-(sp)
trap #14
addq.l #4,sp
```

**Return Value**

See above.

**Caveats**

You should never install the 38th device (BIOS device number 44). It would be indistinguishable from the case where Bconmap() was unavailable. In the unlikely event that this case arises, you should install two new devices and assign your new device to the second one.

All current versions of Falcon030 TOS (4.00 – 4.04) contain a bug that prevents
the **BIOS** from accessing the extra available devices. A patch program named FPATCH2.PRG is available from Atari Corporation to correct this bug in software.

**COMMENTS**

To add a serial device to the table, use **Bconmap(-2)** to return a pointer to a **BCONMAP** structure. `maptab` points to a list of **MAPTAB** structures (the first entry in **MAPTAB** is the table for device number 6). The list will contain `maptabsize` devices. Allocate a block of memory large enough to store the old table plus your new entry and copy the old table and your new device structure there making sure to increment `maptabsize`. Finally, alter `maptab` to point to your new structure.

```c
typedef struct
{
    WORD (*Bconstat)();
    LONG (*Bconin)();
    LONG (*Bcostat)();
    VOID (*Bconout)();
    ULONG (*Rsconf)();
    IOREC *iorec; /* See Iorec() */
} MAPTAB;

typedef struct
{
    MAPTAB *maptab;
    WORD maptabsize;
} BCONMAP;
```

**SEE ALSO** **Bconin()**, **Bconout()**, **Rsconf()**, **Iorec()**

---

**Bioskeys()**

**VOID Bioskeys( VOID )**

**Bioskeys()** is used to reset to the power-up defaults of the keyboard configuration tables.

**OPCODE** 24 (0x18)

**AVAILABILITY** All TOS versions.

**BINDING**

```
move.w  #$18,-(sp)
trap    #$14
addq.l  #$4,sp
```

**COMMENTS** This call is only necessary to restore changes made by modifying the tables given by **Keytbl()**.
Blitmode()

WORD Blitmode( mode )
WORD mode;

Blitmode() detects a hardware BLiTTER chip and can alter its configuration if present.

_OPCODE 64 (0x40)

_AVAILABILITY This call is available as of TOS 1.02.

_PARAMETERS mode is used to set the BLiTTER configuration. If mode is BLIT_INQUIRE (-1), the call will return the current state of the BLiTTER without modifying its state. To change the method of OS blit operations, call Blitmode() with one of the following values:

<table>
<thead>
<tr>
<th>Name</th>
<th>mode</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>BLIT_SOFT</td>
<td>0</td>
<td>If set, use hardware BLiTTER chip, otherwise use software routines.</td>
</tr>
<tr>
<td>BLIT_HARD</td>
<td>1</td>
<td>If set, hardware BLiTTER chip is available.</td>
</tr>
</tbody>
</table>

_BINDING move.w mode,-(sp)
move.w #$40,-(sp)
trap #14
addq.l #4,sp

_RETURN VALUE Blitmode() returns the old mode value. Bit #0 of mode contains the currently set blitter mode as shown above. Bit #1 is set to indicate the presence of a hardware blitter chip or clear if no blitter chip is installed.

 COMMENTS You should use this call once to verify the existence of the BLiTTER prior to attempting to change its configuration.

Buffoper()

LONG Buffoper( mode )
WORD mode;

Buffoper() sets/read the state of the hardware sound system.

_OPCODE 136 (0x88)
**Availability**

Available if ‘_SND’ cookie has third bit set.

**Parameters**

*mode* is a bit array which may be composed of all or none of the following flags indicating the desired sound system state as follows:

<table>
<thead>
<tr>
<th>Name</th>
<th>Bit Mask</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>PLAY_ENABLE</td>
<td>0x01</td>
<td>Enable DMA Sound Playback. The sound must have been previously identified to the XBIOS with the Buffptr() function.</td>
</tr>
<tr>
<td>PLAY_REPEAT</td>
<td>0x02</td>
<td>Setting this flag will cause any sound currently playing or started as a result of this call to be looped indefinitely (until Buffoper(0) is used).</td>
</tr>
<tr>
<td>RECORD_ENABLE</td>
<td>0x04</td>
<td>Enable DMA Sound Recording. The sound must have been previously identified to the XBIOS with the Buffptr() function.</td>
</tr>
<tr>
<td>RECORD_REPEAT</td>
<td>0x08</td>
<td>Setting this flag during a record will cause the recording to continue indefinitely within the currently set recording buffer (as set by Buffptr())</td>
</tr>
</tbody>
</table>

Alternately, calling this function with a *mode* parameter of SND_INQUIRE (-1) will return a bit mask indicating the current sound system state as shown above.

**Binding**

```
move.w mode,-(sp)
move.w #$88,-(sp)
trap #14
addq.l #4,sp
```

**Return Value**

Buffoper() normally returns 0 for no error or non-zero otherwise (except in inquire mode as indicated above).

**Comments**

The sound system uses a 32 bit FIFO. The FIFO is only guaranteed to be clear when the record enable bit is clear. When transferring new data to the record buffers, the record enable bit should be cleared to flush the FIFO.

**See Also**

Setbuffer()

---

**Buffptr()**

LONG Buffptr( sptr )
SBUF PTR *sptr;

Buffptr() returns the current position of the playback and record pointers.

**Opcode**

141 (0x8D)
**Availability**
Available if ‘_SND’ cookie has third bit set.

**Parameter**
`sptr` is a pointer to a `SBUF PTR` structure which is filled in with the current pointer values. `SBUF PTR` is defined as follows:

```c
typedef struct
{
   VOIDP playptr;
   VOIDP recordptr;
   VOIDP reserved1;
   VOIDP reserved2;
} SBUF PTR;
```

**Binding**
```
pea  sptr
move.w #$8d,-(sp)
trap  #14
addq.l #6,sp
```

**Return Value**
`Buffptr()` returns 0 if the operation was successful or non-zero otherwise.

**See Also**
`Setbuffer()`, `Buffoper()`

---

**Cursconf()**

```c
WORD Cursconf( mode, rate )
```

**Return Value**
Cursconf() configures the VT-52 cursor.

**Opcode**
21 (0x15)

**Availability**
All TOS versions.

**Parameters**
`mode` defines the operation as follows:

<table>
<thead>
<tr>
<th>Name</th>
<th>mode</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>CURS_HIDE</td>
<td>0</td>
<td>Hide cursor.</td>
</tr>
<tr>
<td>CURS_SHOW</td>
<td>1</td>
<td>Show cursor.</td>
</tr>
<tr>
<td>CURS_BLINK</td>
<td>2</td>
<td>Enable cursor blink.</td>
</tr>
<tr>
<td>CURS_NOBLINK</td>
<td>3</td>
<td>Disable cursor blink.</td>
</tr>
<tr>
<td>CURS_SETRATE</td>
<td>4</td>
<td>Set blink rate to <code>rate</code>.</td>
</tr>
<tr>
<td>CURS_GETRATE</td>
<td>5</td>
<td>Return current blink rate.</td>
</tr>
</tbody>
</table>

**Binding**
```
move.w rate,-(sp)
move.w mode,-(sp)
move.w #$15,-(sp)
```
trap #14
addq.l #6, sp

RETURN VALUE
Cursconf() only returns a meaningful value under mode 5 in which it returns the current blink rate.

COMMENTS
The blink rate is specified in number of vertical blanks per blink.

Dbmsg()

VOID Dbmsg( rsrvd, msg_num, msg_arg )

WORD rsrvd, msg_num;
LONG msg_arg;

Dbmsg() allows special debugging messages to be sent to a resident debugger application.

OPCODE
11 (0x0B)

AVAILABILITY
The only debugger that currently supports this call is the Atari Debugger.

PARAMETERS
rsrvd is currently reserved and should always be 5. msg_num is the message number which you want to send to the debugging host. Values of 0x0000 to 0xEFFF are reserved for applications to define. Values of 0xF000 to 0xFFFF are reserved for special debugging messages.

If msg_num is in the application defined range, it and the LONG contained in msg_arg will be displayed by the debugger and the application will be halted.

If msg_num is between 0xF001 and 0xF0FF inclusive then msg_arg is interpreted as a character pointer pointing to a string to be output by the debugger and debugging to halt. The string length is determined by the low byte of msg_num. If msg_num is DB_NULLSTRING (0xF000), the string will be output until a NULL is reached.

If msg_num is DB_COMMAND (0xF100), msg_arg is interpreted as a character pointer to a string containing a debugger command. The command format is specific to the debugger which you are running.

A useful example of this format when running under the Atari debugger allows a string to be output to the debugger without terminating debugging as shown in the following example:

Dbmsg( 5, DB_COMMAND, “echo ‘Debugging Message’;g” );
The Atari Debugger only understands the value **DB_COMMAND** (0xF100) for **msg_num** as of version 3.

Though it is normally harmless to run an application with embedded debugging messages when no debugger is present in the system, distribution versions of applications should have these instructions removed.

**Devconnect()**

LONG Devconnect( source, dest, clk, prescale, protocol )

**WORD source, dest, clk, prescale, protocol;**

**Devconnect()** attaches a source device in the sound system to one or multiple destination devices through the use of the connection matrix.

**OPCODE**

139 (0x8B)

**AVAILABILITY**

Available if ‘_SND’ cookie has third bit set.

**PARAMETERS**

**source** indicates the source device to connect as follows:

<table>
<thead>
<tr>
<th>Name</th>
<th>source</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>DMAPLAY</td>
<td>0</td>
<td>DMA Playback</td>
</tr>
<tr>
<td>DSPXMIT</td>
<td>1</td>
<td>DSP Transmit</td>
</tr>
<tr>
<td>EXTINP</td>
<td>2</td>
<td>External Input</td>
</tr>
<tr>
<td>ADC</td>
<td>3</td>
<td>Microphone/Yamaha PSG</td>
</tr>
</tbody>
</table>

**dest** is a bit mask which is used to choose which destination devices to connect as follows:

<table>
<thead>
<tr>
<th>Name</th>
<th>Mask</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>DMAREC</td>
<td>0x01</td>
<td>DMA Record</td>
</tr>
<tr>
<td>DSPRECV</td>
<td>0x02</td>
<td>DSP Receive</td>
</tr>
<tr>
<td>EXTOUT</td>
<td>0x04</td>
<td>External Out</td>
</tr>
<tr>
<td>DAC</td>
<td>0x08</td>
<td>DAC (Headphone or Internal Speaker)</td>
</tr>
</tbody>
</table>

**clk** is the clock the source device will use as follows:
### Name | clk | Meaning
--- | --- | ---
CLK_25M | 0 | Internal 25.175 MHz clock
CLK_EXT | 1 | External clock
CLK_32M | 2 | Internal 32 MHz clock

`prescale` chooses the source clock prescaler. Sample rate is determined by the formula:

\[
rate = \frac{\text{clockrate}}{256 \times \text{prescale} + 1}
\]

Valid prescaler values for the internal CODEC using the 25.175 MHz clock are:

### Name | prescale | Meaning/Sample Rate
--- | --- | ---
CLK_COMPAT | 0 | TT030/STe compatibility mode. Use prescale value set with `Soundcmd()`.
CLK_50K | 1 | 49170 Hz
CLK_33K | 2 | 32880 Hz
CLK_25K | 3 | 24585 Hz
CLK_20K | 4 | 19668 Hz
CLK_16K | 5 | 16390 Hz
CLK_12K | 7 | 12292 Hz
CLK_10K | 9 | 9834 Hz
CLK_8K | 11 | 8195 Hz

`protocol` sets the handshaking mode. A value of `HANDSHAKE` (0) enables handshaking, `NO_SHAKE` (1) disables it. When transferring sound or video data through the CODEC it is usually recommended that handshaking be disabled. When incoming data must be 100% error free, however, handshaking should be enabled.

#### Binding
```assembly
move.w protocol,-(sp)
movw w prescale,-(sp)
mov.w clk,-(sp)
mov.w dest,-(sp)
mov.w source,-(sp)
mov.w #$8B,-(sp)
trap #14
lea 12(sp),sp
```

#### Return Value
`Devconnect()` returns 0 if the operation was successful or non-zero otherwise.

#### Caveats
Setting the prescaler to an invalid value will result in a mute condition.

---

**The Atari Compendium**
DMAread()

LONG DMAread( sector, count, buf, dev )
LONG sector;
WORD count;
VOIDP buf;
WORD dev;

DMAread() reads raw sectors from an ACSI or SCSI device.

OPCODE
42 (0x2A)

AVAILABILITY
This call is available as of TOS version 2.00.

PARAMETERS
sector specifies the sector number to begin reading at. count specifies the number of sectors to read. buf is a pointer to the address where incoming data will be stored. dev specifies the device to read from as follows:

<table>
<thead>
<tr>
<th>dev</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>0-7</td>
<td>ACSI devices 0-7</td>
</tr>
<tr>
<td>8-15</td>
<td>SCSI devices 0-7</td>
</tr>
</tbody>
</table>

BINDING
move.w dev,-(sp)
pea buf
move.w count,-(sp)
move.l sector,-(sp)
move.w #$2A,-(sp)
trap #14
lea 14(sp),sp

RETURN VALUE
DMAread() returns 0 if the operation was successful or a negative BIOS error code otherwise.

CAVEATS
SCSI devices will write data until the device exits its data transfer phase. Since this call is not dependent on sector size, you should ensure that the buffer is large enough to hold sectors from devices with large sectors (CD-ROM = 2K, for example).

COMMENTS
ACSI transfers must be done to normal RAM. If you need to read sectors into alternative RAM, use the 64KB pointer found with the ‘_FRB’ cookie as an intermediate transfer point while correctly managing the ‘_flock’ system variable.

SCSI transfers on the TT030 do not actually use DMA. Handshaking is used to
transfer bytes individually. This means that alternative RAM may be used. The Falcon030 uses DMA for SCSI transfers making transfers to alternative RAM illegal.

**SEE ALSO** DMAwrite(), Rwabs()

---

**DMAwrite()**

`LONG DMAwrite( sector, count, buf, dev )`

`LONG sector;`

`WORD count;`

`VOIDP buf;`

`WORD dev;`

DMAwrite() writes raw sectors to ACSI or SCSI devices.

**OPCoded**

43 (0x2B)

**Availability**

TOS versions >= 2.00

**Parameters**

`sector` is the starting sector number to write data to. `count` is the number of sectors to write. `buf` defines the starting address of the data to write. `dev` is the device number as specified in DMAread().

**Binding**

```
move.w dev, -(sp)
pea buf
move.w count, -(sp)
move.l sector, -(sp)
move.w #$2B, -(sp)
trap #14
lea 14(sp), sp
```

**Return Value**

DMAwrite() returns 0 if successful or a negative BIOS error code otherwise.

**Comments**

ACSI transfers must be done from normal RAM. If you need to read sectors into alternative RAM, use the 64KB pointer found with the ‘_FRB’ cookie as an intermediate transfer point while correctly managing the ‘_flock’ system variable.

SCSI transfers do not actually use DMA. Handshaking is used to transfer bytes individually.

**See Also**

DMAread(), Rwabs()
Dosound()

VOID Dosound( cmdlist )
char *cmdlist;

Dosound() initializes and starts an interrupt driven sound playback routine using
the PSG.

O P C O D E
32 (0x20)

A V A I L A B I L I T Y
All TOS versions.

P A R A M E T E R S
If cmdlist is positive, it will be interpreted as a pointer to a character array
containing a sequential list of commands required for the sound playback. Each
command is executed in order and has a meaning as follows:

<table>
<thead>
<tr>
<th>Command Byte</th>
<th>Meaning</th>
</tr>
</thead>
</table>
| 0x00 - 0x0F  | Select a PSG register (the register number is the command byte). The
next byte in the list will be loaded into this register. See Appendix I for a
detailed listing of registers, musical frequencies, and sound durations. |
| 0x80         | Store the next byte in a temporary register for use by command 0x81. |
| 0x81         | Three bytes follow this command. The first is the PSG register to load with
the value in the temporary register (set with command 0x80). The second
is a signed value to add to the temporary register until the value in the third
byte is met. |
| 0x82         | If a 0 follows this command, this signals the end of processing, otherwise
the value indicates the number of 50Hz ticks to wait until the processing of
the next command. |

Passing the value DS_INQUIRE (-1) for cmdlist will cause the pointer to the
current sound buffer to be returned or NULL if no sound is currently playing.

B I N D I N G
pea  cmdlist
move.w #20,-(sp)
trap  #14
addq.l #6,sp

C A V E A T S
This routine is driven by interrupts. Do not use an array created on the stack to
store the command list that may go out of scope before the sound is complete.

This function will cause the OS to crash under MultiTOS versions prior to 1.08 if
every running application is not set to ‘Supervisor’ or ‘Global’ memory
protection.

Dosound( DS_INQUIRE ) will cause the OS to crash under MultiTOS versions
1.08 and below.
Dsp_Available()

VOID Dsp_Available( xavail, yavail )
LONG *xavail, *yavail;

Dsp_Available() returns the amount of free program space in X and Y DSP memory.

OPCODE 106 (0x6A)

AVAILABILITY This call is only available if the fifth bit of the ‘_SND’ cookie is set.

PARAMETERS Upon return, the longwords pointed to by xavail and yavail will contain the length of memory (in bytes) available for DSP programs and subroutines.

BINDING

Pea yavail
Pea xavail
move.w #$6A,-(sp)
trap #14
lea 10(sp),sp

SEE ALSO Dsp_Reserve()

Dsp_BlkBytes()

VOID Dsp_BlkBytes( data_in, size_in, data_out, size_out )
UBYTE *data_in;
LONG size_in;
UBYTE *data_out;
LONG size_out;

Dsp_BlkBytes() transfers a block of unsigned character data to the DSP and returns the output from the running program or subroutine.

OPCODE 124 (0x7C)

AVAILABILITY This call is only available if the fifth bit of the ‘_SND’ cookie is set.

PARAMETERS data_in is a pointer to an unsigned character array which is transferred to the DSP. size_in is the length (in bytes) of data to transfer.

data_out is a pointer to the unsigned character array to be filled in from the low byte of the DSP’s transfer register. size_out is the length (in bytes) of the output buffer array.
Dsp_BlkHandShake

VOID Dsp_BlkHandShake( data_in, size_in, data_out, size_out )

char *data_in;
LONG size_in;
char *data_out;
LONG size_out;

Dsp_BlkHandShake() handshakes a block of bytes to the DSP and returns the output generated by the running subroutine or program.

Opcode
97 (0x61)

Availability
This call is only available if the fifth bit of the '_SND' cookie is set.

Parameters
data_in is a pointer to data being sent to the DSP. size_in specifies the number of DSP words of data to be transferred. Dsp_GetWordSize() can be used to determine the number of bytes that occur for a DSP word.

data_out is a pointer to the buffer to which processed data will be returned from the DSP. size_out indicates the number of DSP words to transfer.

Binding

move.l size_out,-(sp)
pea data_out
move.l size_in,-(sp)
pea data_in
move.w #$7C,-(sp)
trap #14
lea 18(sp),sp

Caveats
No handshaking is performed with this call. Error sensitive data should be transferred with Dsp_BlkHandShake().

Comments
Bytes are not sign extended before transfer. Also, due to the length of static memory in the DSP, size_in and size_out should not exceed 65536.

See Also
Dsp_BlkWords()
**Dsp_BlkUnpacked()**

VOID Dsp_BlkUnpacked( data_in, size_in, data_out, size_out )

LONG *data_in;
LONG size_in;
LONG *data_out;
LONG size_out;

Dsp_BlkUnpacked() transfers data to the DSP from a longword array. Data processed by the running subroutine or program is returned.

**OPCODE**

98 (0x62)

**AVAILABILITY**

This call is only available if the fifth bit of the ‘_SND’ cookie is set.

**PARAMETERS**

*data_in* is a pointer to an array of LONGs from which data is transferred to the DSP. As many bytes are transferred from each LONG as there are bytes in a DSP WORD. For example, if Dsp_GetWordSize() returns 3, the lower three bytes of each LONG are transferred into each DSP WORD.

*size_in* represents the number of LONGs in the array to transfer. *data_out* is a pointer to an array of LONGs *size_out* in length in which data sent from the DSP is returned.

**BINDING**

move.l size_out,-(sp)
pa data_out
move.l size_in,-(sp)
pa data_in
move.w #$62,-(sp)
trap #14
lea 18(sp),sp

**CAVEATS**

This function only works with DSP’s which return 4 or less from Dsp_GetWordSize(). In addition, no handshaking is performed with this call. Data which is sensitive to errors should use Dsp_BlkHandShake().

**SEE ALSO**

Dsp_DoBlock()
Dsp_BlkWords()

VOID Dsp_BlkWords( data_in, size_in, data_out, size_out )
WORD *data_in;
LONG size_in;
WORD *data_out;
LONG size_out;

Dsp_BlkWords() transfers an array of WORDs to the DSP and returns the output generated by the running subroutine or program.

_OPCODE 123 (0x7B)

_AVAILABILITY This call is only available if the fifth bit of the ‘_SND’ cookie is set.

_PARAMETERS data_in is a pointer to the WORD array to be transferred to the DSP. size_in is the length (in WORDs) of data to transfer.

data_out is a pointer to the WORD array to be filled in during the data output phase of the DSP from the middle and low bytes of the transfer register. size_out is the length (in WORDs) of the buffer for the output array.

_BINDING move.l size_out,-(sp)
pea data_out
move.l size_in,-(sp)
pea data_in
move.w #$7B,-(sp)
trap #14
lea 18(sp),sp

_CAVEATS No handshaking is performed with this call. Data which is sensitive to errors should use Dsp_BlkHandShake().

_COMMENTS WORDs are sign extended before transfer. Also, due to the length of static memory in the DSP, size_in and size_out should not exceed 32768.

SEE ALSO Dsp_BlkBytes()
Dsp_DoBlock()

VOID Dsp_DoBlock( data_in, size_in, data_out, size_out )
char *data_in;
LONG size_in;
char *data_out;
LONG size_out;

Dsp_DoBlock() transfers byte-wise packed data to the DSP and returns the data processed by the running subroutine or program.

_OPCODE 96 (0x60)

_AVAILABILITY This call is only available if the fifth bit of the ‘_SND’ cookie is set.

_PARAMETERS data_in is a character array containing data to transfer to the DSP. size_in specifies the number of DSP words to transfer. For example, if Dsp_GetWordSize() returns 3, the first 3 bytes from data_in are stored in the first DSP word, the next 3 bytes are stored in the next DSP word and so on.

data_out points to a character array where the output will be stored in a similar manner. size_out represents the size of this array.

_BINDING move.l size_out,-(sp)
pea data_out
move.l size_in,-(sp)
pea data_in
move.w #$60,-(sp)
trap #14
lea 18(sp),sp

 CAVEATS No handshaking is performed with this call. Data which is sensitive to errors should use Dsp_BlkHandShake().

SEE ALSO Dsp_BlkHandShake()
**Dsp_ExecBoot()**

VOID Dsp_ExecBoot( codeptr, codesize, ability )

char *codeptr;
LONG codesize;
WORD ability;

**Dsp_ExecBoot()** completely resets the DSP and loads a new bootstrap program into the first 512 DSP words of memory.

**OPCODE**

110 (0x6E)

**AVAILABILITY**

This call is only available if the fifth bit of the ‘_SND’ cookie is set.

**PARAMETERS**

*codeptr* points to the beginning of the DSP program data to be transferred. *codesize* indicates the size (in DSP words) of program data to transfer. *ability* indicates the bootstrapper’s unique ability code.

**BINDING**

move.w ability,-(sp)
mov.l codesize,-(sp)
pea codeptr
move.w #$6E,-(sp)
trap #14
lea 12(sp),sp

**COMMENTS**

This call is only designed for special development and testing purposes. Use of this call takes over control of the DSP system.

This call is limited to transferring up to 512 DSP words of code.

**SEE ALSO**

Dsp_LoadProg(), Dsp_ExecProg()

---

**Dsp_ExecProg()**

VOID Dsp_ExecProg( codeptr, codesize, ability )

char *codeptr;
LONG codesize;
WORD ability;

**Dsp_ExecProg()** transfers a DSP program stored in binary format in memory to the DSP and executes it.

**OPCODE**

109 (0x6D)
### Availability

This call is only available if the fifth bit of the ‘_SND’ cookie is set.

### Parameters

* `codeptr` points to the start of the binary program in memory. `codesize` indicates the number of DSP words to transfer. `ability` indicates the program’s unique ability code.

### Binding

```
move.w  ability,-(sp)
move.l  codesize,-(sp)
pea     codeptr
move.w  #$6D,-(sp)
trap    #14
lea     12(sp),sp
```

### Comments

`codesize` should not exceed the amount of memory reserved by the `Dsp_Reserve()` call.

### See Also

`Dsp_LoadProg()`, `Dsp_Reserve()`

---

### Dsp_FlushSubroutines()

VOID Dsp_FlushSubroutines( VOID )

`Dsp_FlushSubroutines()` removes all subroutines from the DSP.

### Opcode

115 (0x73)

### Availability

This call is only available if the fifth bit of the ‘_SND’ cookie is set.

### Binding

```
move.w  #$73,-(sp)
trap    #14
addq.l  #2,sp
```

### Comments

This call should only be used when a program requires more memory than is returned by `Dsp_Available()`.

### See Also

`Dsp_Available()`

---

### Dsp_GetProgAbility()

WORD Dsp_GetProgAbility( VOID )

`Dsp_GetProgAbility()` returns the current ability code for the program currently residing in DSP memory.

### Opcode

114 (0x72)
Dsp_GetWordSize() – 4.41

**Availability**
This call is only available if the fifth bit of the ‘_SND’ cookie is set.

**Binding**
move.w $72,-(sp)
trap $14
addq.l #2,sp

**Return Value**
Dsp_GetProgAbility() returns the **WORD** ability code for the current program loaded in the DSP.

**Comments**
If you know the defined ability code of the program you wish to use, you can use this call to see if the program already exists on the DSP and avoid reloading it.

**See Also**
Dsp_InqSubrAbility()

---

**Dsp_GetWordSize()**

**WORD** Dsp_GetWordSize( VOID )

Dsp_GetWordSize() returns the size of a DSP word in the installed Digital Signal Processor.

**Opcode**
103 (0x67)

**Availability**
This call is only available if the fifth bit of the ‘_SND’ cookie is set.

**Binding**
move.w $67,-(sp)
trap $14
addq.l #2,sp

**Return Value**
Dsp_GetWordSize() returns the number of bytes per DSP word.

**Comments**
This value is useful with many DSP-related XBIOS calls to provide upward compatibility as the DSP hardware is not guaranteed to remain the same.

---

**Dsp_Hf0()**

**WORD** Dsp_Hf0( flag )
**WORD** flag;

Dsp_Hf0() reads/writes to bit #3 of the HSR.

**Opcode**
119 (0x77)
This call is only available if the fifth bit of the ‘_SND’ cookie is set.

flag has three legal values as follows:

<table>
<thead>
<tr>
<th>Name</th>
<th>flag</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>HF_CLEAR</td>
<td>0</td>
<td>Clear bit #3 of the DSP’s HSR.</td>
</tr>
<tr>
<td>HF_SET</td>
<td>1</td>
<td>Set bit #3 of the DSP’s HSR.</td>
</tr>
<tr>
<td>HF_INQUIRE</td>
<td>-1</td>
<td>Return the current value of bit #3 of the DSP’s HSR.</td>
</tr>
</tbody>
</table>

move.w flag,-(sp)
move.w #$77,-(sp)
trap #14
addq.l #4,sp

If flag is HF_INQUIRE (-1), Dsp_Hf0() returns the current state of bit #3 of the HSR register.

WORD Dsp_Hf1(flag)

Dsp_Hf1() reads/writes to bit #4 of the HSR.

120 (0x78)

This call is only available if the fifth bit of the ‘_SND’ cookie is set.

flag has three legal values as follows:

<table>
<thead>
<tr>
<th>Name</th>
<th>flag</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>HF_CLEAR</td>
<td>0</td>
<td>Clear bit #4 of the DSP’s HSR.</td>
</tr>
<tr>
<td>HF_SET</td>
<td>1</td>
<td>Set bit #4 of the DSP’s HSR.</td>
</tr>
<tr>
<td>HF_INQUIRE</td>
<td>-1</td>
<td>Return the current value of bit #4 of the DSP’s HSR.</td>
</tr>
</tbody>
</table>

move.w flag,-(sp)
move.w #$78,-(sp)
trap #14
addq.l #4,sp

If flag is HF_INQUIRE (-1), Dsp_Hf1() returns the current state of bit #4 of the HSR register.
**Dsp_Hf2()**

**WORD Dsp_Hf2( VOID )**

*Dsp_Hf2()* returns the current status of bit #3 of the DSP’s HCR.

**OPCODE**

121 (0x79)

**AVAILABILITY**

This call is only available if the fifth bit of the ‘_SND’ cookie is set.

**BINDING**

```assembly
move.w #$79,-(sp)
trap   #14
addq.l #2,sp
```

**RETURN VALUE**

*Dsp_Hf2()* returns the current setting of bit #3 of the HCR register (valid values are 0 or 1).

**SEE ALSO**

*Dsp_Hf3()*

---

**Dsp_Hf3()**

**WORD Dsp_Hf3( VOID )**

*Dsp_Hf3()* returns the current status of bit #4 of the DSP’s HCR.

**OPCODE**

122 (0x7A)

**AVAILABILITY**

This call is only available if the fifth bit of the ‘_SND’ cookie is set.

**BINDING**

```assembly
move.w #$7A,-(sp)
trap   #14
addq.l #2,sp
```

**RETURN VALUE**

*Dsp_Hf3()* returns the current setting of bit #4 of the HCR register (valid values are 0 or 1).

**SEE ALSO**

*Dsp_Hf2()*
Dsp_HStat()

BYTE Dsp_Hstat( VOID )

Dsp_HStat() returns the value of the DSP’s ICR register.

OPCODE 125 (0x7D)

AVAILABILITY This call is only available if the fifth bit of the ‘_SND’ cookie is set.

BINDING

\[
\begin{align*}
\text{move.w} & \quad \#7D,-(sp) \\
\text{trap} & \quad \#14 \\
\text{addq.l} & \quad \#2,sp
\end{align*}
\]

RETURN VALUE Dsp_Hstat() returns an 8-bit value representing the current state of the DSP’s ICR register as follows:

<table>
<thead>
<tr>
<th>Name</th>
<th>Bit</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>ICR_RXDF</td>
<td>0</td>
<td>ISR Receive data register full (RXDF)</td>
</tr>
<tr>
<td>ICR_TXDE</td>
<td>1</td>
<td>ISR Transmit data register empty (TXDE)</td>
</tr>
<tr>
<td>ICR_TRDY</td>
<td>2</td>
<td>ISR Transmitter ready (TRDY)</td>
</tr>
<tr>
<td>ICR_HF2</td>
<td>3</td>
<td>ISR Host flag 2 (HF2)</td>
</tr>
<tr>
<td>ICR_HF3</td>
<td>4</td>
<td>ISR Host flag 3 (HF3)</td>
</tr>
<tr>
<td>—</td>
<td>5</td>
<td>Reserved</td>
</tr>
<tr>
<td>ICR_DMA</td>
<td>6</td>
<td>ISR DMA Status (DMA)</td>
</tr>
<tr>
<td>ICR_HREQ</td>
<td>7</td>
<td>ISR Host Request (HREQ)</td>
</tr>
</tbody>
</table>

Dsp_InqSubrAbility()

WORD Dsp_InqSubrAbility( ability )

WORD ability;

Dsp_InqSubrAbility() determines if a subroutine with the specified ability code exists in the DSP.

OPCODE 117 (0x75)

AVAILABILITY This call is only available if the fifth bit of the ‘_SND’ cookie is set.

PARAMETERS ability is the ability code you wish to check.

BINDING

\[
\begin{align*}
\text{move.w} & \quad \text{ability},-(sp) \\
\text{move.w} & \quad \#75,-(sp)
\end{align*}
\]
Dsp_InStream() – 4.45

RETURN VALUE  Dsp_InqSubrAbility() returns a handle to the subroutine if found or 0 if not.

SEE ALSO   Dsp_RunSubroutine()

Dsp_InStream()

VOID Dsp_InStream( data_in, block_size, num_blocks, blocks_done )
char *data_in;
LONG block_size;
LONG num_blocks;
LONG *blocks_done;

Dsp_InStream() passes data to the DSP via an interrupt handler.

OPCODE  99 (0x63)

AVAILABILITY  This call is only available if the fifth bit of the ‘_SND’ cookie is set.

PARAMETERS  data_in is a pointer to unsigned character data which should be transferred to the DSP. block_size indicates the number of DSP WORDs that will be transferred at each interrupt. num_blocks indicates the number of blocks to transfer.

The LONG pointed to by blocks_done will be constantly updated to let the application know the progress of the transfer.

BINDING  

pea    blocks_done 
move.l num_blocks,-(sp)
move.l block_size,-(sp)
pea    data_in 
move.w #$63,-(sp)
trap   #14 
lea    18(sp),sp

CAVEATS  No handshaking is performed with this call. If the data you are transmitting is error sensitive, use Dsp_BlkHandShake().

COMMENTS  This call is suited for transferring small blocks while other blocks are being prepared for transfer. For larger blocks, Dsp_DoBlock() would be more suitable.

SEE ALSO   Dsp_BlkHandShake(), Dsp_DoBlock()
Dsp_IOStream()

VOID Dsp_IOStream(data_in, data_out, block_insize, block_outsize, num_blocks, blocks_done)
char *data_in, *data_out;
LONG block_insize, block_outsize, num_blocks;
LONG *blocks_done;

Dsp_IOStream() uses two interrupt handlers to transmit and receive data from the DSP.

OPCODE
101 (0x65)

AVAILABILITY
This call is only available if the fifth bit of the ‘_SND’ cookie is set.

PARAMETERS
data_in is a pointer to a buffer in which each output block is placed. data_out is a pointer to a buffer used to receive each data block from the DSP.

block_insize and block_outsize represent the size of the blocks to send and receive, respectively, in DSP WORDs. num_blocks is the total number of blocks to transfer.

The LONG pointed at by blocks_done is constantly updated to indicate the number of blocks actually transferred.

BINDING
pea blocks_done
move.l num_blocks,-(sp)
move.l block_outsize,-(sp)
move.l block_insize,-(sp)
pea data_out
pea data_in
move.w #$65,-(sp)
trap #14
lea 26(sp),sp

CAVEATS
This call makes the assumption that the DSP will be ready to accept a new block as input every time it finishes sending a block back to the host.

COMMENTS
No handshaking is performed with this call. If your data is error-sensitive, you should use Dsp_BlkHandShake().

SEE ALSO
Dsp_InStream(), Dsp_OutStream()
**Dsp_LoadProg()**

WORD Dsp_LoadProg(file, ability, buf)
char *file;
WORD ability;
char *buf;

Dsp_LoadProg() loads a `.LOD' file from disk, transmits it to the DSP, and executes it.

**Opcode**
108 (0x6C)

**Availability**
This call is only available if the fifth bit of the `_SND' cookie is set.

**Parameters**
file is a pointer to a NULL-terminated string containing a valid GEMDOS file specification. ability is the unique ability code that will be assigned to this program. buf should point to a temporary buffer where the DSP will place the binary code it generates. The minimum size of the buffer is determined by the following formula:

\[
3 \times (\#\text{program/data words} + (3 \times \#\text{blocks in program}))
\]

**Binding**

<table>
<thead>
<tr>
<th>Instruction</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>pea</td>
<td>buf</td>
</tr>
<tr>
<td>move.w ability</td>
<td>-(sp)</td>
</tr>
<tr>
<td>pea</td>
<td>file</td>
</tr>
<tr>
<td>move.w #6C</td>
<td>-(sp)</td>
</tr>
<tr>
<td>trap #14</td>
<td></td>
</tr>
<tr>
<td>lea #12(sp),sp</td>
<td></td>
</tr>
</tbody>
</table>

**Return Value**
Dsp_LoadProg() returns a 0 if successful or -1 otherwise.

**Comments**
Before loading you should determine if a program already exists on the DSP with your chosen ability with Dsp_GetProgAbility().

**See Also**
Dsp_LoadSubroutine()
Dsp_LoadSubroutine()

WORD Dsp_LoadSubroutine( ptr, size, ability )
char *ptr;
LONG size;
WORD ability;

Dsp_LoadSubroutine() transmits subroutine code to the DSP.

OPCODE 116 (0x74)

AVAILABILITY This call is only available if the fifth bit of the '_SND' cookie is set.

PARAMETERS ptr points to a memory buffer which contains DSP binary subroutine code. size is the length of code to transfer (specified in DSP words). ability is the WORD identifier for the unique ability of this subroutine.

BINDING
move.w ability,-(sp)
move.l size,-(sp)
pea ptr
move.w #$74,-(sp)
trap #14
lea 12(sp),sp

RETURN VALUE Dsp_LoadSubroutine() returns the handle assigned to the subroutine or 0 if an error occurred.

COMMENTS DSP subroutines have many restrictions and you should see the previous discussion of the DSP for more information.

SEE ALSO Dsp_RunSubroutine(), Dsp_InqSubrAbility()

Dsp_Lock()

WORD Dsp_Lock( VOID )

Dsp_Lock() locks the use of the DSP to the calling application.

OPCODE 104 (0x68)

AVAILABILITY This call is only available if the fifth bit of the '_SND' cookie is set.

BINDING
move.w #$68,-(sp)
trap #14
addq.1 #2,sp
**RETURN VALUE**  
*Dsp_Lock()* returns a 0 if successful or -1 if the DSP has been locked by another application.

**COMMENTS**  
*Dsp_Lock()* should be performed before each use of the DSP to prevent other applications from modifying DSP memory or flushing subroutines. A corresponding *Dsp_Unlock()* should be issued at the end of each usage. You should limit the amount of time the DSP is locked so other applications may utilize it.

**SEE ALSO**  
*Dsp_Unlock()*

---

**Dsp_LodToBinary()**

LONG Dsp_LodToBinary(file, codeptr )
char *file,*codeptr;

*Dsp_LodToBinary()* reads a `.LOD` file and converts the ASCII data to binary program code ready to be sent to the DSP via *Dsp_ExecProg()* or *Dsp_ExecBoot()*.

**_OPCODE**  
111 (0x6F)

**AVAILABILITY**  
This call is only available if the fifth bit of the `._SND` cookie is set.

**PARAMETERS**  
*file* is a character pointer to a null-terminated GEMDOS file specification.  
*codeptr* should point to a large enough buffer to hold the resulting binary program code.

**BINDING**

```
pea codeptr
pea file
move.w #$6F,-(sp)
trap #14
lea 10(sp),sp
```

**RETURN VALUE**  
*Dsp_LodToBinary()* returns the size of the resulting program code in DSP words or a negative error code.

**SEE ALSO**  
*Dsp_ExecProg(), Dsp_LoadProg()*
Dsp_MultBlocks()

VOID Dsp_MultBlocks( numsend, numreceive, sendblks, receiveblks )
LONG numsend, numreceive;
DSPBLOCK *sendblks, *receiveblks;

Dsp_MultBlocks() transmit and receive multiple blocks of DSP data of varying size.

OPCDE

127 (0x7F)

AVAILABILITY

This call is only available if the fifth bit of the ‘_SND’ cookie is set.

PARAMETERS

numsend and numreceive indicate the number of blocks of DSP data to send and receive respectively. sendblks and receiveblks are both pointers to arrays of type DSPBLOCK which contain information for each block. DSPBLOCK is defined as follows:

typedef struct
{
#define BLOCK_LONG 0
#define BLOCK_WORD 1
#define BLOCK_UBYTE 2
  /* 0 = LONGs, 1 = WORDs, 2 = UBYTEs */
  WORD blocktype;
  /* Num elements in block */
  LONG blocksize;
  /* Start address of block */
  VOIDP blockaddr;
} DSPBLOCK;

BINDING

pea receiveblks
pea sendblks
move.l numreceive,-(sp)
move.l numsend,-(sp)
move.w #$7F,-(sp)
trap #14
lea 20(sp),sp

CAVEATS

No handshaking is performed with this call. To transfer blocks with handshaking use Dsp_BlkHandShake().
VOID Dsp_OutStream( data_out, block_size, num_blocks, blocks_done )
char *data_out;
LONG block_size;
LONG num_blocks;
LONG *blocks_done;

Dsp_OutStream() transfers data from the DSP to a user-specified buffer using interrupts.

OPCODE 100 (0x64)

AVAILABILITY This call is only available if the fifth bit of the ‘_SND’ cookie is set.

PARAMETERS This call transfers data from the DSP to the buffer pointed to by data_out via an interrupt handler. block_size specifies the number of DSP WORDs to be transferred and num_blocks specifies the number of blocks to transfer.

The LONG pointed to by blocks_done will be constantly updated by the interrupt handler to indicate the number of blocks successfully transferred. The process is complete when blocks_done is equal to num_blocks.

BINDING
pea blocks_done
move.l num_blocks,-(sp)
move.l block_size,-(sp)
pea data_out
move.w #$64,-(sp)
trap #1
lea 18(sp),sp

SEE ALSO Dsp_DoBlock(), Dsp_MultBlocks(), Dsp_InStream()

VOID Dsp_RemoveInterrupts( mask )
WORD mask;

Dsp_RemoveInterrupts() turns off the generation of DSP interrupts.

OPCODE 102 (0x66)

AVAILABILITY This call is only available if the fifth bit of the ‘_SND’ cookie is set.
**PARAMETERS**

*mask* is an **WORD** bit mask indicating which interrupts to turn off composed of one or both of the following values:

<table>
<thead>
<tr>
<th>Name</th>
<th>Mask</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>RTS_OFF</td>
<td>0x01</td>
<td>Disable DSP Ready to Send Interrupts</td>
</tr>
<tr>
<td>RTR_OFF</td>
<td>0x02</td>
<td>Disable DSP Ready to Receive Interrupts</td>
</tr>
</tbody>
</table>

**BINDING**

```
move.w      mask,-(sp)
move.w      #$66,-(sp)
trap        #14
addq.l      #4,sp
```

**COMMENTS**

This call is used to terminate interrupts when an interrupt driven block transfer function does not terminate as expected (this will occur when less than the expected number of blocks is returned) and to shut off interrupts installed by `Dsp_SetVectors()`.

**SEE ALSO**

`Dsp_SetVectors()`

---

**Dsp_RequestUniqueAbility()**

**WORD Dsp_RequestUniqueAbility( VOID )**

`Dsp_RequestUniqueAbility()` generates a random ability code that is currently not in use.

**OPCODE**

113 (0x71)

**AVAILABILITY**

This call is only available if the fifth bit of the '_SND' cookie is set.

**BINDING**

```
move.w      #$71,-(sp)
trap        #14
addq.l      #2,sp
```

**RETURN VALUE**

`Dsp_RequestUniqueAbility()` returns a unique ability code to assign to a subroutine or program.

**COMMENTS**

Using this function allows you to call `Dsp_InqSubrAbility()` and `Dsp_GetProgAbility()` to determine if the DSP code your application has already loaded is still present (i.e. has not been flushed by another application).

**SEE ALSO**

`DspInqSubrAbility()`, `Dsp_GetProgAbility()`
Dsp_Reserve()

WORD Dsp_Reserve( xreserve, yreserve )
LONG xreserve, yreserve;

Dsp_Reserve() reserves DSP memory for program usage.

OPCODE 107 (0x6B)

AVAILABILITY This call is only available if the fifth bit of the '_SND' cookie is set.

PARAMETERS xreserve and yreserve specify the amount of memory (in DSP words) to reserve for a DSP program in X and Y memory space respectively. xreserve and yreserve must include all program/data space so that subroutines do not overwrite your reserved area.

BINDING
move.l yreserve,-(sp)
movel xreserve,-(sp)
movew #$6B,-(sp)
trap #14
lea 10(sp),sp

RETURN VALUE Dsp_Reserve() returns a 0 if the memory was reserved successfully or -1 if not enough DSP memory was available.

COMMENTS If this call fails you should call Dsp_FlushSubroutines() and then retry it. If it fails a second time, the DSP lacks enough memory space to run your program.

Dsp_RunSubroutine()

WORD Dsp_RunSubroutine( handle )

Dsp_RunSubroutine() begins execution of the specified subroutine.

OPCODE 118 (0x76)

AVAILABILITY This call is only available if the fifth bit of the '_SND' cookie is set.

PARAMETERS handle is the WORD identifier of the DSP subroutine to engage.

BINDING
move.w handle,-(sp)
movew #$76,-(sp)
trap #14
addq.l #4,sp
Dsp_RunSubroutine() returns a 0 if successful or a negative code indicating failure.

Dsp_LoadSubroutine()

Dsp_SetVectors()

VOID Dsp_SetVectors( receiver, transmitter )
VOID (*receiver)();
LONG (*transmitter)();

Dsp_SetVectors() sets the location of application interrupt handlers that are called when the DSP is either ready to send or receive data.

OPCODE 126 (0x7E)

AVAILABILITY This call is only available if the fifth bit of the ‘_SND’ cookie is set.

PARAMETERS receiver is the address of an interrupt handler which is called when the DSP is ready to send a DSP word of data or NULLFUNC ( VOID (*)(0L) ) if you do not wish to set this interrupt.

Likewise, transmitter is a pointer to an interrupt handler which is called when the DSP is ready to receive a DSP word of data or NULLFUNC if you do not wish to install a transmitter interrupt.

Any function installed to handle transmitter interrupts should return a LONG which has one of the following values:

<table>
<thead>
<tr>
<th>Name</th>
<th>transmitter Return Value</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>DSPSEND_NOTHING</td>
<td>0x00000000</td>
<td>Do not send any data to the DSP.</td>
</tr>
<tr>
<td>DSPSENDZERO</td>
<td>0xFF000000</td>
<td>Transmit a DSP word of 0 to the DSP.</td>
</tr>
<tr>
<td>—</td>
<td>Any other</td>
<td>Transmit the low 24 bits to the DSP.</td>
</tr>
</tbody>
</table>

BINDING

move.l #transmitter,-(sp)
move.l #receiver,-(sp)
move.w #$7E,-(sp)
trap #14
lea 10(sp),sp

COMMENTS Use Dsp_RemoveInterrupts() to turn off interrupts set with this call.

SEE ALSO Dsp_RemoveInterrupts()
Dsp_TriggerHC()

VOID Dsp_TriggerHC(vector);
WORD vector;

Dsp_TriggerHC() causes a host command set aside for DSP programs to execute.

OPCODE 112 (0x70)

AVAILABILITY This call is only available if the fifth bit of the `_SND` cookie is set.

PARAMETERS vector specifies the vector to execute.

BINDING
move.w vector,-(sp)
move.w #$70,-(sp)
trap #14
addq.l #4,sp

CAVEATS Currently vectors 0x13 and 0x14 are the only vectors available for this purpose. All other vectors are overwritten by the system on program load and are used by the system and subroutines.

Dsp_Unlock()

VOID Dsp_Unlock( VOID )

Dsp_Unlock() unlocks the sound system from use by a process which locked it previously using Dsp_Lock().

OPCODE 105 (0x69)

AVAILABILITY This call is only available if the fifth bit of the `_SND` cookie is set.

BINDING
move.w #$69,-(sp)
trap #14
addq.l #2,sp

SEE ALSO Dsp_Lock()
Dsptristate()

LONG Dsptristate( dspxmit, dsprec )
WORD dspxmit, dsprec;

Dsptristate() connects or disconnects the DSP from the connection matrix.

OPCODE
137 (0x89)

AVAILABILITY
Available if ‘_SND’ cookie has bits 3 and 4 set.

PARAMETERS

dpxmit and dsprec specify whether data being transmitted and/or recorded into
the DSP passes through the connection matrix. A value of DSP_TRISTATE (0)
indicates a ‘tristate’ condition where data is not fed through the matrix. A value of
DSP_ENABLE (1) enables the use of the connection matrix.

BINDING
move.w dsprec,-(sp)
move.w dspxmit,-(sp)
move.w #$89,-(sp)
trap #14
addq.l #6,sp

RETURN VALUE
Dsptristate() returns 0 if no error occurred or non-zero otherwise.

COMMENTS
This call is used in conjunction with Devconnect() to link the DSP to the internal
sound system.

SEE ALSO
Devconnect()

EgetPalette()

VOID EgetPalette( start, count, paldata )
WORD start, count;
WORD *paldata;

EgetPalette() copies the current TT030 color palette data into a specified buffer..

OPCODE
85 (0x55)

AVAILABILITY
This call is available when the high word of the '_VDO' cookie has a value of 2.

PARAMETERS
start gives the index (0-255) of the first color register to copy data into. count
specifies the total number of registers to copy. paldata is a pointer to an array
where the TT030 palette data will be stored. Each WORD will be formatted as
follows:

<table>
<thead>
<tr>
<th>Bits 15-12</th>
<th>Bits 11-8</th>
<th>Bits 7-4</th>
<th>Bits 3-0</th>
</tr>
</thead>
<tbody>
<tr>
<td>Reserved</td>
<td>Red</td>
<td>Green</td>
<td>Blue</td>
</tr>
</tbody>
</table>

**BINDING**

- `pea` palette
- `move.w` count, -(sp)
- `move.w` start, -(sp)
- `move.w` #$55, -(sp)
- `trap` #14
- `lea` #10(sp), sp

**CAVEATS**

This call is machine-dependent to the TT030. It is therefore recommended that `vq_color()` be used in most instances.

**COMMENTS**

Unlike `Setpalette()` this call encodes color nibbles from the most significant to least significant bit (3-2-1-0) as opposed to the compatibility method of 0-3-2-1.

**SEE ALSO**

Esetpalette(), `vq_color()`

---

**EgetShift()**

**WORD** `EgetShift( VOID )`

`EgetShift()` returns the current mode of the video shifter.

**OPCODE**

81 (0x51)

**AVAILABILITY**

This call is available when the high word of the ‘_VDO’ cookie has a value of 2.

**BINDING**

- `move.w` #$51, -(sp)
- `trap` #14
- `addq.l` #2, sp

**RETURN VALUE**

`EgetShift()` returns a **WORD** bit array which is divided as follows:

<table>
<thead>
<tr>
<th>Mask Name</th>
<th>Bit(s)</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>ES_BANK</td>
<td>0–3</td>
<td>These bits determine the current color bank being used by the TT (in all modes with less than 256 colors). The macro <code>ColorBank()</code> as defined below will extract the current bank code.</td>
</tr>
<tr>
<td></td>
<td>—</td>
<td>Unused</td>
</tr>
</tbody>
</table>

The macro `ColorBank(x)` is defined as:

```
#define ColorBank(x) ((x) & ES_BANK)
```
These bits determine the current mode of the TT video shifter as follows:

<table>
<thead>
<tr>
<th>Name</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>ST_LOW</td>
<td>0x000</td>
</tr>
<tr>
<td>ST_MED</td>
<td>0x010</td>
</tr>
<tr>
<td>ST_HIGH</td>
<td>0x020</td>
</tr>
<tr>
<td>TT_MED</td>
<td>0x030</td>
</tr>
<tr>
<td>TT_HIGH</td>
<td>0x060</td>
</tr>
<tr>
<td>TT_LOW</td>
<td>0x070</td>
</tr>
</tbody>
</table>

The current shifter mode code can be extracted with the following macro:

```c
#define ScreenMode(x) ((x) & ES_MODE)
```

This bit determines if the TT video shifter is currently in grayscale mode. The following macro can be used to extract this information:

```c
#define IsGrayMode(x) ((x) & ES_GRAY)
```

If this bit is set, the TT video shifter is currently in smear mode. The following macro can be used to extract this information:

```c
#define IsSmearMode(x) ((x) & ES_SMEAR)
```

**SEE ALSO** EsetGray(), EsetShift(), EsetSmear(), EsetBank()

### EsetBank()

**WORD** EsetBank(bank)

**WORD** bank;

EsetBank() chooses which of 16 banks of color registers is currently active.

**OPCODE** 82 (0x52)

**AVAILABILITY** This call is available when the high word of the ‘_VDO’ cookie has a value of 2.

**PARAMETERS** *bank* specifies the index of the color bank to activate. A value of ESB_INQUIRE (-1) does not change anything but still returns the current bank.

**BINDING**

```assembly
move.w bank, -(sp)
mov.w #52, -(sp)
trap #14
addq.l #4, sp
```

**RETURN VALUE** EsetBank() returns the index of the old blank.
**EsetColor()**

WORD EsetColor( idx, color )

**WORD idx, color;**

EsetColor() sets an individual color in the TT030’s palette.

**OPCODE** 83 (0x53)

**AVAILABILITY** This call is available when the high word of the ‘_VDO’ cookie has a value of 2.

**PARAMETERS**

- **idx** specifies the color index to modify (0-255).
- **color** is a TT030 format color WORD bit array divided as follows:

<table>
<thead>
<tr>
<th>Bits 15-12</th>
<th>Bits 11-8</th>
<th>Bits 7-4</th>
<th>Bits 3-0</th>
</tr>
</thead>
<tbody>
<tr>
<td>Reserved</td>
<td>Red</td>
<td>Green</td>
<td>Blue</td>
</tr>
</tbody>
</table>

If **color** is **EC_INQUIRE** (-1) then the call does not change the register but still returns its value.

**BINDING**

- move.w color,-(sp)
- move.w idx,-(sp)
- move.w #$53,-(sp)
- trap #14
- addq.l #6,sp

**RETURN VALUE**

EsetColor() returns the old value of the color register.

**CAVEATS** This call is machine-dependent to the TT030. It is therefore recommended that **vs_color()** be used instead for compatibility.

**COMMENTS**

Unlike **Setpalette()** this call encodes color nibbles from the most significant to least significant bit (3-2-1-0) as opposed to the compatibility method of 0-3-2-1.

**SEE ALSO**

EsetPalette(), vs_color()
EsetGray()

WORD EsetGray( mode )
WORD mode;

EsetGray() reads/modifies the TT030’s video shifter gray mode bit.

OPCODE
86 (0x56)

AVAILABILITY
This call is available when the high word of the ‘_VDO’ cookie has a value of 2.

PARAMETERS
mode is defined as follows:

<table>
<thead>
<tr>
<th>Name</th>
<th>mode</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>ESG_INQUIRE</td>
<td>-1</td>
<td>Return the gray bit of the video shifter.</td>
</tr>
<tr>
<td>ESG_COLOR</td>
<td>0</td>
<td>Set the video shifter to interpret the lower 16 bits of a palette entry as a TT030 color value (RGB 0-15).</td>
</tr>
<tr>
<td>ESG_GRAY</td>
<td>1</td>
<td>Set the video shifter to interpret the lower 8 bits of a palette entry as a TT030 gray value (0-255)</td>
</tr>
</tbody>
</table>

BINDING
move.w mode,-(sp)
move.w #$56,-(sp)
trap #14
addq.l #4,sp

RETURN VALUE
EsetGray() returns the previous value of the video shifter’s gray bit.

CAVEATS
This call is machine-dependent to the TT030.

SEE ALSO
EsetShift()

EsetPalette()

VOID EsetPalette( start, count, paldata )
WORD start,count;
WORD *paldata;

EsetPalette() copies TT030 color WORDs from the specified buffer into the TT030 Color Lookup Table (CLUT).

OPCODE
84 (0x54)

AVAILABILITY
This call is available when the high word of the ‘_VDO’ cookie has a value of 2.
PARAMETERS

- `start` specifies the index of the starting color register to copy color data to.
- `count` indicates the number of palette `WORDs` to copy.
- `paldata` is a pointer to an array of palette `WORDs` to copy.

BINDING

- `pea palette`
- `move.w count,-(sp)`
- `move.w start,-(sp)`
- `move.w #$54,-(sp)`
- `trap #14`
- `lea 10(sp),sp`

CAVEATS

This call is machine-dependent to the TT030. It is therefore recommended that `vs_color()` be used instead for compatibility.

COMMENTS

For the format of the color `WORDs`, see `EgetPalette()`.

SEE ALSO

`EgetPalette()`, `vq_color()`

---

## EsetShift()

**WORD** EsetShift( **mode** )

**WORD** mode;

EsetShift() reads/modifies the TT030 video shifter.

OPCODE

80 (0x50)

AVAILABILITY

This call is available when the high word of the '_VDO' cookie has a value of 2.

PARAMETERS

- `mode` is a `WORD` bit array which defines the new setting of the video shifter as follows:

<table>
<thead>
<tr>
<th>Name</th>
<th>Bit(s)</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>—</td>
<td>0–3</td>
<td>These bits determine the current color bank being used by the TT</td>
</tr>
<tr>
<td></td>
<td></td>
<td>(in all modes with less than 256 colors).</td>
</tr>
<tr>
<td>—</td>
<td>4–7</td>
<td>Unused</td>
</tr>
<tr>
<td>—</td>
<td>8–10</td>
<td>These bits determine the current mode of the TT video shifter as follows:</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Name</td>
<td>Bit Mask</td>
<td></td>
</tr>
<tr>
<td>ST_LOW</td>
<td>0x0000</td>
<td></td>
</tr>
<tr>
<td>ST_MED</td>
<td>0x0100</td>
<td></td>
</tr>
<tr>
<td>ST_HIGH</td>
<td>0x0200</td>
<td></td>
</tr>
<tr>
<td>TT_MED</td>
<td>0x0300</td>
<td></td>
</tr>
<tr>
<td>TT_HIGH</td>
<td>0x0600</td>
<td></td>
</tr>
<tr>
<td>TT_LOW</td>
<td>0x0700</td>
<td></td>
</tr>
</tbody>
</table>
### EsetSmear()

**WORD** `EsetSmear( mode )`

**WORD** `mode;`

`EsetSmear()` reads/modifies the current state of the video shifter’s smear mode bit.

**OPCODE** 87 (0x57)

**AVAILABILITY** This call is available when the high word of the ‘_VDO’ cookie has a value of 2.

**PARAMETERS** `mode` specifies the action of this call as follows:

<table>
<thead>
<tr>
<th>Name</th>
<th><code>mode</code></th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>ESM_INQUIRE</td>
<td>-1</td>
<td>Return the smear bit of the video shifter.</td>
</tr>
<tr>
<td>ESM_NORMAL</td>
<td>0</td>
<td>Set the video shifter to process video data normally.</td>
</tr>
<tr>
<td>ESM_SMEAR</td>
<td>1</td>
<td>Set the video shifter to repeat the color of the last displayed pixel each time a 0x0000 is read from video memory.</td>
</tr>
</tbody>
</table>

**BINDING**

```
move.w     mode,-(sp)
move.w     #$50,-(sp)
trap       #14
addq.l     #4,sp
```

**RETURN VALUE** `EsetSmear()` returns the prior setting of the video shifter’s smear mode bit.

**SEE ALSO** `Egetshift()`, `EsetShift()`
Flopfmt()

WORD Flopfmt(buf, skew, dev, spt, track, side, intlv, magic, virgin)
VOIDP buf;
WORD *skew;
WORD dev, spt, track, side, intlv;
LONG magic;
WORD virgin;

Flopfmt() formats a specified track on a floppy disk.

OPCODE 10 (0x0A)

AVAILABILITY All TOS versions.

PARAMETERS buf is a pointer to a word-aligned buffer large enough to hold one disk track which is used to build a copy of each sector to write. skew should be NULL for non-interleaved sectors or point to a WORD array containing spt entries which specifies the sector interleave order.

dev specifies which floppy drive to format (‘A’ = FLOP_DRIVEA (0), ‘B’ = FLOP_DRIVEB (1)). spt indicates the number of sectors to format. track indicates which track to format.

side indicates the side to format. intlv should be FLOP_NOSKEW (1) for consecutive sectors or FLOP_SKEW (-1) to interleave the sectors based on the array pointed to by skew.

magic is a fixed magic number which must be FLOP_MAGIC (0x87654321).

virgin is the value to assign to uninitialized sector data (should be FLOP_VIRGIN (0xE5E5)).

BINDING

    move.w virgin,-(sp)
    move.l magic,-(sp)
    move.w intlv,-(sp)
    move.w side,-(sp)
    move.w track,-(sp)
    move.w spt,-(sp)
    move.w dev,-(sp)
    pea skew
    pea buf
    move.w #$0A,-(sp)
    trap #14
    lea 26(sp),sp

RETURN VALUE Flopfmt() returns 0 if the track was formatted successfully or non-zero otherwise.
Also, upon exit, \textit{buf} will be filled in with a \texttt{WORD} array of sectors that failed formatting terminated by an entry of 0. If no errors occurred then the first \texttt{WORD} of \textit{buf} will be 0.

**COMMENTS**

The steps required to format a floppy disk are as follows:

1. Call \texttt{Flopfmt()} to format the disk as desired.
2. Call \texttt{Protobt()} to create a prototype boot sector in memory.
3. Call \texttt{Flopwr()} to write the prototype boot sector to track 0, side 0, sector 1.

Interleaved sector formatting is only possible as of \texttt{TOS} 1.2. \textit{skew} should be set to \texttt{NULL} and \textit{intlv} should be set to \texttt{FLOP_NOSKEW} under \texttt{TOS} 1.0.

Specifying an \textit{intlv} value of \texttt{FLOP_SKEW} and a \textit{skew} array equalling \{ 1, 2, 3, 4, 5, 6, 7, 8, 9 \} is the same as specifying an \textit{intlv} value of \texttt{FLOP_NOSKEW}. To accomplish a 9 sector 2:1 interleave you would use a \textit{skew} array which looked like: \{ 1, 6, 2, 7, 3, 8, 4, 9, 5 \}.

The ‘_FDC’ cookie (if present) contains specific information regarding the installed floppy drives. The lower three bytes of the cookie value contain a three-letter code indicating the manufacturer of the drive (Atari is 0x415443 ‘ATC’). The high byte determines the capabilities of the highest density floppy drive currently installed as follows:

<table>
<thead>
<tr>
<th>Name</th>
<th>Value</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>FLOPPY_DSDD</td>
<td>0</td>
<td>Standard Density (720K)</td>
</tr>
<tr>
<td>FLOPPY_DSHD</td>
<td>1</td>
<td>High Density (1.44MB)</td>
</tr>
<tr>
<td>FLOPPY_DSED</td>
<td>2</td>
<td>Extra High Density (2.88MB)</td>
</tr>
</tbody>
</table>

To format a high density diskette, multiple the \textit{spt} parameter by 2. To format a extra-high density diskette, multiply the \textit{spt} parameter by 4.

This call forces a ‘media changed’ state on the device which will be returned on the next \texttt{Mediach()} or \texttt{Rwabs()} call.

**SEE ALSO**

\texttt{Floprate()}, \texttt{Floprd()}, \texttt{Flopwr()}

**Floprate()**

\texttt{WORD Floprate( dev, rate )}
\texttt{WORD dev, rate;}

Floprate() sets the seek rate of the specified floppy drive.
**Opcode** 41 (0x29)

**Availability** Available on all TOS versions except 1.00.

**Parameters** 

*dev* indicates the floppy drive whose seek rate you wish to modify (‘A:’ = FLOP_DRIVEA (0), ‘B:’ = FLOP_DRIVEB (1)). *rate* specifies the seek rate as follows:

<table>
<thead>
<tr>
<th>Name</th>
<th>rate</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>FRATE_6</td>
<td>0</td>
<td>Set seek rate to 6ms</td>
</tr>
<tr>
<td>FRATE_12</td>
<td>1</td>
<td>Set seek rate to 12ms</td>
</tr>
<tr>
<td>FRATE_2</td>
<td>2</td>
<td>Set seek rate to 2ms</td>
</tr>
<tr>
<td>FRATE_3</td>
<td>3</td>
<td>Set seek rate to 3ms</td>
</tr>
</tbody>
</table>

A *rate* value of FRATE_INQUIRE (-1) will inquire the current seek rate without modifying it.

**Binding**

```
move.w   rate, -(sp)
move.w   dev, -(sp)
move.w   #$29, -(sp)
trap     #14
addq.l   #6, sp
```

**Return Value** Floprate() returns the prior seek rate for the specified drive.

**Comments** TOS version 1.00 can have its seek rates set by setting the system variable (_seekrate (WORD *)0x440) to the desired value (as in *rate*). Note that you can only set the seek rate for both drives in this manner.

---

**Floprd()**

WORD Floprd( buf, rsrvd, dev, sector, track, side, count )
VOIDP buf;
LONG rsrvd;

**Opcode** 8 (0x08)

**Availability** All TOS versions.

**Parameters** *buf* points to a word-aligned buffer where the data to be read will be stored. *rsrvd* is currently unused and should be 0. *dev* specifies the floppy drive to read from.
(‘A:’ = FLOP_DRIVEA (0), ‘B:’ = FLOP_DRIVEB (1)). The function reads count physical sectors starting at sector sector, track track, side side.

**BINDING**

```assembly
move.w count, -(sp)
move.w side, -(sp)
move.w track, -(sp)
move.w sector, -(sp)
move.w dev, -(sp)
move.l rsrvd, -(sp)
pea buf
move.w #$08, -(sp)
trap #14
lea 20 (sp), sp
```

**RETURN VALUE**

Floprd() returns 0 if the operation was successful or non-zero otherwise.

**CAVEATS**

This function reads sectors in physical order (not taking interleave into account). Use Rwabs() to read logical sectors.

**SEE ALSO**

Flopwr(), Flopfmt(), Flopver(), Rwabs()

---

**Flopver()**

```assembly
19 (0x13)

PARAMETERS

buf is a pointer to a word-aligned buffer to compare the sector against. rsrvd is unused and should be 0. dev specifies the drive to verify (‘A:’ = FLOP_DRIVEA (0), ‘B:’ = FLOP_DRIVEB (1)). This function verifies count sectors starting at sector sector, track track, side side.

**BINDING**

```assembly
move.w count, -(sp)
move.w side, -(sp)
move.w track, -(sp)
move.w sector, -(sp)
move.w dev, -(sp)
move.l rsrvd, -(sp)
pea buf
move.w #$13, -(sp)
trap #14
lea 20 (sp), sp
```
**RETURN VALUE** Flopwr() returns 0 if all sectors were successfully verified or a non-zero value otherwise.

**CAVEATS** This function only verifies sectors in physical order.

**COMMENTS** As with Flopfmt(), upon the return of the function, buf is filled in with a WORD array containing a list of any sectors which failed. The array is terminated with a NULL.

**SEE ALSO** Flopwr(), Flopfmt()

---

**Flopwr()**

WORD Flopwr(buf, rsrvd, dev, sector, track, side, count)

VOIDP buf;
LONG rsrvd;
WORD dev, sector, track, side, count;

Flopwr() writes sectors to the floppy drive.

**OPCODE** 9 (0x09)

**AVAILABILITY** All TOS versions.

**PARAMETERS** buf is a pointer containing data to write. rsrvd is currently unused and should be set to 0. dev specifies the floppy drive to write to (‘A:’ = 0,’B:’ = 1). This function writes count sectors starting at sector sector, track track, side side.

**BINDING**

- move.w count,-(sp)
- move.w side,-(sp)
- move.w track,-(sp)
- move.w sector,-(sp)
- move.w dev,-(sp)
- move.l rsrvd,-(sp)
- pea buf
- move.w #$09,-(sp)
- trap #14
- lea 20(sp),sp

**RETURN VALUE** Flopwr() returns 0 if the sectors were successfully written or non-zero otherwise.

**CAVEATS** This function writes sectors in physical order only (ignoring interleave). Use Rwabs() to write sectors in logical order.

**COMMENTS** If this call is used to write to track 0, sector 1, side 0, the device will enter a
‘media might have changed’ state indicated upon the next `Rwabs()` or `Mediach()` call.

**SEE ALSO**

`Floprd()`, `Flofmt()`, `Flopver()`, `Rwabs()`

---

**Getrez()**

**WORD Getrez( VOID )**

Getrez() returns a machine-dependent code representing the current screen mode/ratio.

**OPCODE**

4 (0x04)

**AVAILABILITY**

All TOS versions.

**BINDING**

move.w #$04,-(sp)
trap #14
addq.l #2,sp

**RETURN VALUE**

Getrez() returns a value representing the current video display mode. To find the value you will receive back based on current Atari manufactured video hardware, refer to the following chart:

<table>
<thead>
<tr>
<th>Screen Dimension:</th>
<th>Colors:</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>2</td>
</tr>
<tr>
<td>320x200</td>
<td>X</td>
</tr>
<tr>
<td>320x240</td>
<td>X</td>
</tr>
<tr>
<td>320x480</td>
<td>X</td>
</tr>
<tr>
<td>640x200</td>
<td>1</td>
</tr>
<tr>
<td>640x400</td>
<td>2</td>
</tr>
<tr>
<td>640x480</td>
<td>2</td>
</tr>
<tr>
<td>1280x960</td>
<td>6</td>
</tr>
</tbody>
</table>

† This value varies. TT030 Medium resolution returns a value of 4, however, the Falcon returns a value of 2.

**CAVEATS**

This call is *extremely* machine-dependent. Dependence on this call will make your program incompatible with third-party video boards and future hardware. Use the values returned by `v_opnvwk()` to determine screen attributes.

**COMMENTS**

Use of this call in preparing to call `v_opnvwk()` is acceptable and must be done to specify the correct fonts to load from GDOS.
Gettime()

LONG Gettime( VOID )

Gettime() returns the current IKBD time.

OPCODE
23 (0x17)

AVAILABILITY
All TOS versions.

BINDING
move.w #$17,-(sp)  
trap #14  
addq.l #2,sp

RETURN VALUE
Gettime() returns a LONG bit array packed with the current IKBD time as follows:

<table>
<thead>
<tr>
<th>Bits</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>0-4</td>
<td>Seconds/2 (0-29)</td>
</tr>
<tr>
<td>5-10</td>
<td>Minute (0-59)</td>
</tr>
<tr>
<td>11-15</td>
<td>Hour (0-23)</td>
</tr>
<tr>
<td>16-20</td>
<td>Day (1-31)</td>
</tr>
<tr>
<td>21-24</td>
<td>Month (1-12)</td>
</tr>
<tr>
<td>25-31</td>
<td>Year-1980 (0-127)</td>
</tr>
</tbody>
</table>

The return value can be represented in a C structure as follows:

typedef struct
{  
    unsigned year:7;  
    unsigned month:4;  
    unsigned day:5;  
    unsigned hour:5;  
    unsigned minute:6;  
    unsigned second:5;  
} BIOS_TIME;

SEE ALSO
Settime(), Tgettime(), Tgetdate()
Giaccess()

WORD Giaccess( data, register )
WORD data, register;

Giaccess() reads/sets the registers of the FM sound chip and Port A/B peripherals.

OPCODE
28 (0x1C)

AVAILABILITY
All TOS versions.

PARAMETERS
The lower eight bits of data are written to the register selected by register if the value for register is OR’ed with 0x80 (high bit set). If this bit is not set, data is ignored and the value of the register is returned. register selects the register to read/write to as follows:

<table>
<thead>
<tr>
<th>Name</th>
<th>register</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>PSG_APITCHLOW</td>
<td>0</td>
<td>Set the pitch of the PSG’s channel A to the value in registers 0 and 1.</td>
</tr>
<tr>
<td>PSG_BPITCHHIGH</td>
<td>1</td>
<td>Register 0 contains the lower 8 bits of the frequency and the lower 4 bits of register 1 contain the upper 4 bits of the frequency’s 12-bit value.</td>
</tr>
<tr>
<td>PSG_BPITCHLOW</td>
<td>2</td>
<td>Set the pitch of the PSG’s channel B to the value in registers 0 and 1.</td>
</tr>
<tr>
<td>PSG_BPITCHHIGH</td>
<td>3</td>
<td>Register 0 contains the lower 8 bits of the frequency and the lower 4 bits of register 1 contain the upper 4 bits of the frequency’s 12-bit value.</td>
</tr>
<tr>
<td>PSG_CPITCHLOW</td>
<td>2</td>
<td>Set the pitch of the PSG’s channel C to the value in registers 0 and 1.</td>
</tr>
<tr>
<td>PSG_CPITCHHIGH</td>
<td>3</td>
<td>Register 0 contains the lower 8 bits of the frequency and the lower 4 bits of register 1 contain the upper 4 bits of the frequency’s 12-bit value.</td>
</tr>
<tr>
<td>PSG_NOISEPITCH</td>
<td>6</td>
<td>The lower five bits of this register set the pitch of white noise. The lower the value, the higher the pitch.</td>
</tr>
<tr>
<td>PSG_MODE</td>
<td>7</td>
<td>This register contains an eight bit map which determines various aspects of sound generation. Setting each bit on causes the following actions:</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Name</th>
<th>Bit Mask</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>PSG_ENABLEA</td>
<td>0x01</td>
<td>Chnl A tone enable</td>
</tr>
<tr>
<td>PSG_ENABLEB</td>
<td>0x02</td>
<td>Chnl B tone enable</td>
</tr>
<tr>
<td>PSG_ENABLEC</td>
<td>0x04</td>
<td>Chnl C tone enable</td>
</tr>
<tr>
<td>PSG_NOISEA</td>
<td>0x08</td>
<td>Chnl A white noise on</td>
</tr>
<tr>
<td>PSG_NOISEB</td>
<td>0x10</td>
<td>Chnl B white noise on</td>
</tr>
<tr>
<td>PSG_NOISEC</td>
<td>0x20</td>
<td>Chnl C white noise on</td>
</tr>
<tr>
<td>PSG_PRTAOUT</td>
<td>0x40</td>
<td>Port A: 0 = input 1 = output</td>
</tr>
<tr>
<td>PSG_PRTBOUT</td>
<td>0x80</td>
<td>Port B: 0 - input 1 = output</td>
</tr>
<tr>
<td><strong>Register</strong></td>
<td><strong>Value</strong></td>
<td><strong>Description</strong></td>
</tr>
<tr>
<td>----------------------</td>
<td>-----------</td>
<td>-----------------------------------------------------------------------------------------------------------------------------------------------</td>
</tr>
<tr>
<td>PSG_AVOLUME</td>
<td>8</td>
<td>This register controls the volume of channel A. Values from 0-15 are absolute volumes with 0 being the softest and 15 being the loudest. Setting bit 4 causes the PSG to ignore the volume setting and to use the envelope setting in register 13.</td>
</tr>
<tr>
<td>PSG_BVOLUME</td>
<td>9</td>
<td>This register controls the volume of channel B. Values from 0-15 are absolute volumes with 0 being the softest and 15 being the loudest. Setting bit 4 causes the PSG to ignore the volume setting and to use the envelope setting in register 13.</td>
</tr>
<tr>
<td>PSG_CVOLUME</td>
<td>10</td>
<td>This register controls the volume of channel C. Values from 0-15 are absolute volumes with 0 being the softest and 15 being the loudest. Setting bit 4 causes the PSG to ignore the volume setting and to use the envelope setting in register 13.</td>
</tr>
<tr>
<td>PSG_FREQLOW</td>
<td>11</td>
<td>Register 11 contains the low byte and register 12 contains the high byte of the frequency of the waveform specified in register 13. This value may range from 0 to 65535.</td>
</tr>
<tr>
<td>PSG_FREQHIGH</td>
<td>12</td>
<td></td>
</tr>
<tr>
<td>PSG_ENVELOPE</td>
<td>13</td>
<td>The lower four bits of the register contain a value which defines the envelope waveform of the PSG. The best definition of values is obtained through experimentation.</td>
</tr>
<tr>
<td>PSG_PORTA</td>
<td>14</td>
<td>This register accesses Port A of the Yamaha PSG. It is recommended that the functions <code>Ongibit()</code> and <code>Offgibit()</code> be used to access this register.</td>
</tr>
<tr>
<td>PSG_PORTB</td>
<td>15</td>
<td>This register accesses Port B of the Yamaha PSG. This register is currently assigned to the data in/out line of the Centronics Parallel port.</td>
</tr>
</tbody>
</table>

**Binding**

```
move.w   register,-(sp)
move.w   data,-(sp)
move.w   #$1C,-(sp)
trap     #14
addq.l   #6,sp
```

**Return Value**

`Giaccess()` returns the value of the register in the lower eight bits of the word if `data` was OR’ed with 0x80.

---

**Gpio()**

**LONG Gpio( mode, data )**

**WORD mode, data;**

`Gpio()` reads/writes data over the general purpose pins on the DSP connector.

**OPCODE**

138 (0x8A)

**Availability**

Available if ‘_SND’ cookie has bit 3 set.
**PARAMETERS**

*mode* specifies the meaning of *data* and the return value as follows:

<table>
<thead>
<tr>
<th>Name</th>
<th>mode</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>GPIO_INQUIRE</td>
<td>0</td>
<td>Return the old value.</td>
</tr>
<tr>
<td>GPIO_READ</td>
<td>1</td>
<td>Read the three general purpose pins and return their state in the lower three bits of the returned value. <em>data</em> is ignored.</td>
</tr>
<tr>
<td>GPIO_WRITE</td>
<td>2</td>
<td>Write the lower three bits of <em>data</em> to the corresponding DSP pins. The return value is 0.</td>
</tr>
</tbody>
</table>

**BINDING**

```assembly
move.w data, -(sp)
mov.e mode, -(sp)
mov.w #$8A, -(sp)
trap #14
addq.l #6, sp
```

---

**Ikbdws()**

**VOID Ikbdws(len, buf)**

**WORD len;**

**CHAR *buf;**

Ikbdws() writes the contents of a buffer to the intelligent keyboard controller.

**OPCODE**

25 (0x19)

**AVAILABILITY**

All TOS versions.

**PARAMETERS**

This function writes *len* + 1 characters from buffer *buf* to the IKBD.

**BINDING**

```assembly
pea buf
mov.e len, -(sp)
mov.w #$19, -(sp)
trap #14
addq.l #8, sp
```
Initmous()

VOID Initmous( mode, param, vec )
WORD mode;
VOIDP param;
VOID (*vec)();

Initmous() determines the method of handling IKBD mouse packets from the system.

**OPCODE**
0 (0x00)

**AVAILABILITY**
All TOS versions.

**PARAMETERS**

*mode* indicates a IKBD reporting mode and defines the meaning of the other parameters as listed below. *hand* points to a mouse packet handler which is called when each mouse packet is sent. Register A0 contains the mouse packet address when called.

<table>
<thead>
<tr>
<th>Name</th>
<th>mode</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>IM_DISABLE</td>
<td>0</td>
<td>Disable mouse reporting.</td>
</tr>
<tr>
<td>IM_RELATIVE</td>
<td>1</td>
<td>Enable relative mouse reporting mode. Packets report offsets from the previous mouse position. In this mode, <em>param</em> is a pointer to a structure as follows:</td>
</tr>
</tbody>
</table>

```c
struct param
{
    BYTE topmode;
    BYTE buttons;
    BYTE xparam;
    BYTE yparam;
}
```

*topmode* is **IM_YBOT** (0) to indicate that Y=0 means bottom of the screen. A *topmode* value of **IM_YTOP** (1) indicates that Y=0 means the top of the screen.

*buttons* is a bit array which affect the way mouse clicks are handled. A value of **IM_KEYS** (4) causes mouse buttons to generate keycodes rather than mouse packets. A value of **IM_PACKETS** (3) causes the absolute mouse position to be reported on each button press.

*xparam* and *yparam* specify the number of mouse X/Y increments between position report packets.

This mode is the default mode of the AES and VDI.
Enable absolute mouse reporting mode. Packets report actual screen positions. In this mode, `param` is a pointer to a structure as follows:

```c
struct param {
    BYTE topmode;
    BYTE buttons;
    BYTE xparam;
    BYTE yparam;
    WORD xmax;
    WORD ymax;
    WORD xinitial;
    WORD yinitial;
};
```

topmode, buttons, xparam, and yparam are the same as for mode 2.

xmax and ymax specify the maximum X and Y positions the mouse should be allowed to move to. xinitial and yinitial specify the mouse’s initial location.

Enable mouse keycode mode. Keyboard codes for mouse movements are sent rather than actual mouse packets. `param` is handled the same as in mode 1.

Changing the mouse packet handler to anything but relative mode will cause the AES and VDI to stop receiving mouse input.

**See Also**

Kbdvbase()

### `iorec()`

**IOREC *iorec( dev )**

**WORD dev;**

`iorec()` returns the address in memory of system data structures relating to the buffering of input data.
PARAMETERS  

`dev` specifies the device to return information about as follows:

<table>
<thead>
<tr>
<th>Name</th>
<th>dev</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>IO_SERIAL</td>
<td>0</td>
<td>Currently mapped serial device (see Bconmap())</td>
</tr>
<tr>
<td>IO_KEYBOARD</td>
<td>1</td>
<td>Keyboard</td>
</tr>
<tr>
<td>IO_MIDI</td>
<td>2</td>
<td>MIDI</td>
</tr>
</tbody>
</table>

BINDING

```assembly
move.w dev, - (sp)  
move.w #$0E, - (sp)  
trap #14             
addq.l #4, sp         
```

RETURN VALUE  

`Iorec()` returns the address of an IOREC array with either one element (Keyboard or MIDI) or two elements (RS-232 - 1st = input, 2nd = output). The IOREC structure is defined as follows:

```c
typedef struct
{
    /* start of buffer */
    char *ibuf;

    /* size of buffer */
    WORD ibufsize;

    /* head index mark of buffer */
    WORD ibufhd;

    /* tail index mark of buffer */
    WORD ibuftl;

    /* low-water mark of buffer */
    WORD ibuflow;

    /* high-water mark of buffer */
    WORD ibufhi;
} IOREC;
```

SEE ALSO  

Bconmap()

Jdisint()

VOID Jdisint( intno )  
WORD intno;

Jdisint() disables an MFP interrupt.

OPCODE  

26 (0x1A)
**Availability**

All TOS versions.

**Parameters**

`intno` specifies the interrupt to disable (see `Mfpint()` for a list).

**Binding**

```assembly
move.w intno,-(sp)
mov.w #$1A,-(sp)
trap #14
addq.l $4,sp
```

**See Also**

`Jenabint()`, `Mfpint()`

---

**Jenabint()**

VOID `Jenabint( intno )`

WORD `intno`;

`Jenabint()` enables an MFP interrupt.

**Opcode**

27 (0x1B)

**Availability**

All TOS versions.

**Parameters**

`intno` specifies the interrupt to enable (see `Mfpint()` for a list).

**Binding**

```assembly
move.w intno,-(sp)
mov.w #$1B,-(sp)
trap #14
addq.l $4,sp
```

**See Also**

`Jdsint()`, `Mfpint()`

---

**Kbdvbase()**

KBDVECS *Kbdvbase( VOID )

`Kbdvbase()` returns a pointer to a system structure containing a ‘jump’ table to system vector handlers.

**Opcode**

34 (0x22)

**Availability**

All TOS versions.

**Binding**

```assembly
move.w #$22,-(sp)
trap #14
```
addq.l #2,sp

**RETURN VALUE**

Kbdvbase() returns a pointer to a system structure `KBDVECS` which is defined as follows:

```c
typedef struct
{
    VOID (*midivec)( UBYTE data ); /* MIDI Input */
    VOID (*vkbderr)( UBYTE data ); /* IKBD Error */
    VOID (*vmiderr)( UBYTE data ); /* MIDI Error */
    VOID (*statvec)(char *buf); /* IKBD Status */
    VOID (*mousevec)(char *buf); /* IKBD Mouse */
    VOID (*clockvec)(char *buf); /* IKBD Clock */
    VOID (*joyvec)(char *buf); /* IKBD Joystick */
    VOID (*midisys)( VOID ); /* Main MIDI Vector */
    VOID (*ikbdsys)( VOID ); /* Main IKBD Vector */
    char ikbdstate; /* See below */
} KBDVECS;
```

`midivec` is called with the received data byte in d0. If an overflow error occurred on either ACIA, `vkbderr` or `vmiderr` will be called, as appropriate by `midisys` or `ikbdsys` with the contents of the ACIA data register in d0.

`statvec`, `mousevec`, `clockvec`, and `joyvec` all are called with the address of the packet in register A0.

`midisys` and `ikbdsys` are called by the MFP ACIA interrupt handler when a character is ready to be read from either the midi or keyboard ports.

`ikbdstate` is set to the number of bytes remaining to be read by the `ikbdsys` handler from a multiple-byte status packet.

**COMMENTS**

If you intercept any of these routines you should either JMP through the old handler or RTS.

**SEE ALSO**

Initmous()

---

**Kbrate()**

```
WORD Kbrate( delay, rate )
WORD delay, rate;
```

Kbrate() reads/modifies the keyboard repeat/delay rate.

**OPCODE**

35 (0x23)

**AVAILABILITY**

All TOS versions.
PARAMETERS

*delay* specifies the amount of time (in 50Hz ticks) before a key begins repeating. *rate* indicates the amount of time between repeats (in 50Hz ticks). A parameter of KB_INQUIRE (-1) for either of these values leaves the value unchanged.

BINDING

```
move.w  rate,-(sp)
move.w  delay,-(sp)
move.w  #$23,-(sp)
trap    #14
addq.l  #6,sp
```

RETURN VALUE

Kbrate() returns a WORD with the low byte being the old value for *rate* and the high byte being the old value for *delay*.

---

**Keytbl()**

```c
KEYTAB *Keytbl( normal, shift, caps )
char *unshift, *shift, *caps;
```

Keytbl() reads/modifies the internal keyboard mapping tables.

OPCODE

16 (0x10)

AVAILABILITY

All TOS versions.

PARAMETERS

*normal* is a pointer to an array of 128 CHARs which can be indexed by a keyboard scancode to return the correct ASCII value for a given unshifted key. *shift* and *caps* point to similar array except their values are only utilized when SHIFT and CAPS-LOCK respectively are used. Passing a value of KT_NOCHANGE ((char *)-1) will leave the table unchanged.

BINDING

```
pea    caps
pea    shift
pea    normal
move.w #$10,-(sp)
trap   #14
lea    14(sp),sp
```

RETURN VALUE

Keytbl() returns a pointer to a KEYTAB structure defined as follows:

```c
typedef struct
{
    char *unshift;
    char *shift;
    char *caps;
} KEYTAB;
```

The entries in this table each point to the current keyboard lookup table in their category.
Entries are indexed with a keyboard scancode to obtain the ASCII value of a key. A value of 0 indicates that no ASCII equivalent exists.

**SEE ALSO** Bioskeys()

---

**Locksnd()**

LONG Locksnd( VOID )

Locksnd() prevents other applications from simultaneously attempting to use the sound system.

**OPCODE** 128 (0x80)

**AVAILABILITY** Available if the ‘_SND’ cookie has bit 2 set.

**BINDING**

```
move.w #$80,-(sp)
trap #14
addq.l #2,sp
```

**RETURN VALUE** Locksnd() returns 1 if the sound system was successfully locked or SNDLOCKED (-129) if the sound system was already locked.

**COMMENTS** This call should be used prior to any usage of the 16-bit DMA sound system.

**SEE ALSO** Unlocksnd()

---

**Logbase()**

VOIDP Logbase( VOID )

Logbase() returns a pointer to the base of the logical screen.

**OPCODE** 3 (0x03)

**AVAILABILITY** All TOS versions.

**BINDING**

```
move.w #$03,-(sp)
trap #14
addq.l #2,sp
```

**RETURN VALUE** Logbase() returns a pointer to the base of the logical screen.
The logical screen should not be confused with the physical screen. The logical screen is the memory area where the VDI does any drawing. The physical screen is the memory area where the video shifter gets its data from. Normally they are the same; however, keeping the addresses separate facilitates screen flipping.

**See Also**

Physbase()

---

### Metainit()

**VOID Metainit( metainfo )**

```c
METAINFO *metainfo;
```

Metainit() returns information regarding the current version and installed drives of MetaDOS.

**Opcode**

48 (0x30)

**Availability**

To test for the availability of MetaDOS the following steps must be taken:

1. Fill the METAINFO structure with all zeros.
2. Call Metainit().
3. If `metainfo.version` is NULL, MetaDOS is not installed.

**Parameters**

`metainfo` is a pointer to a METAINFO structure which is filled in by the call. METAINFO is defined as:

```c
typedef struct
{
    /* Bitmap of drives (Bit 0 = A, 1 = B, etc... */
    ULONG drivemap;

    /* String containing name and version */
    char *version;

    /* Currently unused */
    LONG reserved[2];
} METAINFO;
```

**Binding**

```assembly
pea metainfo
move.w #$30, -(sp)
trap #14
addq.l #6, sp
```

---

### Mfpint()

---

**The Atari Compendium**
VOID Mfpint( intno, vector )
WORD intno;
VOID (*vector)();

Mfpint() defines an interrupt handler for an MFP interrupt.

O P C O D E 13 (0x0D)

A V A I L A B I L I T Y All TOS versions.

P A R A M E T E R S intno is an index to a vector to replace with vector as follows:

<table>
<thead>
<tr>
<th>Name</th>
<th>intno</th>
<th>Vector</th>
</tr>
</thead>
<tbody>
<tr>
<td>MFP_PARALLEL</td>
<td>0</td>
<td>Parallel port</td>
</tr>
<tr>
<td>MFP_DCD</td>
<td>1</td>
<td>RS-232 Data Carrier Detect</td>
</tr>
<tr>
<td>MFP_CTS</td>
<td>2</td>
<td>RS-232 Clear To Send</td>
</tr>
<tr>
<td>MFP_BITBLT</td>
<td>3</td>
<td>BitBlt Complete</td>
</tr>
<tr>
<td>MFP_TIMERD or MFP_BAUDRATE</td>
<td>4</td>
<td>Timer D (RS-232 baud rate generator)</td>
</tr>
<tr>
<td>MFP_200HZ</td>
<td>5</td>
<td>Timer C (200Hz system clock)</td>
</tr>
<tr>
<td>MFP_ACIA</td>
<td>6</td>
<td>Keyboard/MIDI vector</td>
</tr>
<tr>
<td>MFP_DISK</td>
<td>7</td>
<td>Floppy/ Hard disk vector</td>
</tr>
<tr>
<td>MFP_TIMERB or MFP_HBLANK</td>
<td>8</td>
<td>Timer B (Horizontal blank)</td>
</tr>
<tr>
<td>MFP_TERR</td>
<td>9</td>
<td>RS-232 transmit error</td>
</tr>
<tr>
<td>MFP_TBE</td>
<td>10</td>
<td>RS-232 transmit buffer empty</td>
</tr>
<tr>
<td>MFP_RERR</td>
<td>11</td>
<td>RS-232 receive error</td>
</tr>
<tr>
<td>MFP_RBF</td>
<td>12</td>
<td>RS-232 receive buffer full.</td>
</tr>
<tr>
<td>MFP_TIMERA or MFP_DMASOUND</td>
<td>13</td>
<td>Timer A (DMA sound)</td>
</tr>
<tr>
<td>MFP_RING</td>
<td>14</td>
<td>RS-232 ring indicator</td>
</tr>
<tr>
<td>MFP_MONODETECT</td>
<td>15</td>
<td>Mono monitor detect/DMA sound complete</td>
</tr>
</tbody>
</table>

B I N D I N G pea vector
move.w intno,-(sp)
move.w #$0D,-(sp)
trap #14
addq.l #8,sp

C A V E A T S This call does not return the address of the old handler.

C O M M E N T S The only RS-232 vector that may be set on the Falcon030 with this function is the ring indicator.

N e w l y i n s t a l l e d i n t e r r u p t s m u s t b e e n a b l e d w i t h J e n a b i n t().

T H E A T A R I C O M P E N D I U M
SEE ALSO Jenabint(), Jdisint()

**Midiws()**

VOID Midiws(count, buf)

WORD count;
char *buf;

Midiws() outputs a data buffer to the MIDI port.

**OPCODE**

12 (0x0C)

**AVAILABILITY**

All TOS versions.

**PARAMETERS**

`count` + 1 characters are written from the buffer pointed to by `buf`.

**BINDING**

<table>
<thead>
<tr>
<th>Code</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>pea</td>
<td></td>
</tr>
<tr>
<td>move.w</td>
<td><code>count</code>-{(sp)}</td>
</tr>
<tr>
<td>move.w</td>
<td>#$0C-{(sp)}</td>
</tr>
<tr>
<td>trap</td>
<td>#14</td>
</tr>
<tr>
<td>addq.l</td>
<td>#8,sp</td>
</tr>
</tbody>
</table>

**NVMaccess()**

WORD NVMaccess(op, start, count, buffer)

WORD op, start, count;
char *buffer;

NVMaccess() reads/modifies data in non-volatile (battery backed-up) memory.

**OPCODE**

46 (0x2E)

**AVAILABILITY**

This function’s availability is variable. If it returns 0x2E (its opcode) when called, the function is non-existent and the operation was not carried out.

**PARAMETERS**

`op` indicates the operation to perform as follows:

<table>
<thead>
<tr>
<th>Name</th>
<th>op</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>NVM_READ</td>
<td>0</td>
<td>Read <code>count</code> bytes of data starting at offset <code>start</code> and place the data in <code>buffer</code>.</td>
</tr>
<tr>
<td>NVM_WRITE</td>
<td>1</td>
<td>Write <code>count</code> bytes of data from <code>buffer</code> starting at offset <code>start</code>.</td>
</tr>
<tr>
<td>NVM_RESET</td>
<td>2</td>
<td>Resets and clears all data in non-volatile memory.</td>
</tr>
</tbody>
</table>
**Offgibit()**

VOID Offgibit( mask )

WORD mask;

**Offgibit()** clears individual bits of the sound chip’s Port A.

**OPCODE**

29 (0x1D)

**AVAILABILITY**

All TOS versions.

**PARAMETERS**

mask is a bit mask arranged as shown below. For each of the lower eight bits in mask set to 0, that bit will be reset. Other bits (set as 1) will remain unchanged.

<table>
<thead>
<tr>
<th>Name</th>
<th>Mask</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>GI_FLOPPYSIDE</td>
<td>0x01</td>
<td>Floppy side select</td>
</tr>
<tr>
<td>GI_FLOPPYA</td>
<td>0x02</td>
<td>Floppy A select</td>
</tr>
<tr>
<td>GI_FLOPPYB</td>
<td>0x04</td>
<td>Floppy B select</td>
</tr>
<tr>
<td>GI_RTS</td>
<td>0x08</td>
<td>RS-232 Request To Send</td>
</tr>
<tr>
<td>GI_DTR</td>
<td>0x10</td>
<td>RS-232 Data Terminal Ready</td>
</tr>
<tr>
<td>GI_STROBE</td>
<td>0x20</td>
<td>Centronics strobe</td>
</tr>
<tr>
<td>GI_GPO</td>
<td>0x40</td>
<td>General purpose output (On a Falcon030, this bit controls the state of the internal speaker)</td>
</tr>
<tr>
<td>GI_SCCPORT</td>
<td>0x80</td>
<td>On a Mega STe or TT030, calling Ongibit(0x80) will cause SCC channel A to control the Serial 2 port rather than the LAN. To select the LAN, use Offgibit(0x7F).</td>
</tr>
</tbody>
</table>

**BINDING**

move.w mask,-(sp)

**RETURN VALUE**

NVMaccess() returns 0 if the operation succeeded or a negative error code otherwise.

**CAVEATS**

All of the locations are reserved for use by Atari and none are currently documented.

**COMMENTS**

Currently there is a total of 50 bytes in non-volatile RAM.
move.w #$1D,-(sp)
trap #14
addq.l #4,sp

**Ongibit()**

VOID Ongibit( mask )

WORD mask;

Ongibit() sets individual bits of the sound chip’s assigned Port A.

**Opcode**

30 (0x1E)

**Availability**

All TOS versions.

**Parameters**

mask is a bit mask arranged as defined in Offgibit(). For each of the lower eight bits in mask set to 1, that bit will be set. Other bits (set as 0) will remain unchanged.

**Binding**

move.w mask,-(sp)
move.w #$1E,-(sp)
trap #14
addq.l #4,sp

**See Also**

Giaccess(), Offgibit()

---

**Physbase()**

VOIDP Physbase( VOID )

Physbase() returns the address of the physical base of screen memory.

**Opcode**

2 (0x02)

**Availability**

All TOS versions.

**Binding**

move.w #$02,-(sp)
trap #14
addq.l #2,sp

**See Also**

Giaccess(), Offgibit()

**Return Value**

Physbase() returns the physical base address of the screen.

**Comments**

The physical base address is the memory area where the video shifter reads its
Protobt() – 4.85

data. The logical address is the memory area where the VDI draws. These are normally the same but are addressed individually to enable screen flipping.

SEE ALSO Logbase()

Protobt()

VOID Protobt(buf, serial, type, execflag )
VOIDP buf;
LONG serial;
WORD type, execflag;

Protobt() creates a prototype floppy boot sector in memory for writing to a floppy drive.

OPCODE 18 (0x12)

AVAILABILITY All TOS versions.

PARAMETERS buf is a 512 byte long buffer where the prototyped buffer will be written. If you are creating an executable boot sector, the memory buffer should contain the code you require. serial can be any of the following values:

<table>
<thead>
<tr>
<th>Name</th>
<th>serial</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>SERIAL_NOCHANGE</td>
<td>-1</td>
<td>Don’t change the serial number already in memory</td>
</tr>
<tr>
<td>SERIAL_RANDOM</td>
<td>&gt;0x01000000</td>
<td>Use a random number for the serial number</td>
</tr>
<tr>
<td>—</td>
<td>any other positive number</td>
<td>Set the serial number to serial.</td>
</tr>
</tbody>
</table>

execflag specifies the executable status of the boot sector as follows:

<table>
<thead>
<tr>
<th>Name</th>
<th>type</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>DISK_NOCHANGE</td>
<td>-1</td>
<td>Don’t change disk type.</td>
</tr>
<tr>
<td>DISK_SSSD</td>
<td>0</td>
<td>40 Track, Single-Sided (180K)</td>
</tr>
<tr>
<td>DISK_DSSD</td>
<td>1</td>
<td>40 Track, Double-Sided (360K)</td>
</tr>
<tr>
<td>DISK_SSDD</td>
<td>2</td>
<td>80 Track, Single-Sided (360K)</td>
</tr>
<tr>
<td>DISK_DSSDD</td>
<td>3</td>
<td>80 Track, Double-Sided (720K)</td>
</tr>
<tr>
<td>DISK_DSHD</td>
<td>4</td>
<td>High Density (1.44MB)</td>
</tr>
<tr>
<td>DISK_DSED</td>
<td>5</td>
<td>Extra-High Density (2.88MB)</td>
</tr>
<tr>
<td>Name</td>
<td>execflag</td>
<td>Meaning</td>
</tr>
<tr>
<td>-----------------</td>
<td>----------</td>
<td>----------------------------------</td>
</tr>
<tr>
<td>EXEC_NOCHANGE</td>
<td>-1</td>
<td>Don’t alter executable status</td>
</tr>
<tr>
<td>EXEC_NO</td>
<td>0</td>
<td>Disk is not executable</td>
</tr>
<tr>
<td>EXEC_YES</td>
<td>1</td>
<td>Disk is executable</td>
</tr>
</tbody>
</table>

**BINDING**

move.w      execflag,-(sp)
move.w      type,-(sp)
move.l      serial,-(sp)
pea         buf
move.w      #$12,-(sp)
trap        #14
lea         14(sp),sp

**CAVEATS**

type values of DISK_DSHD and DISK_DSED are only available when the high byte of the ‘_FDC’ cookie has a value of FLOPPY_DSHD (1) and FLOPPY_DSED (2) respectively.

**COMMENTS**

To create an MS-DOS compatible disk you must set the first three bytes of the prototyped boot sector to 0xE9, 0x00, and 0x4E.

**SEE ALSO**

Flopfmt(), Flopwr()

---

**Prtblk()**

WORD Prtblk( blk )
PRTBLK *blk;

Prtblk() accesses the built-in bitmap/text printing code.

**OPCODE**

36 (0x24)

**AVAILABILITY**

All TOS versions.

**PARAMETERS**

blk is a PRTBLK pointer containing information about the bitmap or text to print.
PRTBLK is defined as follows:

typedef struct {
    VOIDP blkptr;       /* pointer to screen scanline */
    UWWORD offset;      /* bit offset of first column */
    UWWORD width;       /* width of bitmap in bits */
    UWWORD height;      /* height of bitmap in scanlines */
    UWWORD left;        /* left print margin (in pixels) */
    UWWORD right;       /* right print margin (in pixels) */
    UWWORD srcres;      /* same as Getrez() */
    UWWORD destres;     /* 0 = draft, 1 = final */
    UWWORD *colpal;     /* color palette pointer */
}
* 0 = B/W Atari  
* 1 = Color Atari  
* 2 = Daisy Wheel  
* 3 = B/W Epson  
*/
UWORD type;  
/* 0 = parallel, 1 = serial */
UWORD port;  
/* halftone mask pointer or NULL to use default */
char *masks;
} PRTBLK;

BINDING
pea prtblk
move.w #$24,-(sp)
trap #14
addq.l #6,sp

CAVEATS
This call is extremely device dependent. v_bit_image() with GDOS installed should be used instead. Only ST compatible screen resolution bitmaps may be printed with this utility function.

COMMENTS
When printing text, blkptr should point to the text string, width should be the length of the text string, height should be 0, and masks should be NULL.

In graphic print mode, masks can be NULL to use the default halftone masks.

The system variable _prt_cnt (WORD *)0x4EE should be set to 1 to disable the ALT-HELP key before calling this function. It should be restored to a value of -1 when done.

SEE ALSO  Scrdump(), SetPrt()

Puntaes()

VOID Puntaes( VOID )

Puntaes() discards the AES (if memory-resident) and restarts the system.

OPCODE
39 (0x27)

AVAILABILITY  All TOS versions.

BINDING
move.w  #$27,-(sp)
trap  #14
addq.l  #2,sp

RETURN VALUE
If successful, this function will not return control to the caller.
Caveats  
Puntaes() is only valid with disk-loaded AES’s.

Comments  
Puntaes() discards the AES by freeing any memory it allocated, resetting the system variable `os_magic` (this variable should contain the magic number 0x87654321, however if reset, the AES will not initialize), and rebooting the system.

Random()  

LONG Random( VOID )

Random() returns a 24 bit random number.

Opcode  
17 (0x11)

Availability  
All TOS versions.

Binding  
move.w #$11,-(sp)
trap #14
addq.l #2,sp

Return Value  
Random() returns a 24-bit random value in the lower three bytes of the returned LONG.

Caveats  
The algorithm used provides an exact 50% occurrence of bit 0.

Rsconf()  

ULONG Rsconf( speed, flow, ucr, rsr, tsr, scr )
WORD speed, flow, ucr, rsr, tsr, scr;

Rsconf() reads/modifies the configuration of the serial device currently mapped to BIOS device #1 (GEMDOS ‘aux.’).

Opcode  
15 (0x0F)

Availability  
All TOS versions.

Parameters  
speed sets the serial device speed as follows:

<table>
<thead>
<tr>
<th>Name</th>
<th>speed</th>
<th>Baud Rate</th>
</tr>
</thead>
<tbody>
<tr>
<td>BAUD_19200</td>
<td>0</td>
<td>19200</td>
</tr>
<tr>
<td>BAUD_9600</td>
<td>1</td>
<td>9600</td>
</tr>
</tbody>
</table>
If `speed` is set to `BAUD_INQUIRE` (-2), the last baud rate set will be returned.

`flow` selects the flow control method as follows:

<table>
<thead>
<tr>
<th>Name</th>
<th>flow</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>FLOW_NONE</td>
<td>0</td>
<td>No flow control</td>
</tr>
<tr>
<td>FLOW_SOFT</td>
<td>1</td>
<td>XON/XOFF flow control (CTRL-S/CTRL-Q)</td>
</tr>
<tr>
<td>FLOW_HARD</td>
<td>2</td>
<td>RTS/CTS flow control (hardware)</td>
</tr>
<tr>
<td>FLOW_BOTH</td>
<td>3</td>
<td>Both methods of flow control</td>
</tr>
</tbody>
</table>

`ucr`, `rsr`, and `tsr` are each status bit arrays governing the serial devices. Each parameter uses only the lower eight bits of the `WORD`. They are defined as follows:

<table>
<thead>
<tr>
<th>Mask</th>
<th><code>ucr</code></th>
<th><code>rsr</code> and <code>tsr</code></th>
</tr>
</thead>
<tbody>
<tr>
<td>0x01</td>
<td>Unused</td>
<td>Receiver enable:</td>
</tr>
<tr>
<td></td>
<td></td>
<td><code>RS_RECVENABLE</code></td>
</tr>
<tr>
<td>0x02</td>
<td>Enable odd parity</td>
<td>Sync strip</td>
</tr>
<tr>
<td></td>
<td><code>RS_ODDPARITY</code> (0x02)</td>
<td><code>RS_SYNCSTRIP</code></td>
</tr>
<tr>
<td></td>
<td><code>RS_EVENPARITY</code> (0x00)</td>
<td></td>
</tr>
<tr>
<td>0x04</td>
<td>Parity enable</td>
<td>Match busy</td>
</tr>
<tr>
<td></td>
<td><code>RS_PARITYENABLE</code></td>
<td><code>RS_MATCHBUSY</code></td>
</tr>
<tr>
<td>0x08</td>
<td>Bits 3-4 of the <code>ucr</code></td>
<td>Break detect</td>
</tr>
<tr>
<td></td>
<td>collectively define the start and stop bit configuration as follows:</td>
<td><code>RS_BRKDETECT</code></td>
</tr>
<tr>
<td></td>
<td>00 = No Start or Stop bits</td>
<td></td>
</tr>
<tr>
<td></td>
<td><code>RS_NOSTOP</code> (0x00)</td>
<td></td>
</tr>
<tr>
<td></td>
<td>01 = 1 Start bit, 1 Stop bit</td>
<td></td>
</tr>
<tr>
<td></td>
<td><code>RS_1STOP</code> (0x08)</td>
<td></td>
</tr>
<tr>
<td></td>
<td>10 = 1 Start bit, 1½ Stop bits</td>
<td></td>
</tr>
<tr>
<td></td>
<td><code>RS_15STOP</code> (0x10)</td>
<td></td>
</tr>
<tr>
<td></td>
<td>11 = 1 Start bit, 2 Stop bits</td>
<td></td>
</tr>
<tr>
<td></td>
<td><code>RS_2STOP</code> (0x18)</td>
<td></td>
</tr>
<tr>
<td>0x10</td>
<td>See above.</td>
<td>Frame error</td>
</tr>
<tr>
<td></td>
<td></td>
<td><code>RS_FRAMEERR</code></td>
</tr>
<tr>
<td>0x20</td>
<td>Bits 5 and 6 together define the number of bits per word as follows:</td>
<td>Parity error <code>RS_PARITYERR</code></td>
</tr>
<tr>
<td></td>
<td>00 = 8 bits</td>
<td></td>
</tr>
<tr>
<td></td>
<td><code>RS_8BITS</code> (0x00)</td>
<td></td>
</tr>
<tr>
<td></td>
<td>01 = 7 bits</td>
<td></td>
</tr>
<tr>
<td></td>
<td><code>RS_7BITS</code> (0x20)</td>
<td></td>
</tr>
<tr>
<td>0x40</td>
<td>See above.</td>
<td>Overrun error</td>
</tr>
<tr>
<td>------</td>
<td>------------</td>
<td>---------------</td>
</tr>
<tr>
<td>0x80</td>
<td>CLK/16</td>
<td>Buffer full</td>
</tr>
</tbody>
</table>

**BINDING**

```
movew scr,-(sp)
movew tsr,-(sp)
movew rsr,-(sp)
movew ucr,-(sp)
movew flow,-(sp)
movew speed,-(sp)
movew $0F,-(sp)
trap #14
lea 14(sp),sp
```

**RETURN VALUE**

Rsconf() returns the last set baud rate if speed is set to **RS_LASTBAUD** (-2). Otherwise, it returns the old settings in a packed **LONG** with ucr being in the high byte, down to scr being in the low byte.

**COMMENTS**

Bits in the ucr, rsr, tsr, and scr should be set atomically. To correctly change a value, read the old value, mask it as appropriate and then write it back.

Baud rates higher than 19,200 bps available with SCC-based serial devices may be set by using the appropriate Fcntl() call under MiNT or by directly programming the SCC chip.

**CAVEATS**

The baud rate inquiry mode (speed = **RS_LASTBAUD**) does not work at all on TOS versions less than 1.04. TOS version 1.04 requires the patch program TOS14FX2.PRG (available from Atari Corp.) to allow this mode to function. All other TOS versions support the function normally.

**SEE ALSO**

Bconmap()
### Scrdmp()

**VOID Scrdmp( VOID )**

`Scrdmp()` starts the built-in hardware screen dump routine.

**OPCODE**

20 (0x14)

**AVAILABILITY**

All TOS versions.

**BINDING**

```
move.w #$14,-(sp)
trap #14
addq.l #2,sp
```

**CAVEATS**

`Scrdmp()` only dumps ST compatible screen resolutions.

**COMMENTS**

This routine is extremely device-dependent. You should use the **VDI** instead.

**SEE ALSO**

Prtblk(), v_hardcopy()

---

### Setbuffer()

**LONG Setbuffer( mode, begaddr, endaddr )**

**WORD mode;**  
**VOIDP begaddr;**  
**VOIDP endaddr;**

`Setbuffer()` sets the starting and ending addresses of the internal play and record buffers.

**OPCODE**

131 (0x83)

**AVAILABILITY**

Available when bit #2 of the ‘_SND’ cookie is set.

**PARAMETERS**

`mode` specifies which registers are to be set. A `mode` value of **PLAY** (0) sets the play registers, a value of **RECORD** (1) sets the record registers. `begaddr` specifies the starting location of the buffer. `endaddr` specifies the first invalid location for sound data past `begaddr`.

**BINDING**

```
pea endaddr
pea begaddr
move.w mode,-(sp)
move.w #$83,-(sp)
trap #14
lea 12(sp),sp
```
RETURN VALUE

Setbuffer() returns a 0 if successful or non-zero otherwise.

SEE ALSO

Buffoper()

Setcolor()

WORD Setcolor( idx, new )
WORD idx, new;

Setcolor() sets a ST/TT030 color register.

OPCODE

7 (0x07)

AVAILABILITY

All TOS versions.

PARAMETERS

idx specifies the color register to modify (0-16 on an ST, 0-255 on a STe or TT030). new is a bit array specifying the new color as follows:

<table>
<thead>
<tr>
<th>Bits 15-12</th>
<th>Bits 11-8</th>
<th>Bits 7-4</th>
<th>Bits 3-0</th>
</tr>
</thead>
<tbody>
<tr>
<td>Unused</td>
<td>Red</td>
<td>Green</td>
<td>Blue</td>
</tr>
</tbody>
</table>

Each color value has its bits packed in an unusual manner to stay compatible between machines. Bits are ordered 0, 3, 2, 1 with 0 being the least significant bit. If new is COL_INQUIRE (-1) then the old color is returned.

BINDING

move.w new, -(sp)
move.w idx, -(sp)
move.w #$06,-(sp)
trap #14
addq.l #6, sp

RETURN VALUE

Setcolor() returns the old value of the color register.

CAVEATS

This call is extremely device-dependent. vs_color() should be used instead.

COMMENTS

The top bit of each color nibble is unused on the original ST machines.

SEE ALSO

VsetRGB(), EsetColor(), Setpalette()
Setinterrupt()

LONG SetInterrupt( mode, cause )
WORD mode, cause;

SetInterrupt() defines the conditions under which an interrupt is generated by the sound system

**OPCODE**

135 (0x87)

**AVAILABILITY**

Available when bit #2 of the ‘_SND’ cookie is set.

**PARAMETERS**

*mode* configures interrupts to occur when the end of a buffer is reached. A value of **INT_TIMERA** (0) for *mode* sets Timer A, a value of **INT_I7** (1) sets the MFP i7 interrupt. *cause* defines the conditions for the interrupt as follows:

<table>
<thead>
<tr>
<th>Name</th>
<th>cause</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>INT_DISABLE</td>
<td>0</td>
<td>Disable interrupt</td>
</tr>
<tr>
<td>INT_PLAY</td>
<td>1</td>
<td>Interrupt at end of play buffer</td>
</tr>
<tr>
<td>INT_RECORD</td>
<td>2</td>
<td>Interrupt at end of record buffer</td>
</tr>
<tr>
<td>INT_BOTH</td>
<td>3</td>
<td>Interrupt at end of both buffers</td>
</tr>
</tbody>
</table>

**BINDING**

move.w cause,-(sp)
move.w mode,-(sp)
move.w #$87,-(sp)
trap #14
addq.l #6,sp

**RETURN VALUE**

SetInterrupt() returns 0 if no error occurred or non-zero otherwise.

**COMMENTS**

If either buffer is in repeat mode, these interrupts can be used to double-buffer sounds.

**SEE ALSO**

Buffoper()

---

Setmode()

LONG Setmode( mode )
WORD mode;

Setmode() sets the mode of operation for the play and record registers.

**OPCODE**

132 (0x84)
**Availability** Available if bit #2 of the `_SND` cookie is set.

**Parameters** *mode* defines the playback and record mode as follows:

<table>
<thead>
<tr>
<th>Name</th>
<th>mode</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>MODE_STEREO8</td>
<td>0</td>
<td>8-bit Stereo Mode</td>
</tr>
<tr>
<td>MODE_STEREO16</td>
<td>1</td>
<td>16-bit Stereo Mode</td>
</tr>
<tr>
<td>MODE_MONO</td>
<td>2</td>
<td>8-bit Mono Mode</td>
</tr>
</tbody>
</table>

**Binding**

```
move.w      mode, -(sp)
move.w      #$84, sp
trap        #14
addq.l      #4, sp
```

**Return Value** Setmode() returns 0 if the operation was successful or non-zero otherwise.

**Caveats** Recording only works in 16-bit stereo mode.

**See Also** Buffoper()

---

**Setmontracks()**

LONG Setmontracks( track )

**Parameters** *track* specifies the playback track to monitor (0-3).

**Binding**

```
move.w      track, -(sp)
move.w      #$86, -(sp)
trap        #14
addq.l      #4, sp
```

**Return Value** Setmontracks() returns a 0 if the operation was successful or non-zero otherwise.
Setpalette()

VOID Setpalette( palette )
WORD *palette;

Setpalette() loads the ST color lookup table with a new palette.

OPCODE 6 (0x06)

AVAILABILITY All TOS versions.

PARAMETERS palette is a pointer to a WORD array containing 16 color encoded WORDs as defined in Setcolor().

BINDING
pea palette
move.w #$06,-(sp)
trap #14
addlq #6,sp

COMMENTS The actual palette data is not copied from the specified array until the next vertical blank interrupt. For this reason, this call should be followed by Vsync() to be sure the array memory is not modified or reallocated prior to the transfer.

SEE ALSO Setcolor(), EsetPalette(), VsetRGB(), vs_color()

Setprt()

WORD Setprt( new )
WORD new;

Setprt() sets the OS’s current printer configuration bits.

OPCODE 33 (0x21)

AVAILABILITY All TOS versions.

PARAMETERS new is a WORD bit array defined as follows:

<table>
<thead>
<tr>
<th>Mask</th>
<th>When clear</th>
<th>When Set</th>
</tr>
</thead>
<tbody>
<tr>
<td>0x01</td>
<td>Dot Matrix</td>
<td>Daisy Wheel</td>
</tr>
<tr>
<td></td>
<td>PRT_DOTMATRIX</td>
<td>PRT_DAISY</td>
</tr>
<tr>
<td>0x02</td>
<td>Monochrome</td>
<td>Color</td>
</tr>
<tr>
<td></td>
<td>PRT_MONO</td>
<td>PRT_COLOR</td>
</tr>
<tr>
<td>0x04</td>
<td>Atari Printer</td>
<td>Epson Printer</td>
</tr>
</tbody>
</table>

THE ATARI COMPENDIUM
If `new` is set to `PRT_INQUIRE (-1) Setprt()` will return the current configuration without modifying the current setup.

**BINDING**

```
move.w new, -(sp)
move.w #$33, -(sp)
trap #14
addq.l #4, sp
```

**RETURN VALUE**

`Setprt()` returns the prior configuration.

**CAVEATS**

This call only affects the internal screen dump code which only operates on ST compatible resolutions.

**SEE ALSO**

`Prtblk()`, `Scrdmp()`, `v_hardcopy()`

---

### Setscreen()

VOID Setscreen( `log`, `phys`, `mode` )

VOIDP `log`, `phys`;

WORD `mode`;

Setscreen() changes the base addresses and mode of the current screen.

**OPCODE**

5 (0x05)

**AVAILABILITY**

All TOS versions.

**PARAMETERS**

`log` is the address for the new logical screen base. `phys` is the new address for the physical screen base. `mode` defines the screen mode to switch to (same as `Getrez()`). If any of these three parameters is set to `SCR_NOCHANGE (-1)` then that value will be left unchanged.

**BINDING**

```
move.w mode, -(sp)
pea phys
pea log
move.w #$5, -(sp)
trap #14
lea 12(sp), sp
```
CAVEATS

Changing screen modes with this call does not reinitialize the AES. The VDI and VT52 emulator are, however, correctly reinitialized. The AES should not be used after changing screen mode with this call until the old screen mode is restored.

COMMENTS

The Atari ST and Mega ST required that its physical screen memory be on a 256 byte boundary. All other Atari computers only require a WORD boundary.

To access the unique video modes of the Falcon030 the call VsetScreen() (which is actually an alternate binding of this call with the same opcode) should be used in place of this call.

SEE ALSO

VsetMode(), VsetScreen(), EsetShift()

Settime()

VOID Settime( time )

LONG time;

Settime() sets a new IKBD date and time.

OPCODE

22 (0x16)

AVAILABILITY

All TOS versions.

PARAMETERS

*time* is a LONG bit array defined as follows:

<table>
<thead>
<tr>
<th>Bits</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>0-4</td>
<td>Seconds / 2 (0-29)</td>
</tr>
<tr>
<td>5-10</td>
<td>Minute (0-59)</td>
</tr>
<tr>
<td>11-15</td>
<td>Hour (0-23)</td>
</tr>
<tr>
<td>16-20</td>
<td>Day (1-31)</td>
</tr>
<tr>
<td>21-24</td>
<td>Month (1-12)</td>
</tr>
<tr>
<td>25-31</td>
<td>Year - 1980 (0-127)</td>
</tr>
</tbody>
</table>

The value can be represented in a C structure as follows:

```c
typedef struct
{
  unsigned year:7;
  unsigned month:4;
  unsigned day:5;
  unsigned hour:5;
  unsigned minute:6;
  unsigned second:5;
} ...
```
) BIOS_TIME;

BINDING
move.l  time,-(sp)
movw  #16,-(sp)
trap  #14
addq.l  #6,sp

COMMENTS
As of TOS 1.02, this function also updates the GEMDOS time.

SEE ALSO
Gettime(), Tsettime(), Tsetdate()

Settracks()

LONG Settracks( playtracks, retracks )
WORD playtracks, retracks;

Settracks() sets the number of recording and playback tracks.

OPCONE
133 (0x85)

AVAILABILITY
Available only when bit #2 of the '_SND' cookie is set.

PARAMETERS
playtracks specifies the number of playback tracks (0-3) and retracks specifies
the number of recording tracks.

BINDING
move.w  retracks,-(sp)
movw  playtracks,-(sp)
movw  #85,-(sp)
trap  #14
addq.l  #6,sp

RETURN VALUE
Settracks() returns 0 if the operation was successful or non-zero otherwise.

COMMENTS
The tracks specified are stereo tracks. When in 8-bit Mono mode, two samples are
read at a time.

SEE ALSO
Setmode(), Setmontracks()
**Sndstatus()**

LONG Sndstatus( reset )

WORD reset;

*Sndstatus()* can be used to test the error condition of the sound system and to completely reset it.

**OPCODE**

140 (0x8C)

**AVAILABILITY**

Available only when bit #2 of the ‘_SND’ cookie is set.

**PARAMETERS**

*reset* is a flag indicating whether the sound system should be reset. A value of **SND_RESET** (1) will reset the sound system.

**BINDING**

```
move.w         reset,-(sp)
move.w #$8C,-(sp)
trap           #14
addq.l        #4,sp
```

**RETURN VALUE**

*Sndstatus()* returns a **LONG** bit array indicating the current error status of the sound system defined as follows:

<table>
<thead>
<tr>
<th>Bit(s)</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>0-3</td>
<td>These bits form a value indicating the error condition of the sound system as follows:</td>
</tr>
<tr>
<td>Name</td>
<td>Mask</td>
</tr>
<tr>
<td>SND_ERROR</td>
<td>0xF</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Name</th>
<th>Value</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>SND_OK</td>
<td>0</td>
<td>No Error</td>
</tr>
<tr>
<td>SND_BADCONTROL</td>
<td>1</td>
<td>Invalid Control Field</td>
</tr>
<tr>
<td>SND_BADSYS</td>
<td>2</td>
<td>Invalid Sync Format</td>
</tr>
<tr>
<td>SND_BADCLOCK</td>
<td>3</td>
<td>Clock out of range</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Bit(s)</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>4</td>
<td>If this bit is set, left channel clipping has occurred. Use the mask <strong>SND_LEFTCLIP</strong> (0x10) to isolate this bit.</td>
</tr>
<tr>
<td>5</td>
<td>If this bit is set, right channel clipping has occurred. Use the mask <strong>SND_RIGHTCLIP</strong> (0x20) to isolate this bit.</td>
</tr>
<tr>
<td>6-31</td>
<td>Unused.</td>
</tr>
</tbody>
</table>

**COMMENTS**

On reset, the following things happen:

- DSP is tristated
- Gain and attentuation are zeroed
- Old matrix connections are reset
- ADDERIN is disabled
• Mode is set to 8-Bit Stereo
• Play and record tracks are set to 0
• Monitor track is set to 0
• Interrupts are disabled
• Buffer operation is disabled

**Soundcmd()**

LONG Soundcmd( mode, data )
WORD mode, data;

*Soundcmd()* sets various configuration parameters in the sound system.

**OPCODE**

130 (0x82)

**AVAILABILITY**

Available only when bit #2 of ‘_SND’ cookie is set.

**PARAMETERS**

*mode* specifies how *data* is interpreted as follows:

<table>
<thead>
<tr>
<th>Name</th>
<th>mode</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>LTATTEN</td>
<td>0</td>
<td>Set the left attenuation (increasing attenuation is the same as</td>
</tr>
<tr>
<td></td>
<td></td>
<td>decreasing volume). <em>data</em> is a bit mask as follows:</td>
</tr>
<tr>
<td></td>
<td></td>
<td>XXXX XXXX LLLL XXXX</td>
</tr>
<tr>
<td></td>
<td></td>
<td>‘L’ specifies a valid value between 0 and 15 used to set the</td>
</tr>
<tr>
<td></td>
<td></td>
<td>attenuation of the left channel in -1.5db increments. The bits</td>
</tr>
<tr>
<td></td>
<td></td>
<td>represented by ‘X’ are reserved and should be 0.</td>
</tr>
<tr>
<td>RATTEN</td>
<td>1</td>
<td>Set the right attenuation. <em>data</em> is a bit mask as follows:</td>
</tr>
<tr>
<td></td>
<td></td>
<td>XXXX XXXX RRRR XXXX</td>
</tr>
<tr>
<td></td>
<td></td>
<td>‘R’ specifies a valid value between 0 and 15 used to set the</td>
</tr>
<tr>
<td></td>
<td></td>
<td>attenuation of the right channel in -1.5db increments. The bits</td>
</tr>
<tr>
<td></td>
<td></td>
<td>represented by ‘X’ are reserved and should be 0.</td>
</tr>
<tr>
<td>LTGAIN</td>
<td>2</td>
<td>Set the left channel gain (boost the input to the ADC). <em>data</em> is a</td>
</tr>
<tr>
<td></td>
<td></td>
<td>bit mask as follows:</td>
</tr>
<tr>
<td></td>
<td></td>
<td>XXXX XXXX LLLL XXXX</td>
</tr>
<tr>
<td></td>
<td></td>
<td>‘L’ specifies a valid value between 0 and 15 used to set the</td>
</tr>
<tr>
<td></td>
<td></td>
<td>gain of the left channel in 1.5db increments. The bits</td>
</tr>
<tr>
<td></td>
<td></td>
<td>represented by ‘X’ are reserved and should be 0.</td>
</tr>
<tr>
<td>Command</td>
<td>Value</td>
<td>Description</td>
</tr>
<tr>
<td>------------</td>
<td>-------</td>
<td>-------------</td>
</tr>
</tbody>
</table>
| RTGAIN     | 3     | Set the right channel gain (boost the input to the ADC). *data* is a bit mask as follows: XXXX XXXX RRRR XXXX
|            |       | ‘R’ specifies a valid value between 0 and 15 used to set the gain of the right channel in 1.5Db increments. The bits represented by ‘X’ are reserved and should be 0. |
| ADDERIN    | 4     | Set the 16 bit ADDER to receive its input from the source(s) specified in *data*. *data* is a bit mask where each bit indicates a possible source. Bit 0 represents the ADC (*ADDR_ADC*). Bit 1 represents the connection matrix (*ADDR_MATRIX*). Setting either or both of these bits determines the source of the ADDER. |
| ADCINPUT   | 5     | Set the inputs of the left and right channels of the ADC. *data* is a bit mask with bit 0 being the right channel: LEFT_MIC (0x00) or LEFT_PSG (0x02) and bit 1 being the left channel: RIGHT_MIC (0x00) or RIGHT_PSG (0x01). Setting a bit causes that channel to receive its input from the Yamaha PSG. Clearing a bit causes that channel to receive its input from the microphone. |
| SETPRESCALE| 6     | This mode is only valid when *Devconnect()* is used to set the prescaler to TT030 compatibility mode. In that case, *data* represents the TT030 compatible prescale value as follows: |

<table>
<thead>
<tr>
<th>Name</th>
<th>Value</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>CCLK_6K</td>
<td>0</td>
<td>Divide by 1280 (6.25 MHz)</td>
</tr>
<tr>
<td>CCLK_12K</td>
<td>1</td>
<td>Divide by 640 (12.5 Mhz)</td>
</tr>
<tr>
<td>CCLK_25K</td>
<td>2</td>
<td>Divide by 320 (25 MHz)</td>
</tr>
<tr>
<td>CCLK_50K</td>
<td>3</td>
<td>Divide by 160 (50 MHz)</td>
</tr>
</tbody>
</table>

Setting *data* to SND_INQUIRE (-1) with any command will cause that command’s current value to be returned and the parameter unchanged.

```
BINDING
move.w   data,-(sp)
move.w   mode,-(sp)
move.w   #$82,-(sp)
trap     #14
addq.l   #2,sp
```

**RETURN VALUE**

`Soundcmd()` returns the prior value of the specified command if *data* is SND_INQUIRE (-1).

Using the SETPRESCALE mode to set a frequency of 6.25 MHz (CCLK_6K) will cause the sound system to mute on a Falcon030 as it does not support this sample rate.

**CAVEATS**

On current systems, a bug exists that causes a *mode* value of LTGAIN to set the gain for both channels.

**SEE ALSO**

`Devconnect()`
**Ssbrk()**

VOIDP Ssbrk( len )  
WORD len;

*Ssbrk(*) is designed to reserve memory at the top of RAM prior to the initialization of GEMDOS.

**OPCODE**  
1 (0x01)

**AVAILABILITY**  
All TOS versions.

**PARAMETERS**  
*len* is a WORD value specifying the number of bytes to reserve at the top of RAM.

**BINDING**  
move.w len,-(sp)  
move.w #$01,-(sp)  
trap #14  
addq.l #4,sp

**RETURN VALUE**  
*Ssbrk(*) returns a pointer to the allocated block.

**CAVEATS**  
*Ssbrk(*) was only used on early development systems. Currently the function is unimplemented and does not do anything.

---

**Supexec()**

LONG Supexec( func )  
LONG (*func)( VOID );

*Supexec(*) executes a user-defined function in supervisor mode.

**OPCODE**  
38 (0x26)

**AVAILABILITY**  
All TOS versions.

**PARAMETERS**  
*func* is the address to a function which will be called in supervisor mode.

**BINDING**  
pea func  
move.w #$26,-(sp)  
trap #14  
addq.l #6,sp
Unlocksnd() - 4.103

**RETURN VALUE**

*Supexec()* returns the **LONG** value returned by the user function.

**CAVEATS**

Care must be taken when calling the operating system in supervisor mode. The **AES** must not be called while in supervisor mode.

**SEE ALSO**

*Super()*

---

**Unlocksnd()**

**LONG Unlocksnd( VOID )**

*Unlocksnd()* unlocks the sound system so that other applications may utilize it.

**OPCODE**

129 (0x81)

**AVAILABILITY**

All TOS versions.

**BINDING**

```
move.w #$81,-(sp)
trap #14
addq.l #2,sp
```

**RETURN VALUE**

*Unlocksnd()* returns a 0 if the sound system was successfully unlocked or **SNDNOTLOCK** (-128) if the sound system wasn’t locked prior to the call.

**SEE ALSO**

*Locksnd()*

---

**VgetMonitor()**

**WORD VgetMonitor( VOID )**

*VgetMonitor()* returns a value which determines the kind of monitor currently being used.

**OPCODE**

89 (0x59)

**AVAILABILITY**

Available if the ‘_VDO’ cookie has a value of 0x00030000 or greater.

**BINDING**

```
move.w #$59,-(sp)
trap #14
addq.l #2,sp
```

**RETURN VALUE**

*VgetMonitor()* returns a value describing the monitor currently connected to the system as follows:
VgetRGB()

VOID VgetRGB(index, count, rgb)

WORD index, count;
RGB *rgb;

VgetRGB() returns palette information as 24-bit RGB data.

OPCODE 94 (0x5E)

AVAILABILITY Available if the ‘_VDO’ cookie has a value of 0x00030000 or greater.

PARAMETERS index specifies the beginning color index in the palette to read data from. count specifies the number of palette entries to read. rgb is a pointer to an array of RGBs which will be filled in by the functions. RGB is defined as:

```c
typedef struct
{
    BYTE reserved;
    BYTE red;
    BYTE green;
    BYTE blue;
} RGB;
```

BINDING pea rgb
move.w count,-(sp)
move.w index,-(sp)
move.w #$5E,-(sp)
trap #14
lea 10(sp),sp

COMMENTS VgetRGB() is device-dependent in nature and it is therefore recommended that vq_color() be used instead.

SEE ALSO VsetRGB()
**VgetSize()**

LONG VgetSize( mode )

WORD mode;

VgetSize() returns the size of a screen mode in bytes.

**Opcode**
91 (0x5B)

**Availability**
Available if the ‘_VDO’ cookie has a value of 0x00030000 or greater.

**Parameters**
mode is a modecode as defined in VsetMode().

**Binding**
move.w mode,-(sp)
move.w #$5B,-(sp)
trap #14
addq.l #4,sp

**Return Value**
VgetSize() returns the size in bytes of a screen mode of type mode.

**VsetMask()**

VOID VsetMask( ormask, andmask, overlay )

LONG ormask, andmask;

WORD overlay;

VsetMask() provides access to ‘overlay’ mode.

**Opcode**
146 (0x92)

**Availability**
Available if the ‘_VDO’ cookie has a value of 0x00030000 or greater.

**Parameters**
When the VDI processes a vs_color() call. It converts the desired color into a hardware palette register. In 16-bit true-color mode, this is a WORD formatted as follows:

RRRR RGGG GGXB BBBB

The ‘X’ is the system overlay bit. In 24-bit true color a LONG is formatted as follows:

XXXXXXXX RRRRRRRR GGGG GGGGGG BBBBBBBB

VsetMask() sets a logical OR and AND mask which are applied to this register.

**The Atari Compendium**
before being stored. The default system value for \textit{ormask} is 0x00000000 and the default value for \textit{andmask} is 0xFFFFFFFF.

\textit{overlay} should be \texttt{OVERLAY\_ON} (1) to enable overlay mode or \texttt{OVERLAY\_OFF} (0) to disable it.

\begin{verbatim}
move.w #overlay,-(sp)
move.l #andmask,-(sp)
move.l #ormask,-(sp)
move.w #$92,-(sp)
trap #14
add.l #12,sp
\end{verbatim}

\textbf{COMMENTS} To make colors defined by the VDI transparent in 16-bit true color with overlay mode enabled, use an \textit{andmask} value of 0xFFFFFFFFDF and an \textit{ormask} value of 0x00000000. To make colors visible, use an \textit{andmask} of 0x00000000 and an \textit{ormask} of 0x00000020.

\textbf{VsetMode()}

\begin{verbatim}
WORD VsetMode( mode )
WORD mode;
\end{verbatim}

\texttt{VsetMode()} places the video shifter into a specific video mode.

\textbf{OPCODE} 88 (0x58)

\textbf{AVAILABILITY} Available if the ‘_VDO’ cookie has a value of 0x00030000 or greater.

\textbf{PARAMETERS} \textit{mode} is a \texttt{WORD} bit array arranged as follows:

\begin{tabular}{|l|c|p{12cm}|}
\hline
Name & Bit(s) & Meaning \\
\hline
BPS1 (0x00) & 0-2 & These bits form a value so that $2^X$ represents the number of bits per pixel. \\
BPS2 (0x01) & & \\
BPS4 (0x02) & & \\
BPS8 (0x03) & & \\
BPS16 (0x04) & & \\
COL80 (0x08) & 3 & 80 Column Flag (if set, 80 columns, otherwise 40) \\
COL40 (0x00) & & \\
VGA (0x10) & 4 & VGA Flag (if set, VGA mode will be used, otherwise television/monitor mode) \\
TV (0x00) & & \\
PAL (0x20) & 5 & PAL Flag (if set, PAL will be used, otherwise NTSC) \\
NTSC (0x00) & & \\
OVERSCAN (0x40) & 6 & Overscan Flag (not valid with VGA) \\
STMODES (0x80) & 7 & ST Compatibility Flag \\
VERTFLAG (0x100) & 8 & Vertical Flag (is set, enables interlace mode on a color monitor or double-line mode on a VGA monitor) \\
\hline
\end{tabular}
If `mode` is `VM_INQUIRE` (-1) then the current mode code is returned without changing the current settings.

### Binding

```assembly
move.w mode, -(sp)
move.w #$58, sp
trap #14
addq.l #4, sp
```

### Return Value

`VsetMode()` returns the prior video mode.

### Caveats

`VsetMode()` does not reset the video base address, reserve memory, or reinitialize the VDI. To do this, use `VsetScreen()`.

### Comments

Some video modes are not legal. 40 column monoplane modes and 80 column VGA true color modes are not supported.

### See Also

`VsetScreen()`, `Setscreen()`

---

## VsetRGB()

```c
VOID VsetRGB(index, count, rgb)
WORD index, count;
RGB *rgb;

VsetRGB() sets palette registers using 24-bit RGB values.
```

### Opcode

93 (0x5D)

### Availability

Available if the ‘_VDO’ cookie has a value of 0x00030000 or greater.

### Parameters

`index` specifies the first palette index to modify. `count` specifies the number of palette entries to modify. `rgb` is a pointer to an array of RGB elements which will be copied into the palette.

### Binding

```assembly
pea rgb
move.w count, -(sp)
move.w index, -(sp)
move.w #$5D, -(sp)
trap #14
lea 10(sp), sp
```

### Comments

This call is device-dependent by nature. It is therefore recommended that `vs_color()` be used instead.
VsetScreen()

VOID VsetScreen( log, phys, mode, modecode )
VOIDP log, phys;
WORD mode, modecode;

VsetScreen() changes the base addresses and mode of the current screen.

OPCODE
5 (0x05)

AVAILABILITY
All TOS versions. The ability of this call to utilize the modecode parameter and the memory allocation feature is limited to systems having a '_VDO' cookie with a value of 0x00030000 or greater.

PARAMETERS
log is the address for the new logical screen base. phys is the new address for the physical screen base. If either log or phys is NULL, the XBIOS will allocate a new block of memory large enough for the current screen and reset the parameter accordingly.

mode defines the screen mode to switch to (same as Getrez()). Setting mode to SCR_MODECODE (3) will cause modecode to be used to set the graphic mode (see VsetMode() for valid values for this parameter), otherwise modecode is ignored. If any of these three parameters is set to SCR_NOCHANGE (-1) then that value will be left unchanged.

BINDING
move.w modecode,-(sp)
movw.mode,-(sp)
pea phys
pea log
move.w #$05,-(sp)
trap #14
lea 14(sp),sp

CAVEATS
Changing screen modes with this call does not reinitialize the AES. The VDI and VT52 emulator are, however, correctly reinitialized. The AES should not be used after changing screen mode with this call until the old screen mode is restored.

COMMENTS
TOS 1.00 and 1.02 required that its physical screen memory be on a 256 byte boundary. All other Atari computers only require a WORD boundary.

This call is actually a revised binding of Setscreen() developed to allow access to the newly available modecode parameter.

SEE ALSO
Setscreen(), VsetMode()
**VsetSync()**

VOID VsetSync( external )
WORD external;

VsetSync() sets the external video sync mode.

**OPCODE**

90 (0x5A)

**AVAILABILITY**

Available if the ‘_VDO’ cookie has a value of 0x00030000 or greater.

**PARAMETERS**

*external* is a WORD bit array defined as follows:

<table>
<thead>
<tr>
<th>Name</th>
<th>Bit</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>VCLK_EXTERNAL</td>
<td>0</td>
<td>Use external clock.</td>
</tr>
<tr>
<td>VCLK_EXTVSYNC</td>
<td>1</td>
<td>Use external vertical sync.</td>
</tr>
<tr>
<td>VCLK_EXTHSYNC</td>
<td>2</td>
<td>Use external horizontal sync.</td>
</tr>
<tr>
<td>–</td>
<td>3-15</td>
<td>Reserved (set to 0)</td>
</tr>
</tbody>
</table>

**BINDING**

move.w external,-(sp)
move.w #$5A,-(sp)
trap #14
addq.l #2,sp

**CAVEATS**

This call only works in Falcon video modes, not in compatibility or any four color modes.

**Vsync()**

VOID Vsync( VOID )

Vsync() pauses program execution until the next vertical blank interrupt.

**OPCODE**

37 (0x25)

**AVAILABILITY**

All TOS versions.

**BINDING**

move.w #$25,-(sp)
trap #14
addq.l #2,sp
WavePlay()

WORD WavePlay(flags, rate, sptr, slen )

WORD flags;
LONG rate;
VOIDP sptr;
LONG slen;

WavePlay() provides a easy method for applications to utilize the DMA sound system on the STe, TT030, and Falcon030 and playback user-defined event sound effects.

OPCODE

165 (0xA5)

AVAILABILITY

Available only when the ‘SAM\0’ cookie exists.

PARAMETERS

flags is a bit mask consisting of the following options:

<table>
<thead>
<tr>
<th>Name</th>
<th>Mask</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>WP_MONO</td>
<td>0x00</td>
<td>The sound to be played back is monophonic.</td>
</tr>
<tr>
<td>WP_STEREO</td>
<td>0x01</td>
<td>The sound to be played back is in stereo.</td>
</tr>
<tr>
<td>WP_8BIT</td>
<td>0x00</td>
<td>The sound to be played back was sampled at 8-bit resolution.</td>
</tr>
<tr>
<td>WP_16BIT</td>
<td>0x02</td>
<td>The sound to be played back was sampled at 16-bit resolution.</td>
</tr>
<tr>
<td>WP_MACRO</td>
<td>0x100</td>
<td>Play back a user-assigned macro or application global sound effect. This flag is exclusive and modifies the meaning of the other parameters to this call as shown below.</td>
</tr>
</tbody>
</table>

rate specifies the sample rate in Hertz (for example 49170L to play back at 49170 Hz). If WP_MACRO was specified in flags, then this parameter is ignored and should be set to 0L.

sptr is a pointer to the sound sample in memory. If WP_MACRO was specified in flags then this parameter should be a LONG containing either the application cookie specified in the .SAA file or the ‘SAM\0’ cookie to play an application global.

slen is the length of the sample in bytes. If WP_MACRO was specified in flags then slen is the macro or application global index as specified in the .SAA file. Valid application global values are as follows:

<table>
<thead>
<tr>
<th>Name</th>
<th>slen</th>
<th>Usage</th>
</tr>
</thead>
</table>

T h e  A t a r i  C o m p e n d i u m
### WavePlay()

**AG_FIND**  0  Call WavePlay() with this value when the user requests display of the 'Find' dialog box.

**AG_REPLACE**  1  Call WavePlay() with this value when the user requests display of the 'Replace' dialog box.

**AG_CUT**  2  Call WavePlay() with this value when the user requests a 'Cut' operation.

**AG_COPY**  3  Call WavePlay() with this value when the user requests a 'Copy' operation.

**AG_PASTE**  4  Call WavePlay() with this value when the user requests a 'Paste' operation.

**AG_DELETE**  5  Call WavePlay() with this value when the user requests a 'Delete' operation. This should not be called when the user presses the 'Delete' key.

**AG_HELP**  6  Call WavePlay() with this value when the user requests display of application ‘Help.’ This should not be called when the user presses the ‘Help’ key.

**AG_PRINT**  7  Call WavePlay() with this value when the user requests display of the ‘Print’ dialog box.

**AG_SAVE**  8  Call WavePlay() with this value when the user requests that the current document be saved. This should not be used for any operation that calls the file selector.

**AG_ERROR**  9  Call WavePlay() with this value when the application encounters an error not presented to the user in an alert or error dialog (error dialogs may be assigned sounds).

**AG_QUIT**  10  Call WavePlay() with this value when the user requests that the application exit. Use this global after the user has confirmed a quit with any dialog box that may have been necessary.

#### Binding

- `move.l` `slen,-(sp)`
- `pea` `sptr`
- `move.l` `rate,-(sp)`
- `move.w` `flags,-(sp)`
- `move.w` `#$A5,-(sp)`
- `trap` `#14`
- `lea` `16(sp),sp`

#### Return Value

WavePlay() returns **WP_OK** (0) if the call was successful, **WP_ERROR** (-1) if an error occurred, or **WP_NOSOUND** (1) to indicate that no sound was played (either because the user had not previously assigned a sound to the given macro or SAM was disabled).

#### Caveats

This function is only available when the System Audio Manager TSR (available from Atari Corp. or SDS) is installed. Extended development information is available online the Atari Developer’s roundtable on GEnie.

Because of previously misdocumented sample rates, the value for `rate` must be 33880 to play back a sample at 32880 Hz, 20770 to play back a sample at 19668 Hz, and 16490 to play back a sample at 16390 Hz.
Even if an application does not install any custom events in a .SAA file, an application must still provide a .SAA file if it wishes to use application globals so that the SAM configuration accessory allows the user to assign those sounds.

A macro is commonly used to access the application global sounds available as follows:

```c
#define WavePlayMacro(a) WavePlay( WP_MACRO, 0L, SAM_COOKIE, a);
```

### Xbtimer()

```c
VOID Xbtimer( timer, control, data, hand )
WORD timer, control, data;
VOID (*hand)( VOID );
```

**Xbtimer()** sets an interrupt on the 68901 chip.

**OPCODE**

31 (0x1F)

**AVAILABILITY**

All TOS versions.

**PARAMETERS**

- **timer** is a value defining which timer to set as follows:

<table>
<thead>
<tr>
<th>Name</th>
<th>Timer</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>XB_TIMERA</td>
<td>0</td>
<td>Timer A (DMA sound counter)</td>
</tr>
<tr>
<td>XB_TIMERB</td>
<td>1</td>
<td>Timer B (Hblank counter)</td>
</tr>
<tr>
<td>XB_TIMERC</td>
<td>2</td>
<td>Timer C (200Hz system clock)</td>
</tr>
<tr>
<td>XB_TIMERD</td>
<td>3</td>
<td>Timer D (RS-232 baud rate generator)</td>
</tr>
</tbody>
</table>

- **control** is placed into the control register of the timer.
- **data** is placed in the data register of the timer.
- **hand** is a pointer to the interrupt handler which is called by the interrupt.

**BINDING**

```assembly
pea hand
move.w data,-(sp)
move.w control,-(sp)
move.w timer,-(sp)
move.w #$1F,-(sp)
trap #14
lea 12(sp),sp
```

**SEE ALSO**

Mfpint(), Jenabint(), Jdisint()
Overview

This chapter will cover those aspects of Atari software programming that can only be accomplished by accessing hardware registers directly. In most cases, Atari has provided OS calls to manipulate the hardware. When an OS call exists to access hardware, it should *always* be used to ensure upward and backward compatibility. Keep in mind that access to hardware registers is limited to those applications operating in supervisor mode only (except where noted otherwise).

Besides those hardware registers discussed here, a complete list of I/O registers, system variables, and interrupt vectors are contained in *Appendix B: Memory Map*.

The 680x0 Processor

Atari computers use the Motorola MC68000 or MC68030. Third party devices have also been created to allow the use of a MC68010, MC68020, or MC68040 processor. The system cookie ‘_CPU’ should be used to determine the currently installed processor. The following table lists the 680x0’s interrupt priority assignments:

<table>
<thead>
<tr>
<th>Level</th>
<th>Assignment</th>
</tr>
</thead>
<tbody>
<tr>
<td>7</td>
<td>NMI</td>
</tr>
<tr>
<td>6</td>
<td>MK68901 MFP</td>
</tr>
<tr>
<td>5</td>
<td>SCC(^1)</td>
</tr>
<tr>
<td>4</td>
<td>VBLANK (Sync)</td>
</tr>
<tr>
<td>3</td>
<td>VME Interrupter(^2)</td>
</tr>
<tr>
<td>2</td>
<td>HBLANK (Sync)</td>
</tr>
<tr>
<td>1</td>
<td>Unused</td>
</tr>
</tbody>
</table>

Interrupts may be disabled by setting the system interrupt mask (bits 8-10 of the SR register) to a value higher than the interrupts you wish to disable. Setting the mask to a value of 7 will effectively disable all interrupts (except the level 7 non-maskable interrupt).

The Data/Instruction Caches

The Atari TT030 and Falcon030 contain onboard data and instruction caches. These caches may be controlled by writing to the CACR register (in supervisor mode). The following table lists longword values that may be written to the CACR to enable or disable the caches:

<table>
<thead>
<tr>
<th>Value to Write to CACR</th>
<th>Effect</th>
</tr>
</thead>
<tbody>
<tr>
<td>0xA0A</td>
<td>Flush and disable both caches.</td>
</tr>
<tr>
<td>0x101</td>
<td>Enable both caches.</td>
</tr>
<tr>
<td>0xA00</td>
<td>Flush and disable the data cache.</td>
</tr>
<tr>
<td>0x100</td>
<td>Enable the data cache.</td>
</tr>
</tbody>
</table>

\(^1\)On a computer without an SCC chip, this interrupt is unused.

\(^2\)On a computer without a VME bus, this interrupt is unused.
5.4 – Hardware

<table>
<thead>
<tr>
<th></th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>0xA</td>
<td>Flush and disable the instruction cache.</td>
</tr>
<tr>
<td>0x1</td>
<td>Enable the instruction cache.</td>
</tr>
</tbody>
</table>

The 68881/882 Floating Point Coprocessor

A MC6888x math coprocessor may be installed in a Mega ST, Mega STe, or a Falcon030. The TT030 has one installed in its standard configuration. The 6888x is interfaced to the 68000 in peripheral mode and to the 68030 in coprocessor mode. Thus, the TT030 and Falcon030 computers access the 6888x in coprocessor mode while the Mega ST and MegaSTe computers access the 6888x in peripheral mode.

Coprocessor Mode

When the 6888x is interfaced in coprocessor mode, using it is as simple as placing floating-point instructions in the standard instruction stream (use a coprocessor ID of 1). The 68030 will properly dispatch the instruction and respond to exceptions through the following vectors:

<table>
<thead>
<tr>
<th>Vector Address</th>
<th>Assignment</th>
</tr>
</thead>
<tbody>
<tr>
<td>0x0000001C</td>
<td>FTRAPcc Instruction</td>
</tr>
<tr>
<td>0x0000002C</td>
<td>F-Line Emulator</td>
</tr>
<tr>
<td>0x00000034</td>
<td>Co-processor Protocol Violation</td>
</tr>
<tr>
<td>0x000000C0</td>
<td>Branch or Set on Unordered Condition</td>
</tr>
<tr>
<td>0x000000C4</td>
<td>Inexact Result</td>
</tr>
<tr>
<td>0x000000C8</td>
<td>Floating-Point Divide by Zero</td>
</tr>
<tr>
<td>0x000000CC</td>
<td>Underflow</td>
</tr>
<tr>
<td>0x000000D0</td>
<td>Operand Error</td>
</tr>
<tr>
<td>0x000000D4</td>
<td>Overflow</td>
</tr>
<tr>
<td>0x000000D8</td>
<td>Signaling NAN</td>
</tr>
</tbody>
</table>

Peripheral Mode

Utilizing an installed math coprocessor interfaced using peripheral mode requires the use of several hardware registers mapped to special coprocessor registers. Unlike most hardware registers, these do not have to be accessed in supervisor mode. Atari computers map the 6888x registers to the following locations:

<table>
<thead>
<tr>
<th>Address</th>
<th>Length</th>
<th>Register</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>0xFFFFFA40</td>
<td>WORD</td>
<td>FPCIR</td>
<td>Status register</td>
</tr>
<tr>
<td>0xFFFFFA42</td>
<td>WORD</td>
<td>FPCTL</td>
<td>Control Register</td>
</tr>
<tr>
<td>0xFFFFFA44</td>
<td>WORD</td>
<td>FPSAV</td>
<td>Save Register</td>
</tr>
<tr>
<td>0xFFFFFA46</td>
<td>WORD</td>
<td>FPREST</td>
<td>Restore Register</td>
</tr>
<tr>
<td>0xFFFFFA48</td>
<td>WORD</td>
<td>FPOPR</td>
<td>Operation word register</td>
</tr>
<tr>
<td>0xFFFFFA4A</td>
<td>WORD</td>
<td>FPCMD</td>
<td>Command register</td>
</tr>
<tr>
<td>0xFFFFFA4C</td>
<td>WORD</td>
<td>FPRES</td>
<td>Reserved</td>
</tr>
<tr>
<td>0xFFFFFA4E</td>
<td>WORD</td>
<td>FPCCR</td>
<td>Condition Code Register</td>
</tr>
<tr>
<td>0xFFFFFA50</td>
<td>LONG</td>
<td>FPOP</td>
<td>Operand Register</td>
</tr>
</tbody>
</table>

To execute a floating point instruction, use the following protocol for communicating data with the 6888x:
1. Wait for the chip to be idle.
2. Write a valid 6888x command to \textit{FPCMD}.
3. If necessary for the command, write an operand to \textit{FPOP}.
4. Wait for the status port to indicate the command is complete.
5. Read any return data from \textit{FPOP}.

Step one is achieved by waiting for a value of 0x0802 to appear in the status register (after ANDing with 0xBFFF) as follows:

\[
\text{while( (FPCIR \& 0xBFFF) \neq 0x0802 ) ;}
\]

Steps two and three involve writing the command word to \textit{FPCMD} and any necessary operand data to \textit{FPOP}. A primitive response code will be generated (and should be read) between each write to either \textit{FPCMD} or \textit{FPOP}. For a listing of primitive response codes returned by the 68881, consult the \textit{MC68881/68882 Floating-Point Coprocessor User’s Manual (2nd edition)}, Motorola publication MC68881UM/AD rev. 2, ISBN 0-13-567-009-8.

After the operation is complete (step 4), data may be read from the 68881 in \textit{FPOP} (step 5).

When sending or receiving data in \textit{FPOP}, the following chart details the transfer ordering and alignment:

<table>
<thead>
<tr>
<th>Order</th>
<th>BYTE</th>
<th>24 23 18 15 8</th>
<th>7 6</th>
</tr>
</thead>
<tbody>
<tr>
<td>1st</td>
<td>WORD</td>
<td></td>
<td></td>
</tr>
<tr>
<td>1st</td>
<td>LONG/SINGLE</td>
<td></td>
<td></td>
</tr>
<tr>
<td>1st</td>
<td>DOUBLE</td>
<td>Double Precision</td>
<td></td>
</tr>
<tr>
<td>2nd</td>
<td></td>
<td>Operands</td>
<td></td>
</tr>
<tr>
<td>2nd</td>
<td>EXTENDED</td>
<td>Extended</td>
<td></td>
</tr>
<tr>
<td>3rd</td>
<td></td>
<td>Precision</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>Operands</td>
<td></td>
</tr>
</tbody>
</table>

The following code demonstrates transferring two single precision floating-point numbers to the 68881, multiplying them, and returning the result.

```c
/* Number of iterations before an error is triggered */
#define FPCOUNT 0x80
#define FPCIR ((WORD *)(0xFFFFFA40L))
#define FPCMD ((WORD *)(0xFFFFFA4AL))
#define FPOP ((float *)(0xFFFFFA50L))
```
WORD fpcount, dum;

/* fperr() is user-defined */

#define FPwait() { fpcount = FPCOUNT; 
                 while(*FPCIR & 0xBFFF) != 0x0802) 
                 if(!(--fpcount)) fperr(); }

#define FPsglset(r,v) { FPwait(); 
                     *FPCMD = (0x5400 | ((r) << 7)); 
                     while(*FPCIR & 0xFFF0) != 0x8C00) 
                    if(!(--fpcount)) fperr();  
                     *FPOP = (v); }

#define FPsglmul(r1,r2) { FPwait(); 
                       *FPCMD = (0x0027 | ((r2) << 10) | ((r1) << 7));  
                       dum = *FPCIR + 1; }

/* dum = FPCIR +1; forces the status register to be read 
   (we assume the data’s good) */

#define FPsglget(r,var) { FPwait(); 
                       *FPCMD = (0x6400 | ((r) << 7)); 
                       while(*FPCIR != 0xb104) 
                       var = *FPOP; }

/* 
* void sglmul( float *f1, float *f2 ); 
* 
* Multiplies f1 by f2. Returns result in f1. 
* 
*/

void 
sglmul( float &f1, float &f2 )
{
    FPsglset( 0, *f1 );
    FPsglset( 1, *f2 );
    FPsglmul( 0, 1 );
    FPsglget( 0, *f1 );
}

### Cartridges

All Atari computers support an external 128K ROM cartridge port. Cartridges may be created to support applications or diagnostic tools. The 128K of address space allocated to cartridges appears from address 0xFA0000 to 0xFBFFFF. Newer Atari computers support larger cartridges (this is because the address space would no longer overlap the OS). All program code must be compiled to be relative of this base address.

The **LONG** appearing at 0xFA0000 determines the type of cartridge installed as follows:

<table>
<thead>
<tr>
<th>Cartridge</th>
<th>LONG Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>Application</td>
<td>0xABCDEF42</td>
</tr>
</tbody>
</table>
Diagnostic Cartridges

Diagnostic cartridges are executed almost immediately after a system reset. The OS uses a 680x0 JMP instruction to begin execution at address 0xFA0004 after having set the Interrupt Priority Level (IPL) to 7, entering supervisor mode, and executing a RESET instruction to reset external hardware devices.

Upon execution, register A6 will contain a return address which should be JMP’d to if you wish to continue system initialization at any point. The stack pointers will contain garbage. In addition, keep in mind that no hardware has been initialized, particularly the memory controller. All system memory sizing and initialization must be performed by the diagnostic cartridge.

Application Cartridges

Application cartridges should contain one or more application headers beginning at location 0xFA0004 as follows (one cartridge may contain one or many applications):

<table>
<thead>
<tr>
<th>Name</th>
<th>Offset</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>CA_NEXT</td>
<td>0x00</td>
<td>Pointer to the next application header (or NULL if there are no more).</td>
</tr>
<tr>
<td>CA_INIT</td>
<td>0x04</td>
<td>Pointer to the application’s initialization code. The high eight bits of this pointer have a special meaning as follows:</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Bit Set</td>
</tr>
<tr>
<td></td>
<td></td>
<td>0</td>
</tr>
<tr>
<td></td>
<td></td>
<td>1</td>
</tr>
<tr>
<td></td>
<td></td>
<td>2</td>
</tr>
<tr>
<td></td>
<td></td>
<td>3</td>
</tr>
<tr>
<td></td>
<td></td>
<td>4</td>
</tr>
<tr>
<td></td>
<td></td>
<td>5</td>
</tr>
<tr>
<td></td>
<td></td>
<td>6</td>
</tr>
<tr>
<td></td>
<td></td>
<td>7</td>
</tr>
<tr>
<td>CA_RUN</td>
<td>0x08</td>
<td>Pointer to application’s main entry point.</td>
</tr>
<tr>
<td>CA_TIME</td>
<td>0x0C</td>
<td><strong>GEMDOS</strong> time stamp.</td>
</tr>
<tr>
<td>CA_DATE</td>
<td>0x0E</td>
<td><strong>GEMDOS</strong> date stamp.</td>
</tr>
<tr>
<td>CA_SIZE</td>
<td>0x10</td>
<td>Size of application in bytes.</td>
</tr>
<tr>
<td>CA_NAME</td>
<td>0x14</td>
<td><strong>NULL</strong> terminated ASCII filename in standard <strong>GEMDOS</strong> 8+3 format.</td>
</tr>
</tbody>
</table>
When application cartridges are present, **GEMDOS** will allow a special ‘c’ (lowercase) drive to be accessed. Executable files appear on this drive as they would on any standard disk. This ‘drive’ may also be installed on the desktop.

### Game Controllers

The Atari 1040STe and Falcon030 support new enhanced joystick controls as well as older style CX-40 controls. For the usage and polling of the older style controls, refer to the following section which discusses the IKBD controller. This section will focus specifically on the newer style of controllers.

#### Joysticks

Enhanced joysticks are read by a two-step process. The **WORD** at address 0xFF9202 is written to using a mask which determines which values may subsequently be read from the **WORDS** at address 0xFF9200 and 0xFF9202. Valid mask values and the keys that may be read follow:

<table>
<thead>
<tr>
<th>Read Controller 0 at 0xFF9200</th>
<th>Write Mask</th>
<th>Bit 0</th>
<th>Bit 1</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>0xFFFF</td>
<td>Clear</td>
<td>Clear</td>
</tr>
<tr>
<td></td>
<td>0xFFFD</td>
<td>Clear</td>
<td>Clear</td>
</tr>
<tr>
<td></td>
<td>0xFFF7</td>
<td>-</td>
<td>-</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Read Controller 0 at 0xFF9202</th>
<th>Write Mask</th>
<th>Bit 8</th>
<th>Bit 9</th>
<th>Bit 10</th>
<th>Bit 11</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>0xFFFF</td>
<td>Up</td>
<td>Down</td>
<td>Left</td>
<td>Right</td>
</tr>
<tr>
<td></td>
<td>0xFFF7</td>
<td>Key #</td>
<td>Key 9</td>
<td>Key 6</td>
<td>Key 3</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Read Controller 1 at 0xFF9200</th>
<th>Write Mask</th>
<th>Bit 12</th>
<th>Bit 13</th>
<th>Bit 14</th>
<th>Bit 15</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>0xFFFF</td>
<td>Up</td>
<td>Down</td>
<td>Left</td>
<td>Right</td>
</tr>
<tr>
<td></td>
<td>0xFFF7</td>
<td>Key #</td>
<td>Key 9</td>
<td>Key 6</td>
<td>Key 3</td>
</tr>
</tbody>
</table>
To read the joystick, write a mask value corresponding to the row of keys/positions you wish to interrogate to 0xFF9202. Next, read back a \textit{WORD} from either 0xFF9200 or 0xFF9202. As indicated in the table, cleared bits mean that a key is being pressed or a joystick is moved in that direction.

**Paddles**

Two paddles may be plugged into each joystick port. Each paddle returns an 8-bit value indicating its position (0 = full counter-clockwise, 255 = full clockwise) at the addresses shown below. Unlike joysticks, paddle positions are returned automatically with no need to write to an address prior to a read. Paddle fire buttons, however, are mapped and read in the same manner as the joysticks. See the discussion of joysticks above for an explanation.

<table>
<thead>
<tr>
<th>Byte Address</th>
<th>Paddle</th>
</tr>
</thead>
<tbody>
<tr>
<td>0xFF9211</td>
<td>X Paddle 0</td>
</tr>
<tr>
<td>0xFF9213</td>
<td>Y Paddle 0</td>
</tr>
<tr>
<td>0xFF9215</td>
<td>X Paddle 1</td>
</tr>
<tr>
<td>0xFF9217</td>
<td>Y Paddle 1</td>
</tr>
</tbody>
</table>

**Light Gun/Pen**

Joystick port 0 supports a light gun or pen. The position that the gun is pointing to is returned in the \textit{WORD} registers at 0xFF9220 (X position) and 0xFF9222 (Y position). Only the lower 10 bits are significant giving a range of values from 0-1023.

**The IKBD Controller**

The Atari 16/32 bit computer line uses the Intelligent Keyboard Controller (IKBD) for keyboard, joystick (old-style CX-40), mouse, and clock communication. The 6850 ACIA serial communications chip is used to transfer data packets from the IKBD interface to the host computer.

The TOS calls \texttt{Bconout(4, ???), Ikbdws()}, and \texttt{Initmous()} handle communication to the controller. Return messages from the controller must be processed by placing a specialized handler in the vector table returned by the \texttt{XBIOS} call \texttt{Kbdvbase()}. \texttt{Kbdvbase()} returns the pointer to a vector table as follows:

```c
typedef struct
{
    void (*midivec)( UBYTE data ); /* Passed in d0 */
    void (*vkbderr)( UBYTE data ); /* Passed in d0 */
    void (*vmiderr)( UBYTE data ); /* Passed in d0 */
    void (*statvec)( char *packet ); /* Passed in a0 */
    void (*mousevec)( char *packet ); /* Passed in a0 */
    void (*clockvec)( char *packet ); /* Passed in a0 */
    void (*joyvec)( char *packet ); /* Passed in a0 */
    void (*midisys)( VOID );
    void (*ikbdsys)( VOID );
    char ikbdstate;

    ...
}
```

---

**THE ATARI COMpendium**
When an IKBD message is pending, the interrupt handler for the ACIAs calls either the midisys handler or the ikbdsys handler to retrieve the data and handle any errors. The default action for the ikbdsys handler is to decide whether the packet contains error, status, joystick, clock, or mouse information and to route it appropriately to vkbdderr, statvec, joyvec, clockvec, or mousevec. Keyboard packets are handled internally by ikbdsys.

Your handler should be patched into the appropriate vector and, if appropriate, expect the packet buffer to be pointed to by register A0. Unless your handler is designed to completely replace the functions of the default handler you should jump through the original vector pointer upon exit, otherwise simply use the 680x0 RTS instruction.

Each byte received through the keyboard ACIA falls into one of the following categories as follows:

<table>
<thead>
<tr>
<th>Category</th>
<th>Value(s)</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>Keyboard Make Code</td>
<td>0x00–0x7F</td>
<td>One of these values is generated each time a key is depressed. This value may be used with Keytbl() to generate an ASCII code for the scan code.</td>
</tr>
<tr>
<td>Keyboard Break Code</td>
<td>0x80–0xFF</td>
<td>This code is generated when a key previously depressed has been released. It represents the make code logically OR'ed with 0x80.</td>
</tr>
<tr>
<td>Status Packet Header</td>
<td>0xF6</td>
<td>This codes indicate the beginning of a multiple byte status packet.</td>
</tr>
<tr>
<td>Absolute Mouse Position</td>
<td>0xF7</td>
<td>See Below</td>
</tr>
<tr>
<td>Relative Mouse Position</td>
<td>0xF8–0xFB</td>
<td>See Below</td>
</tr>
<tr>
<td>Time-of-Day</td>
<td>0xFC</td>
<td>See Below</td>
</tr>
<tr>
<td>Joystick Report</td>
<td>0xFD</td>
<td>See Below</td>
</tr>
<tr>
<td>Joystick 0 Event</td>
<td>0xFE</td>
<td>See Below</td>
</tr>
<tr>
<td>Joystick 1 Event</td>
<td>0xFF</td>
<td>See Below</td>
</tr>
<tr>
<td>Status Packet Data</td>
<td>Any</td>
<td>When the ikbdstate variable (found in the KBDVECS structure) is non-zero, it represents the number of remaining bytes to retrieve that are part of a status packet and should thus not be treated as any of the above codes.</td>
</tr>
</tbody>
</table>
The Keyboard

Keyboard keys generate both a ‘make’ and ‘break’ code for each complete press and release respectively. The ‘make’ code is equivalent to the high byte of the IKBD scan code. ‘make’ codes are not related in any way to ASCII codes. They represent the physical position of the key in the keyboard matrix and may vary in keyboards designed for other countries. The XBIOS function Keytbl() provides lookup values which make internationalization possible. The key ‘break’ code is the ‘make’ code logically ORed with 0x80.

It should be noted that ‘key repeats’ are not generated by the ACIA but by a coordination of the ikbdsys and system timer handlers.

The Mouse

The mouse may be programmed to return position reports in either absolute, relative, or keycode mode (it is by default programmed to return relative position reports).

In relative reporting mode, the IKBD generates a mouse packet each time a mouse button is pressed or released, and every time the mouse is moved over a preset threshold distance (which is configurable). A relative mouse report packet is headed by a byte value between 0xF8 and 0xFB followed by the X and Y movement of the mouse as signed bytes. If the movement is greater than can be represented as signed bytes (-128 to 127), multiple packets are sent.

The header byte determines the state of the mouse buttons as follows:

<table>
<thead>
<tr>
<th>Header</th>
<th>Mouse Button State</th>
</tr>
</thead>
<tbody>
<tr>
<td>0xF8</td>
<td>No buttons depressed.</td>
</tr>
<tr>
<td>0xF9</td>
<td>Left button depressed.</td>
</tr>
<tr>
<td>0xFA</td>
<td>Right button depressed.</td>
</tr>
<tr>
<td>0xFB</td>
<td>Both buttons depressed.</td>
</tr>
</tbody>
</table>

In absolute reporting mode, the IKBD only generates a mouse packet when interrogated. Mouse packets in absolute mode are headed by a 0xF7 byte followed by the MSB and LSB of the X value and the MSB and LSB of the Y value respectively. The minimum and maximum X and Y values are user-definable.

Keycode reporting mode generates keyboard ‘make’ and ‘break’ codes for mouse movements rather than sending standard mouse packets. Mouse movement is translated into the arrow keys and the codes 0x74 and 0x75 represent the left and right mouse button respectively. ‘break’ codes are sent immediately after the corresponding ‘make’ code is delivered.
The Joystick

The basic CX-40 style joystick controls are still present on every Atari computer. Atari recommends that these ports should not be supported when STe/Falcon030 enhanced joysticks are present unless the option for four-player play is desired. While no direct TOS support is available for reading these ports, it is possible using the IKBD controller in one of several joystick reporting modes.

Joystick event reporting mode (the default) sends a joystick packet each time the status of one of the joysticks changes. The joystick packet header is 0xFE if the state of joystick 0 has changed or 0xFF if the status of joystick 1 has changed. This header byte is followed by a BYTE containing the new state of the joystick as follows:

```
Bit 7  Bit 0
□ □ □ □ □ □ □ □  Joystick Position
□ □ □ □ □ □ □ □  Trigger State ( 1 = depressed )
```

The four bits corresponding to joystick position can be interpreted as follows:

```
0101 (5)  0011 (3)  1001 (9)
0100 (4)  0000  1000 (8)
0110 (6)  0010 (2)  1010 (10)
```

Joysticks may be interrogated at any time by sending an interrogate command (as described later in this chapter). The packet response to this command is 0xFD followed by the BYTE report of joystick 0 and 1 (as shown above).

The joysticks may be placed into joystick monitoring or fire button monitoring mode. In these modes, all other IKBD communication is stopped and all processor time is devoted to the processing of packets. Joystick monitoring mode cause the IKBD to send a continuous stream of two-byte packets as follows: The first byte contains the status of joystick buttons 0 and 1 in bits 1 and 0 respectively. The second byte contains the position state of joystick 0 in the high nibble and joystick 1 in the lower nibble (the position state can be interpreted as shown in the diagram above).
Fire button monitoring mode constantly scans joystick button 1 and returns the results in **BYTES** packed with 8 reports each (one per bit). These modes may be paused or halted using the appropriate commands.

Joystick keycode mode is similar to mouse keycode mode. This mode translates all joystick position information into arrow keys. A ‘make’ code of 0x74 is generated when joystick button 0 is depressed and a ‘make’ code of 0x75 is generated when joystick button 1 is depressed. The rate at which the IKBD controller generates these joystick events can be controlled using commands discussed in the following section.

### Time-of-Day

The IKBD controller maintains a separate time-of-day clock that is kept synchronized with **GEMDOS** time by OS calls. A time-of-day packet may be requested using the method shown below under IKBD commands.

The response packet from the IKBD is seven bytes in length identified by its header byte of 0xFC and followed by six **UBYTES** containing the year (last two digits), month, day, hours (0-24), minutes, and seconds in BCD format (ex: a month byte in December would be 0x12).

### IKBD Commands

Commands may be sent to the IKBD using any of the **TOS** function calls described above. Some commands may generate packets while other commands change the operating state of the IKBD controller. Unrecognized command codes are treated as NOPs. The following lists valid IKBD command codes:

<table>
<thead>
<tr>
<th>Command BYTE</th>
<th>Result</th>
</tr>
</thead>
<tbody>
<tr>
<td>0x07</td>
<td>Set mouse button action. This command <strong>BYTE</strong> should be followed by a <strong>BYTE</strong> which describes how the mouse buttons should be treated as follows:</td>
</tr>
<tr>
<td></td>
<td><strong>BYTE</strong></td>
</tr>
<tr>
<td></td>
<td>0x00</td>
</tr>
<tr>
<td></td>
<td>0x01</td>
</tr>
<tr>
<td></td>
<td>0x02</td>
</tr>
<tr>
<td></td>
<td>0x03</td>
</tr>
<tr>
<td></td>
<td>0x04</td>
</tr>
<tr>
<td>0x08</td>
<td>Enable relative mouse position reporting (default).</td>
</tr>
<tr>
<td>0x09</td>
<td>Enable absolute mouse position reporting. This command is followed by the MSB and LSB of the X and Y coordinate maximum values for the mouse.</td>
</tr>
<tr>
<td>0x0A</td>
<td>Enable mouse keycode mode. This command is followed by two <strong>BYTES</strong> indicating the maximum number of mouse ‘ticks’ required to generate a keycode for the X and Y axis respectively.</td>
</tr>
<tr>
<td>Code</td>
<td>Description</td>
</tr>
<tr>
<td>-------</td>
<td>-------------</td>
</tr>
</tbody>
</table>
| 0x0B  | Set mouse threshold. This command is followed by two 
       | `BYTES` which determine the number of mouse 'ticks'
       | required to generate a mouse position report in 
       | relative positioning mode. |
| 0x0C  | Set mouse scale. This command is followed by two
       | `BYTES` which determine the number of mouse 'ticks'
       | for each single coordinate on the X and Y axis 
       | respectively. |
| 0x0D  | Interrogate mouse position. This command generates 
       | an absolute mouse position report. |
| 0x0E  | Load mouse position. This command sets the mouse 
       | position based on the current coordinate system 
       | in absolute reporting mode. The command is 
       | followed by a filler `BYTE` of 0x00 and the 
       | MSB and LSB of the new X and Y axis for the 
       | mouse. |
| 0x0F  | Set Y=0 to the bottom. This command changes the 
       | origin of the mouse coordinate system to the upper 
       | left of the screen. |
| 0x10  | Set Y=0 to the top. This command changes the 
       | origin of the mouse coordinate system to the lower 
       | left of the screen. |
| 0x11  | Resume sending data. This command (or for that 
       | matter any command) will cause the IKBD to resume 
       | sending packet data to the host. |
| 0x12  | Disable all mouse packet reporting. Any valid 
       | mouse command resets this state. If the mouse buttons 
       | have been programmed to act like keyboard keys, this 
       | command will have no effect on them. |
| 0x13  | Pause output. All output from the IKBD controller 
       | is halted until a 'Resume' or other command is 
       | received. |
| 0x14  | Set joystick event reporting mode. This command 
       | causes a joystick report to be generated whenever 
       | the state of either joystick changes. |
| 0x15  | Set joystick interrogation mode. This command 
       | causes the IKBD to generate joystick packets 
       | only when requested by the host. |
| 0x16  | Joystick interrogation. This command causes a 
       | joystick packet indicating the status of both 
       | joysticks to be generated. |
| 0x17  | Enables joystick monitoring mode. Besides serial 
       | communication and the maintenance of the time-of-
       | day clock, this command causes only special 
       | joystick reports to be generated. |
       | The command `BYTE` should be followed by a `BYTE`
       | indicating how often the joystick should be polled 
       | in increments of 1/100ths of a second. |
| 0x18  | Enables fire button monitoring mode. As above, this 
       | mode limits the IKBD to serial communication, 
       | updating the time-of-day clock, and the reporting 
<pre><code>   | of the state of joystick button 1. |
</code></pre>
<table>
<thead>
<tr>
<th>Command</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>0x19</td>
<td>Set joystick keycode mode. This command is followed by six <strong>BYTES</strong> as follows:</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th><strong>BYTE</strong></th>
<th><strong>Meaning</strong></th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>The length of time (in tenths of a second) before the horizontal breakpoint is reached.</td>
</tr>
<tr>
<td>2</td>
<td>Same as above for the vertical plane.</td>
</tr>
<tr>
<td>3</td>
<td>The length of time (in tenths of a second) between key repeats before the velocity breakpoint is reached.</td>
</tr>
<tr>
<td>4</td>
<td>Same as above for the vertical plane.</td>
</tr>
<tr>
<td>5</td>
<td>The length of time (in tenths of a second) between key repeats after the velocity breakpoint is reached.</td>
</tr>
<tr>
<td>6</td>
<td>Same as above for the vertical plane.</td>
</tr>
<tr>
<td>0x1A</td>
<td>Disable joystick event reporting.</td>
</tr>
<tr>
<td>0x1B</td>
<td>Set the time of day clock. This command is followed by six <strong>BYTES</strong> used to set the IKBD clock. These <strong>BYTES</strong> are in binary-coded decimal (BCD) format. Each <strong>BYTE</strong> contains two digits (0-9), one in each nibble. The format for these <strong>BYTES</strong> is as follows:</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th><strong>BYTE</strong></th>
<th><strong>Meaning</strong></th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>Year (last two digits)</td>
</tr>
<tr>
<td>2</td>
<td>Month</td>
</tr>
<tr>
<td>3</td>
<td>Date</td>
</tr>
<tr>
<td>4</td>
<td>Hours (0-23)</td>
</tr>
<tr>
<td>5</td>
<td>Minutes (0-59)</td>
</tr>
<tr>
<td>6</td>
<td>Seconds (0-59)</td>
</tr>
<tr>
<td>0x1C</td>
<td>Interrogate the time-of-day clock. This command returns a packet headed by the value 0xFC followed by six <strong>BYTES</strong> as indicated above.</td>
</tr>
<tr>
<td>0x20</td>
<td>Load <strong>BYTES</strong> into the IKBD memory. This command is followed by at least three <strong>BYTES</strong> containing the MSB and LSB of the address into which to load the data, the number of <strong>BYTES</strong> to load (0-127), and the data itself.</td>
</tr>
<tr>
<td>0x21</td>
<td>Read <strong>BYTES</strong> from the IKBD controller. This command is followed by two <strong>BYTES</strong> containing the MSB and LSB of the address to read from. This returns a packet headed by the <strong>BYTE</strong> values 0xF6 and 0x20 followed by the memory data.</td>
</tr>
<tr>
<td>0x22</td>
<td>Execute a subroutine on the IKBD controller. This command <strong>BYTE</strong> is followed by two <strong>BYTES</strong> containing the MSB and LSB of the memory location of the subroutine to execute.</td>
</tr>
<tr>
<td>0x80</td>
<td>Reset the IKBD controller. This command is actually a two-<strong>BYTE</strong> command. The <strong>BYTE</strong> 0x80 must be followed by a <strong>BYTE</strong> of 0x01 or the command will be ignored.</td>
</tr>
</tbody>
</table>
5.16 – Hardware

0x87
Return a status message containing the current mouse action state. After receiving this command the IKBD will respond by sending a status packet (which may be intercepted at statvec) as follows:

<table>
<thead>
<tr>
<th>BYTE</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>0xF6</td>
</tr>
<tr>
<td>2</td>
<td>0x07</td>
</tr>
<tr>
<td>3</td>
<td>Current mouse action state</td>
</tr>
<tr>
<td></td>
<td>(see command 0x07)</td>
</tr>
<tr>
<td>4-8</td>
<td>0</td>
</tr>
</tbody>
</table>

0x88
Return a status message containing the current mouse mode. After receiving this command the IKBD will respond by sending a status packet (which may be intercepted at statvec) as follows:

<table>
<thead>
<tr>
<th>BYTE</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>0xF6</td>
</tr>
<tr>
<td>2</td>
<td>Current mode as follows:</td>
</tr>
<tr>
<td></td>
<td>0x08 = Relative mode</td>
</tr>
<tr>
<td></td>
<td>0x09 = Absolute mode</td>
</tr>
<tr>
<td></td>
<td>0x0A = Keycode mode</td>
</tr>
<tr>
<td>3</td>
<td>Absolute mode: MSB of maximum X position</td>
</tr>
<tr>
<td></td>
<td>(units to current scale).</td>
</tr>
<tr>
<td></td>
<td>Keycode mode: Horizontal distance threshold</td>
</tr>
<tr>
<td></td>
<td>that must be passed prior to sending a keycode.</td>
</tr>
<tr>
<td></td>
<td>Relative mode: 0</td>
</tr>
<tr>
<td>4</td>
<td>Absolute mode: LSB of maximum X position.</td>
</tr>
<tr>
<td></td>
<td>Keycode mode: Vertical distance threshold</td>
</tr>
<tr>
<td></td>
<td>that must be passed prior to sending a keycode.</td>
</tr>
<tr>
<td></td>
<td>Relative mode: 0</td>
</tr>
<tr>
<td>5</td>
<td>Absolute mode: MSB of maximum Y position.</td>
</tr>
<tr>
<td></td>
<td>Keycode mode: 0</td>
</tr>
<tr>
<td></td>
<td>Relative mode: 0</td>
</tr>
<tr>
<td>6</td>
<td>Absolute mode: LSB of maximum Y position.</td>
</tr>
<tr>
<td></td>
<td>Keycode mode: 0</td>
</tr>
<tr>
<td></td>
<td>Relative mode: 0</td>
</tr>
<tr>
<td>7-8</td>
<td>0</td>
</tr>
</tbody>
</table>

0X89
Same as 0x88.

0X8A
Same as 0x88.
0x8B Return a status message containing the current mouse threshold state. After receiving this command the IKBD will respond by sending a status packet (which may be intercepted at statvec) as follows:

<table>
<thead>
<tr>
<th>BYTE</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>0xF6</td>
</tr>
<tr>
<td>2</td>
<td>0x0B</td>
</tr>
<tr>
<td>3</td>
<td>Number of horizontal mouse ‘ticks’ that must be traveled prior to sending a mouse packet.</td>
</tr>
<tr>
<td>4</td>
<td>Number of vertical mouse ‘ticks’ that must be traveled prior to sending a mouse packet.</td>
</tr>
<tr>
<td>5-8</td>
<td>0</td>
</tr>
</tbody>
</table>

0x8C Return a status message containing the current mouse scaling factor. After receiving this command the IKBD will respond by sending a status packet (which may be intercepted at statvec) as follows:

<table>
<thead>
<tr>
<th>BYTE</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>0xF6</td>
</tr>
<tr>
<td>2</td>
<td>0x0C</td>
</tr>
<tr>
<td>3</td>
<td>Horizontal mouse ‘ticks’ between a change in mouse position on the X axis.</td>
</tr>
<tr>
<td>4</td>
<td>Vertical mouse ‘ticks’ between a change in mouse position on the Y axis.</td>
</tr>
<tr>
<td>5-8</td>
<td>0</td>
</tr>
</tbody>
</table>

0x8F Return a status message containing the current origin point of the Y axis used for mouse position reporting. After receiving this command the IKBD will respond by sending a status packet (which may be intercepted at statvec) as follows:

<table>
<thead>
<tr>
<th>BYTE</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>0xF6</td>
</tr>
<tr>
<td>2</td>
<td>0x0F = Bottom is (Y=0)</td>
</tr>
<tr>
<td></td>
<td>0x10 = Top is (Y=0)</td>
</tr>
<tr>
<td>3-8</td>
<td>0</td>
</tr>
</tbody>
</table>

0x90 Same as 0x8F.

0x92 Return a status message containing the current state of mouse reporting. After receiving this command the IKBD will respond by sending a status packet (which may be intercepted at statvec) as follows:

<table>
<thead>
<tr>
<th>BYTE</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>0xF6</td>
</tr>
<tr>
<td>2</td>
<td>0x00 = Mouse reporting enabled.</td>
</tr>
<tr>
<td></td>
<td>0x12 = Mouse reporting disabled.</td>
</tr>
<tr>
<td>3-8</td>
<td>0</td>
</tr>
</tbody>
</table>
0x94   Return a status message containing the current joystick mode. After receiving this command the IKBD will respond by sending a status packet (which may be intercepted at statvec) as follows:

<table>
<thead>
<tr>
<th>BYTE</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>0xF6</td>
</tr>
</tbody>
</table>

2 Current mode as follows:
   - 0x14 = Event reporting mode
   - 0x15 = Interrogation mode
   - 0x19 = Keycode mode

3 *Keycode mode:* This value represents the amount of time (in tenths of a second) that keycodes are returned to the host for horizontal position events at the initial velocity level (after this time expires, the secondary velocity level is used).
   - *Event recording mode:* 0
   - *Interrogation mode:* 0

4 *Keycode mode:* Same as BYTE 3 for vertical events.
   - *Event recording mode:* 0
   - *Interrogation mode:* 0

5 *Keycode mode:* This value represents the initial horizontal velocity level (in tenths of a second). This is the initial rate at which keycodes are generated.
   - *Event recording mode:* 0
   - *Interrogation mode:* 0

6 *Keycode mode:* Same as byte 5 for vertical events.
   - *Event recording mode:* 0
   - *Interrogation mode:* 0

7 *Keycode mode:* This value represents the secondary horizontal velocity level (in tenths of a second). This is the rate used after the amount of time specified in bytes 3-4 expires.
   - *Event recording mode:* 0
   - *Interrogation mode:* 0

8 *Keycode mode:* Same as byte 7 for vertical events.
   - *Event recording mode:* 0
   - *Interrogation mode:* 0

0x95   Same as 0x94.

0x99   Same as 0x94.
Return a status message containing the current status of the joystick. After receiving this command the IKBD will respond by sending a status packet (which may be intercepted at statvec) as follows:

<table>
<thead>
<tr>
<th>BYTE</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>0xF6</td>
</tr>
<tr>
<td>2</td>
<td>0x00 = Joystick enabled</td>
</tr>
<tr>
<td></td>
<td>0x1A = Joystick disabled</td>
</tr>
<tr>
<td>3-8</td>
<td>0</td>
</tr>
</tbody>
</table>

STe/TT030 DMA Sound

The Atari STe, Mega STe, TT030, and Falcon030 are all equipped with the ability to playback stereo digital audio. Only the Falcon030, however, has supporting XBIOS calls which eliminate the need for the programmer to directly access the sound system hardware. Although the Falcon030 has a more sophisticated sound system than the earlier Atari machines, the hardware registers have been kept compatible so older applications should function as expected. Programmers designing Falcon030 applications which use digital audio should use the appropriate XBIOS calls.

The STe, MegaSTe, and TT030 support 8-bit monophonic or stereophonic sound samples. Samples should be signed (-128 to 127) with alternating left and right channels (for stereo) beginning with the left channel. Samples may be played at 50 kHz, 25 kHz, 12.5 kHz, or 6.25 kHz (6.25 kHz is not supported on the Falcon030).

DMA Sound Registers

Several hardware registers control DMA sound output as follows:

<table>
<thead>
<tr>
<th>Address</th>
<th>Bit Layout</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>0xFF8900</td>
<td>---- -----</td>
<td>Sound DMA Control</td>
</tr>
<tr>
<td>0xFF8902</td>
<td>---- -----</td>
<td>Frame Base Address High (bits 21-16)</td>
</tr>
<tr>
<td>0xFF8904</td>
<td>---- -----</td>
<td>Frame Base Address Middle (bits 15-8)</td>
</tr>
<tr>
<td>0xFF8906</td>
<td>---- -----</td>
<td>Frame Base Address Low (bits 7-1)</td>
</tr>
<tr>
<td>0xFF8908</td>
<td>---- -----</td>
<td>Frame Address Counter (bits 21-16)</td>
</tr>
<tr>
<td>0xFF890A</td>
<td>---- -----</td>
<td>Frame Address Counter (bits 15-8)</td>
</tr>
<tr>
<td>0xFF890C</td>
<td>---- -----</td>
<td>Frame Address Counter (bits 7-1)</td>
</tr>
<tr>
<td>0xFF890E</td>
<td>---- -----</td>
<td>Frame End Address High (bits 21-16)</td>
</tr>
<tr>
<td>0xFF8910</td>
<td>---- -----</td>
<td>Frame End Address Middle (bits 15-8)</td>
</tr>
<tr>
<td>0xFF8912</td>
<td>---- -----</td>
<td>Frame End Address Low (bits 7-1)</td>
</tr>
<tr>
<td>0xFF8920</td>
<td>0000 0000 m000 00rr</td>
<td>Sound Mode Control</td>
</tr>
</tbody>
</table>

Addresses placed in the three groups of address pointer registers must begin on an even address. In addition, only sounds within the first 4 megabytes of memory may be accessed (this limitation has been lifted on the Falcon030). Sounds may not be played from alternate RAM.
Playing a Sound

To begin sound playback, place the start address of the sound in the Frame Base Address registers. Place the address of the end of the sound in the Frame End Address registers. The address of the end of the sound should actually be the first byte in memory past the last byte of the sample.

Set the Sound Mode Control register to the proper value. Bit 7, notated as ‘m’ should be set to 1 for a monophonic sample or 0 for a stereophonic sample. Bits 0 and 1, notated as ‘r’, control the sample playback rate as follows:

<table>
<thead>
<tr>
<th>‘r’</th>
<th>Playback Rate</th>
</tr>
</thead>
<tbody>
<tr>
<td>00</td>
<td>6258 Hz</td>
</tr>
<tr>
<td>01</td>
<td>12517 Hz</td>
</tr>
<tr>
<td>10</td>
<td>25033 Hz</td>
</tr>
<tr>
<td>11</td>
<td>50066 Hz</td>
</tr>
</tbody>
</table>

To begin the sample playback, set bits 0 and 1 of the Sound DMA Control register, notated as ‘c’, as follows:

<table>
<thead>
<tr>
<th>‘c’</th>
<th>Sound Control</th>
</tr>
</thead>
<tbody>
<tr>
<td>00</td>
<td>Sound Disabled (this will stop any sound currently being played)</td>
</tr>
<tr>
<td>01</td>
<td>Sound Enabled (play once)</td>
</tr>
<tr>
<td>11</td>
<td>Sound Enabled (repeat until stopped)</td>
</tr>
</tbody>
</table>

Sound playback may be prematurely halted by writing a 0 to address 0x00FF8900.

Sound Interrupts using MFP Timer A

Discontinuous sample frames may be linked together using the MFP Timer A interrupt. When a sound is played using repeat mode an interrupt is generated at the end of every frame. By configuring Timer A to ‘event count’ mode you can ensure the seamless linkage and variable repeating of frames.

For example, suppose you have three sample frames, A, B, and C, in memory and you want to play A five times, B five times, and C only once. Use the following steps to properly configure Timer A and achieve the desired result:

- Use Xbtimer() to set Timer A to event count mode with a data value of 4 (the first data value should be one less than actually desired since the sound will play once before the interrupt occurs).
- Configure the sound registers as desired and start sound playback in repeat mode.
- When the interrupt fires, place the address of frame B in the sound playback registers (these values aren’t actually used until the current frame finishes).
- Reset Timer A’s data register to 5 and exit your interrupt handler.
• When the second interrupt fires, place the address of frame C in the sound playback registers.
• Reset Timer A’s data register to 1 and exit your interrupt handler.
• When the final interrupt is triggered, write a 0x01 to the sound control register to cause sound playback to end at the end of the current frame.

Sound Interrupts using GPIP 7

Another method of generating interrupts at the end of sound frames is by using the MFP’s General Purpose Interrupt Port (GPIP) 7. This interrupt does not support an event count mode so it will generate an interrupt at the end of every frame. In addition, the interrupt must be configured differently depending on the type of monitor connected to the system (this is because GPIP 7 serves double-duty as the monochrome detect signal).

To program GPIP 7 for interrupts, disable all DMA sound by placing a 0x00 in the sound control register. Next, check bit 7 of the GPIP port at location 0xFFFFA01. If a monochrome monitor is connected the bit will be 0. The bit will be 1 if a color monitor is connected.

Bit 7 of the MFP’s active edge register (at 0xFFFFA03) should be set to the opposite of the GPIP port’s bit 7. This will cause an interrupt to trigger at the end of every frame. Use Mfpint() to set the location of your interrupt handler and Jenabint() to enable interrupts. From this point, interrupts will be generated at the end of every frame playing in ‘play once’ mode or repeat mode until the interrupt is disabled.

The MICROWIRE Interface

The STe and TT030 computers use the MICROWIRE interface to control volume, mixing of the PSG and DMA output, and tone control. The original ST is limited to amplitude control through the use of the appropriate PSG register. The Falcon030 supports new XBIOS calls which allow volume and mixing control.

The MICROWIRE interface is a write-only device accessed using two hardware registers 0xFFFF8924 (mask) and 0xFFFF8922 (data). To write a command to the MICROWIRE you must first place the value 0x07FF into the mask register and then write the appropriate command to the data register. The format for the data WORD is shown below:

<table>
<thead>
<tr>
<th>Bit 15</th>
<th>Bit 14</th>
<th>Bit 13</th>
<th>Bit 12</th>
<th>Bit 11</th>
<th>Bit 10</th>
<th>Bit 9</th>
<th>Bit 8</th>
<th>Bit 7</th>
<th>Bit 6</th>
<th>Bit 5</th>
<th>Bit 4</th>
<th>Bit 3</th>
<th>Bit 2</th>
<th>Bit 1</th>
<th>Bit 0</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>x</td>
<td>x</td>
<td>x</td>
<td>x</td>
<td>x</td>
<td>1</td>
<td>0</td>
<td>c</td>
<td>c</td>
<td>c</td>
<td>c</td>
</tr>
</tbody>
</table>

Bits labeled ‘x’ will be ignored. Bits 9 and 10 should always be %10 to correctly specify the device address which is a constant. Bits labeled ‘c’ specify the command and bits labeled ‘d’ contain the appropriate data for the command. The following table explains the valid MICROWIRE commands:
<table>
<thead>
<tr>
<th>Command</th>
<th>Command Value</th>
<th>'ccc'</th>
<th>'ddddd'</th>
<th>Example Value</th>
<th>Result</th>
</tr>
</thead>
<tbody>
<tr>
<td>Set Master Volume</td>
<td>011</td>
<td></td>
<td></td>
<td>%000000</td>
<td>-80dB Attenuation</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td>%010100</td>
<td>-40dB Attenuation</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td>%101000</td>
<td>0dB Attenuation (Maximum)</td>
</tr>
<tr>
<td>Set Left Channel Volume</td>
<td>101</td>
<td></td>
<td></td>
<td>%000000</td>
<td>-40dB Attenuation</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td>%001010</td>
<td>-20dB Attenuation</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td>%010100</td>
<td>0dB Attenuation (Maximum)</td>
</tr>
<tr>
<td>Set Right Channel Volume</td>
<td>100</td>
<td></td>
<td></td>
<td>%000000</td>
<td>-40dB Attenuation</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td>%001010</td>
<td>-20dB Attenuation</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td>%010100</td>
<td>0dB Attenuation (Maximum)</td>
</tr>
<tr>
<td>Set Treble</td>
<td>010</td>
<td></td>
<td></td>
<td>%000000</td>
<td>-12dB Attenuation</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td>%000110</td>
<td>0dB Attenuation</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td>%001100</td>
<td>+12dB Attenuation (Maximum)</td>
</tr>
<tr>
<td>Set Bass</td>
<td>001</td>
<td></td>
<td></td>
<td>%000000</td>
<td>-12dB Attenuation</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td>%000110</td>
<td>0dB Attenuation</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td>%001100</td>
<td>+12dB Attenuation (Maximum)</td>
</tr>
<tr>
<td>Set PSG/DMA Mix</td>
<td>000</td>
<td></td>
<td></td>
<td>%000000</td>
<td>-12dB Attenuation</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td>%000001</td>
<td>Mix PSG sound output.</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td>%000010</td>
<td>Don't Mix PSG sound output.</td>
</tr>
</tbody>
</table>

When configuring multiple settings at once, you should program a delay between writes since the MICROWIRE takes at least 16μsec to completely read the data register. During a read the MICROWIRE rotates the mask register one bit at a time. You will know a read operation has completed when the mask register returns to 0x07FF. The following assembly segment illustrates this by setting the left and right channel volumes to their maximum values:

```
MWMASK EQU $FFFF8924
MWDATA EQU $FFFF8922
MASKVAL EQU $7FF
HIGHLVOL EQU $554
HIGHRVOL EQU $514

.text
maxvol:
move.w  MASKVAL,MWMASK  ; First write the mask and data values
move.w  #HIGHLVOL,MWDATA
mwwrite:
cmp.w   MASKVAL,MWMASK  
bne.s   mwwrite          ; loop until MWMASK reaches $7FF again
move.w  #HIGHRVOL,MWDATA
rts
.end
```
Video Hardware

Video Resolutions
Atari computers support a wide range of video resolutions as shown in the following tables:

<table>
<thead>
<tr>
<th>Computer System</th>
<th>Modes (width ´ height ´ colors)</th>
<th>Possible Colors</th>
</tr>
</thead>
<tbody>
<tr>
<td>ST, Mega ST</td>
<td>320x200x16 640x200x4 640x400x2</td>
<td>512</td>
</tr>
<tr>
<td>STe, Mega STe</td>
<td>320x200x16 640x200x4 640x400x2</td>
<td>4096</td>
</tr>
<tr>
<td>STacy</td>
<td>640x400x2</td>
<td>N/A</td>
</tr>
<tr>
<td>TT030</td>
<td>320x200x256 640x200x4 640x400x2 320x480x256 640x480x16</td>
<td>4096</td>
</tr>
<tr>
<td>Falcon030</td>
<td>See below.</td>
<td>262,144</td>
</tr>
</tbody>
</table>

Falcon030 Video Modes
The Falcon030 is equipped with a much more flexible video controller than earlier Atari computers. The display area may be output on a standard television, an Atari color or monochrome monitor, or a VGA monitor. Overscan is supported with all monitor configurations with the exception of VGA. Also, hardware support for NTSC and PAL monitors is software configurable.

The Falcon030 supports graphic modes of 40 or 80 columns (320 or 640 pixels across) containing 1, 2, 4, 8, or 16 bits per pixel resulting in 2, 4, 16, 256, or 262,144 colors respectively. All modes except the 16 bit per pixel mode supply the video shifter with palette indexes. The 16 bit per pixel mode is a ‘true-color’ mode where each 16 bit value determines the color rather than being an index into a palette. Each 16 bit WORD value is arranged as follows:

```
| R | R | R | R | G | G | G | G | G | G | B | B | B | B |
```

Falcon030 True-Color Video Word

The ‘R’, ‘G’, and ‘B’, represent the red, green, and blue components of the color. Because red and blue are each allocated five bits, they can represent a color range of 0-31. The green component is allocated six bits so it can represent a color range of 0-63.

The Falcon030 also supports an overlay mode (see VsetMask()) where certain colors can be defined as transparent to a connected Genlock (or similar) device. In this mode, the least significant green bit (Bit #5) is treated as the transparent flag bit and the resolution of the green
color component is slightly reduced. If the transparent flag bit of a pixel is set, that pixel will display video from the Falcon030’s video shifter, otherwise the external video source will be responsible for its display.

Another feature of the Falcon030’s video shifter is an optional interlace/double-line mode. When operating on a VGA monitor, this mode doubles the pixel height effectively reducing the vertical screen resolution by half. On any other video display, this mode engages interlacing which increases the video resolution.

The operating system calls `VsetMode()` or `VsetScreen()` can be used to manipulate the operating mode of the Falcon030’s video shifter. These calls do not, however, do any checking to ensure the selected video mode is actually attainable on the connected monitor or that the mode is legal. In particular, you should not attempt to set the video shifter to either 40 column mode with only one bit per pixel or 80 column VGA mode with 16 bits per pixel.

### Video Memory

Most of the available video modes are palette-based. The number of bits required per pixel depends on the number of palette entries as shown in the table below. The Falcon030 also offers a true color video mode which requires 16 bits per pixel.

<table>
<thead>
<tr>
<th>Palette Entries</th>
<th>Bits per Pixel</th>
</tr>
</thead>
<tbody>
<tr>
<td>2</td>
<td>1</td>
</tr>
<tr>
<td>4</td>
<td>2</td>
</tr>
<tr>
<td>16</td>
<td>4</td>
</tr>
<tr>
<td>256</td>
<td>8</td>
</tr>
</tbody>
</table>

Directly accessing video memory is normally not recommended because it may create compatibility problems with future machines and wreak havoc with other system applications. The VDI provides a rich set of function calls which should be used when outputting to the screen. The function call `vr_trnfm()`, for instance, can be useful in transforming video images into a pattern compatible with the current video shifter. Certain software, however, does need exclusive access to video memory.

With the exception of the 16-bit true color mode of the Falcon030, all video images are stored in memory in WORD interleaved format. The video shifter grabs one at a time from each plane present as shown in the following diagram which represents a 16-color (four plane) screen layout:
The Falcon030’s 16-bit true color mode is pixel-packed so that WORD #0 in memory is the complete color WORD for the pixel at (0, 0), WORD #1 is the complete color WORD for the pixel at (1, 0), etc.

**Fine Scrolling**

All Atari computers except the original ST and Mega ST support both horizontal and vertical fine scrolling in hardware. To accomplish this, an application must place a special handler in the vertical blank vector (at 0x00000070) which resets the scroll registers and video base address as needed.

The following registers are manipulated during the vertical-blank period to shift the screen across any number of virtual ‘screens’:

<table>
<thead>
<tr>
<th>Register</th>
<th>Address</th>
<th>Contents</th>
</tr>
</thead>
<tbody>
<tr>
<td>VBASEHI</td>
<td>0xFFFF8200</td>
<td>Low byte contains bits 23-16 of the video display base address.</td>
</tr>
<tr>
<td>VBASEMID</td>
<td>0xFFFF8202</td>
<td>Low byte contains bits 15-8 of the video display base address.</td>
</tr>
<tr>
<td>VBASELO</td>
<td>0xFFFF820C</td>
<td>Low byte contains bits 7-0 of the video display base address.</td>
</tr>
<tr>
<td>LINEWID</td>
<td>0xFFFF820E</td>
<td>Number of extra WORDs per scanline (normally 0).</td>
</tr>
<tr>
<td>HSCROLL</td>
<td>0xFFFF8264</td>
<td>Low four bits contain the bitwise offset (0-15) of the screen (normally 0 unless scrolling is in effect).</td>
</tr>
<tr>
<td>VCOUNTHI</td>
<td>0xFFFF8204</td>
<td>Low byte contains bits 23-16 of the current video refresh address (use with care).</td>
</tr>
<tr>
<td>VCOUNTMID</td>
<td>0xFFFF8206</td>
<td>Low byte contains bits 15-8 of the current video refresh address (use with care).</td>
</tr>
<tr>
<td>VCOUNTLO</td>
<td>0xFFFF8208</td>
<td>Low byte contains bits 7-0 of the current video refresh address (use with care).</td>
</tr>
</tbody>
</table>
To accommodate virtual screens wider than the display can show, set `LINEWID` to the number of extra WORDs per scanline. For instance, to create a virtual display two screens wide for a 320x200 16-color display, set `LINEWID` to 80.

To scroll vertically, simply alter the video base address by adding or subtracting the number of WORDs per scanline for each line you wish to scroll during the vertical blank.

To scroll horizontally, alter the video base address in WORD increments to move the physical screen left and right over the virtual screen. This by itself will cause the screen to skip in 16 pixel jumps. To scroll smoothly, use the `HSCROLL` register to shift the display accordingly. When `HSCROLL` is non-zero, subtract one from `LINEWID` for each plane.

To illustrate this more clearly, imagine a physical screen of 320x200 (16 colors) which is laid out over 4 virtual screens in a 2x2 grid. The following diagram and table shows example values to move the physical screen to the desired virtual coordinates:

<table>
<thead>
<tr>
<th>Virtual Coordinates</th>
<th>VBASE Address</th>
<th>LINEWID</th>
<th>HSCROLL</th>
</tr>
</thead>
<tbody>
<tr>
<td>(0, 0)</td>
<td>0x80000</td>
<td>80</td>
<td>0</td>
</tr>
<tr>
<td>(16, 0)</td>
<td>0x80004</td>
<td>80</td>
<td>0</td>
</tr>
<tr>
<td>(0, 1)</td>
<td>0x80140</td>
<td>80</td>
<td>0</td>
</tr>
<tr>
<td>(1, 0)</td>
<td>0x80000</td>
<td>76</td>
<td>1</td>
</tr>
<tr>
<td>(0, 10)</td>
<td>0x80B40</td>
<td>80</td>
<td>0</td>
</tr>
<tr>
<td>(100, 100)</td>
<td>0x87BE4</td>
<td>76</td>
<td>4</td>
</tr>
</tbody>
</table>
Overview

The Application Environment Services (AES) compose the highest level of the operating system. The AES uses the VDI, GEMDOS, and XBIOS to provide a global utility library of calls which provide applications with the GEM interface. Usage of the AES makes application development simpler and makes user interfaces more consistent. The services provided by the AES include:

- Application Control/Interaction
- Event Management
- Menu Services
- Object Rendering/Manipulation
- Form Management
- Graphic Utility Functions
- Scrap (Clipboard) Management
- Common Dialog Display
- Window Management
- Resource Management
- Shell (Desktop) Interaction

System-specific AES information and variables may be determined through reserved fields in the application’s global array (see appl_init()) or by using the various modes of appl_getinfo().

Process Handling

The AES manages two types of user programs. Normal GEM applications have file extensions of ‘.PRG’, ‘.APP’, or ‘.GTP’. Desk Accessories have file extensions of ‘.ACC’.

Without MultiTOS, the AES can have a maximum of one application and six desk accessories (four desk accessories under TOS 1.0) executing concurrently. The currently running application (or the Desktop if no application is running) is given primary control over the system. Desk accessories are allocated processor time only when the foreground application releases control by calling one of the event library functions. An application which does not have a standard event loop (as illustrated below) will cause desk accessories to stop functioning while it is being executed.
Under MultiTOS, an unlimited amount of applications and desk accessories may be loaded concurrently\(^1\). MultiTOS is a pre-emptive system where all system processes are given time regardless of other applications.

### Applications

When an application is launched, GEM allocates all remaining system memory and loads the application into this area\(^2\). It is the responsibility of the application to free whatever memory it doesn’t immediately need for its text, data, bss, and stack area. Most high level languages do this for you in the startup stub linked with every application.

GEM applications begin with an `appl_init()` function call. This call will return a valid application ID if the process can be successfully registered or a -1 if it fails. If the call fails, the application should immediately exit without making any AES calls. Upon success, however, the ID should be stored for future use within the application. Applications running under MultiTOS should call `menu_register()` to display the program title in the application list rather than the filename.

The next steps a GEM application will follow are variable, however, most GEM applications will initialize themselves further by performing some or all of the following steps:

- Open a VDI workstation.
- Verify that the computer the application is being run on has the minimum requirements (screen resolution, OS versions, memory needs, hardware features) necessary to continue.
- Load the application `.RSC` file and fix it up as necessary.
- Display the menu bar.
- Change the mouse form to an arrow (the AES defaults to a BUSY_BEE shape).
- Enter the application’s main event loop.

The following represents a basic skeleton for an AES application:

```c
#include <AES.H>
#include <VDI.H>
#include <OSBIND.H>
#include <VDIWORK.H>
#include "skel.h"

#define CNTRL_Q 0x11
```

\(^1\)Some MultiTOS versions limit this based upon the available space in the leftmost menu.

\(^2\)TOS 5.0 does allow the user to set limits on the amount of memory allowed to an application.
int main(int argc, char *argv[]) {

char *altNoVDIWork = "[3][GEM is unable to allocate a workstation. The program must abort.][ OK ]";
char *altNoRSC = "[3][The program cannot locate SKELETON.RSC. Please ensure that it resides in the same directory as SKELETON.PRG.][ OK ]";
short ret, msg[8], kc, quit, dum;

ap_id = appl_init();
if(ap_id == -1)
    return -1;

if(!OpenVwork(&ws))
{
    graf_mouse( ARROW, 0L );
    form_alert(1, altNoVDIWork);
    appl_exit();
    return -1;
}

if(!rsrc_load( RSCname ))
{
    graf_mouse( ARROW, 0L );
    form_alert(1, altNoRSC);
    v_clsvwk(ws.handle);
    appl_exit();
    return -1;
}

if(_AESglobal[1] == -1) /* MultiTOS present? */
{
    menu_register(ap_id, menu_title); /* Yes, make name pretty. */

    rsrc_gaddr( R_TREE, MAINMENU, &mainmenu);
    menu_bar(mainmenu,1);
    graf_mouse( ARROW, 0L );

    quit = FALSE;
    while(!quit)
    {
        ret = evnt_multi(MU_MESAG|MU_KEYBD,2,1,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,
                         &dum,&dum,&dum,&dum,&dum,&kcs_dum);
        if(ret & MU_MESAG)
        {
switch(msg[0])
{
  case MN_SELECTED:
    switch(msg[3])
    {
      /* Other menu selections */
      /* Other menu selections */
      /* Other menu selections */
    }
  case mmExit: /* Defined in SKEL.H */
    quit = TRUE;
    break;
    break;
}
}

if(ret & MU_KEYBD)
{
  switch(kc & 0xFF)
  {
    /* Other keyboard equivalents */
    /* Other keyboard equivalents */
    /* Other keyboard equivalents */
    case CNTRL_Q:
      quit = TRUE;
      break;
  }
}
}

menu_bar( mainmenu, 0 );
v_clsvwk(ws.handle);
rsrce_free();
appl_exit();
return 0;
}

The Command Line

GEM applications, like TOS applications, may be started with a command line (for a detailed description of command line processing, see Chapter 2: GEMDOS). `.PRG` files and `.APP` files will have items on the command line if a document file which was registered with the application was double-clicked or if a valid document file was dropped over the application’s icon in the Desktop. Launching a `.GTP` application will cause the Desktop to prompt the user for a command line in the same manner as `.TTP` programs are handled. Applications which find one or more valid document names on their command line should automatically load them on program start.
Desk Accessories

Upon bootup, any files with the extension `.ACC` found in the root directory of the user’s boot drive will be loaded and executed up until their first event library call. MultiTOS allows desk accessories to be loaded and unloaded after bootup.

Unlike applications, desk accessories are not given all of available system memory on startup. They are only allocated enough memory for their text, data, and bss segments. No stack space is allocated for a desk accessory either. Many high level language stubs reserve space in the BSS or overwrite startup code to provide a stack but keep in mind that desk accessory stacks are usually small compared to applications.

As with applications, GEM desk accessories should begin with an `appl_init()` function call. Upon success, the ID should be stored and used within a `menu_register()` call to place the applications’ name on the menu bar.

Desk accessories, unlike applications, do not begin user interaction immediately. Most desk accessories initialize themselves and enter a message loop waiting for an `AC_OPEN` message. Some desk accessories wait for timer events or custom messages from another application. After being triggered, they usually open a window in which user interaction may be performed (dialogs and alerts may also be presented but are not recommended because they prevent shuffling between other system processes).

Desk accessories should not use a menu bar and should never exit (unless `appl_init()` fails) after calling `menu_register()`. If an error condition occurs which would make the accessory unusable, simply enter an indefinite message loop.

Any resources loaded by an accessory should be loaded prior to entering the first event loop and should never be freed after the accessory has called `menu_register()`. Resource data for desk accessories should be embedded in the executable rather than being soft-loaded because memory allocated to a desk accessory is not freed during a resolution change on TOS versions less than 2.06. This causes resource memory allocated by `rsrc_load()` to be lost to the system after a resolution change and will likely cause memory fragmentation.

An `AC_CLOSE` message is sent to an accessory when it is being closed at the request of the OS. At this point, it should perform any cleanup necessary to release system resources and close files opened at `AC_OPEN` (accessory windows will be closed automatically by the AES). After cleanup, the event loop should be reentered to wait for subsequent `AC_OPEN` messages.

The following code represents a basic skeleton for an AES desk accessory:

```c
#include <AES.H>
#include <VDI.H>
#include <OSBIND.H>
#include <VDIWORK.H>

```
int main(int argc, char *argv[])
{
    char *altNoVDIWork = "[3] [GEM is unable to allocate a workstation. The program must abort.] [ OK ]";
    short ret, msg[8], kc, dum;

    ap_id = appl_init();
    if (ap_id == -1)
        return -1;

    if (!OpenVwork(&ws))
    {
        form_alert(1, altNoVDIWork);
        appl_exit();
        return -1;
    }

    menu_id = menu_register(ap_id, menu_title); /* Place name on menu bar */

    for(;;)
    {
        evnt_mesag(msg);

        switch (msg[0])
        {
        case AC_OPEN:
            if (msg[3] == menu_id)
                OpenAccessoryWindow();
            break;
        case AC_CLOSE:
            if (msg[3] == menu_id)
            {
                v_clsvwk(ws.handle);
            }
            break;
        }
    }
}
The Environment String

One AES environment string exists in the system. This environment string is the one initially allocated for the AES by GEMDOS. The AES environment string should not be confused with GEMDOS environment strings. Each GEMDOS process receives its own environment string when launched. This string may have been purposely altered (or omitted) by its parent.

The AES environment string is a collection of variables which the AES (and other processes) may use as global system variables. Environment data may be set by a CPX designed to configure the environment, in the user’s GEM.CNF file, or by an application.

In actuality, the environment string is actually one or many string entries separated by NULL bytes with the full list being terminated by a double NULL byte. Examples of environment string entries include:

```
PATH=C:\;D:\;E:\BIN\n
TEMP=C:\n
AE_SREDRAW=0
```

The environment variable name is followed by an equal sign which is followed by the variable data. Multiple arguments (such as path names) may be separated by semicolons or commas.\(^3\)

The AES call `shel_envrn()` may be used to search for an environment variable and new modes of `shel_write()` (after AES version 4.0) may be used to alter environment variables or copy the entire environment string.

Most versions of the AES contain a bug which causes the ‘PATH’ environment variable to be set incorrectly upon bootup to ‘PATH=\[nul\]A:\[nul\][nul]’. If an environment string like this is found it may be safely reset or simply ignored.

The Event Dispatcher

Most GEM applications and all desk accessories rely on one of the AES event processing calls to direct program flow. After program initialization, an application enters a message loop which waits for and reacts to messages sent by the AES. Five basic types of events are generated by the AES and each can be read by a specialized event library call as follows:

<table>
<thead>
<tr>
<th>Event Type</th>
<th>AES Function</th>
</tr>
</thead>
<tbody>
<tr>
<td>Message</td>
<td><code>evnt_mesag()</code></td>
</tr>
<tr>
<td>Mouse Button</td>
<td><code>evnt_button()</code></td>
</tr>
<tr>
<td>Keyboard</td>
<td><code>evnt_keybd()</code></td>
</tr>
<tr>
<td>Timer</td>
<td><code>evnt_timer()</code></td>
</tr>
</tbody>
</table>

\(^3\)The AES only began recognizing commas as valid argument separators (for the PATH environment variable) as of AES version 1.4.
In addition to these five basic calls, the AES offers one multi-purpose call which waits for any combination of the above events called `evnt_multi()`. The `evnt_multi()` call is often the most important function call in any GEM application. A typical message loop follows:

```c
#include <AES.H>

void MessageLoop( void )
{
    short mx, my; /* Mouse Position */
    short mb, mc; /* Mouse button/# clicks */
    short ks, kc; /* Key state/code */
    short quit; /* Exit flag */
    short msg[8]; /* Message buffer */
    short events; /* What events are valid? */

    /* Mask for all events */
#define ALL_EVENTS (MU_MESAG|MU_BUTTON|MU_KEYBD|MU_TIMER|MU_M1|MU_M2)

    quit = FALSE;
    while(!quit)
    {
        events = evnt_multi( ALL_EVENTS,
            2, 1, 1, /* Single/double clicks */
            0, 0, 0, 128, 128, /* M1 event */
            1, 0, 0, 128, 128, /* M2 event */
            msg, /* Pointer to msg */
            1000, 0, /* MU_TIMER every 1 sec. */
            &mx, &my, &ks, &kc,
            &mc );

        if( events & MU_MESAG )
        {
            switch( msg[0] ) /* msg[0] is message type */
            {
                case MN_SELECTED: /* msg[0] is message type */
                    HandleMenuClick( msg );
                    break;
                case WM_CLOSED:
                    CloseWindow( msg[3] );
                    break;
                /*
                 * more message events...
                 */
            }
        }

        if( events & MU_BUTTON )
        {
            /*
             * Handle mouse button event.
             */
        }

        if( events & MU_KEYBD )
        {
            /*
             * Handle keyboard event.
             */
        }
    }
}
```

**The Atari Compendium**
When an event library function is called, the program is effectively halted until a message which is being waited for becomes available. Not all applications will require all events so the above code may be considered flexible.

Message Events
Each standard GEM message event (MU_MESAG) uses some or all of an 8 WORD message buffer. Each entry in this buffer is assigned as follows:

<table>
<thead>
<tr>
<th>msg[x]</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>Message type.</td>
</tr>
<tr>
<td>1</td>
<td>The application identifier of the process sending the message.</td>
</tr>
<tr>
<td>2</td>
<td>The length of the message beyond 16 bytes (in bytes).</td>
</tr>
<tr>
<td></td>
<td>For all standard GEM messages, this value is 0.</td>
</tr>
<tr>
<td>3</td>
<td>Depends on message.</td>
</tr>
<tr>
<td>4</td>
<td>Depends on message.</td>
</tr>
<tr>
<td>5</td>
<td>Depends on message.</td>
</tr>
<tr>
<td>6</td>
<td>Depends on message.</td>
</tr>
<tr>
<td>7</td>
<td>Depends on message.</td>
</tr>
</tbody>
</table>

The entry for evnt_mesag() later in this chapter has a comprehensive list of all system messages and the action that should be taken when they are received.
User-Defined Message Events
Applications may write customized messages to other applications (or themselves) using `appl_write()`. The structure of the message buffer should remain the same as shown above. If more than the standard eight WORDs of data are sent, however, `appl_read()` must be used to read the additional bytes. It is recommended that user-defined messages be set to a multiple of 8 bytes.

You can use this method to send your own application standard messages by filling in the message buffer appropriately and using `appl_write()`. This method is often used to force redraw or window events.

Mouse Button Events
When a mouse button (MU_BUTTON) event happens, the `evnt_button()` or `evnt_multi()` call is returned with the mouse coordinates, the number of clicks that occurred, and the keyboard shift state.

Keyboard Events
Keyboard events (MU_KEYBD) are generated whenever a key is struck. The IKBD scan code (see Appendix F: IKBD Scan Codes) and current key shift state are returned by either `evnt_keybd()` or `evnt_multi()`. If your application is designed to run on machines in other countries, you might consider translating the scan codes using the tables returned by the XBIOS call `Keytbl()`.

Timer Events
`evnt_timer()` or `evnt_multi( MU_TIMER, ... )` can be used to request a timer event(s) be scheduled in a certain number of milliseconds. The time between the actual function call and the event may, however, be greater than the time specified.

Mouse Rectangle Events
Mouse rectangle events (MU_M1 and/or MU_M2) are generated by `evnt_mouse()` and `evnt_multi()` when the mouse pointer enters or leaves (depending on how you program it) a specified rectangle.
Resources

GEM resources consist of object trees, strings, and bitmaps used by an application. They encapsulate the user interface and make internationalization easier by placing all program strings in a single file. Resources are generally created using a Resource Construction Set (RCS) and saved to a .RSC file (see Appendix C: Native File Formats) which is loaded by \texttt{rsrc\_load()} at program initialization time.

Resources may also be embedded as data structures in source code (some utility programs convert .RSC files to source code). Desk accessories often do this to avoid complications they have in loading .RSC files.

Resources contain pointers and coordinates which must be fixed up before being used. \texttt{rsrc\_load()} does this automatically, however if you use an embedded resource you must use \texttt{rsrc\_rcfix()} if available or \texttt{rsrc\_obfix()} on each object in each object tree to convert the initial character coordinates of to screen coordinates. This allows resources designed on screens with different aspect ratios and system fonts to appear the same. In any case, you should test your resources on several different screens, especially screen resolutions with different aspect ratios such as ST Medium and ST High.

Once a resource is loaded use \texttt{rsrc\_gaddr()} to obtain pointers to individual object trees which can then be manipulated directly or with the AES Object Library. Replacing resources after they’re loaded is accomplished with \texttt{rsrc\_saddr()}.

Objects

Objects can be boxes, buttons, text, images, and more. An object tree is an array of \texttt{OBJECT} structures linked to form a structured relationship to each other. The \texttt{OBJECT} structure format is as follows:

\begin{verbatim}
typedef struct object
{
    WORD    ob_next;
    WORD    ob_head;
    WORD    ob_tail;
    UWORD   ob_type;
    UWORD   ob_flags;
    UWORD   ob_state;
    VVOIDP  ob_spec;
    WORD    ob_x;
    WORD    ob_y;
    WORD    ob_width;
    WORD    ob_height;
} OBJECT;
\end{verbatim}

Normally \texttt{OBJECT}s are loaded in an application resource file but it is possible to create and manipulate them on-the-fly using the \texttt{objc\_add()}, \texttt{objc\_delete()}, and \texttt{objc\_order()} commands.
The first object in an OBJECT tree is called the ROOT object (OBJECT #0). It’s coordinates are relative to the upper-left hand corner of the screen.

The ROOT object can have any number of children and each child can have children of their own. In each case, the OBJECT’s coordinates, \(ob_x\), \(ob_y\), \(ob_width\), and \(ob_height\) are relative to that of its parent. The AES call \texttt{objc_offset()} can, however, be used to determine the exact screen coordinates of a child object. \texttt{objc_find()} is used to determine the object at a given screen coordinate.

The \(ob\_next\), \(ob\_head\), and \(ob\_tail\) fields determine this relationship between parent OBJECTs and child OBJECTs. The following alert box is an example of an OBJECT tree:

![Please Select an Output Device: Screen Printer OK](image-url)
The tree structure this object has can be represented as follows:

```
[ROOT OBJECT]
    
Object #0 - BOX
    ob_head = 1
    ob_tail = 5
    ob_next = -1

Object #1 - TEXT
    ob_head = -1
    ob_tail = -1
    ob_next = 2

Object #2 - BOX
    ob_head = 3
    ob_tail = 4
    ob_next = 3

Object #3 - BOXTEXT
    ob_head = 1
    ob_tail = 5
    ob_next = -4

Object #4 - BOXTEXT
    ob_head = 1
    ob_tail = 5
    ob_next = 3

Object #5 - BUTTON
    ob_head = -1
    ob_tail = -1
    ob_next = 0
```

The exact usage of `ob_head`, `ob_next`, and `ob_tail` are as follows:

<table>
<thead>
<tr>
<th>Element</th>
<th>Usage</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>ob_head</code></td>
<td>This member gives the exact index from the first object in the OBJECT tree to the first child of the current object. If the object has no children then this value should be -1.</td>
</tr>
<tr>
<td><code>ob_tail</code></td>
<td>This member gives the exact index from the first object in the OBJECT tree to the last child of the current object. If the object has no children then this value should be -1.</td>
</tr>
<tr>
<td><code>ob_next</code></td>
<td>This member gives the exact index from the first object in the OBJECT tree to the next child at the same level. The ROOT object should be set to -1. The last child at any given nesting level should be set to the index of its parent.</td>
</tr>
</tbody>
</table>

The low byte of the `ob_type` field specifies the object type as follows:

<table>
<thead>
<tr>
<th>Name</th>
<th><code>ob_type &amp; 0xFF</code></th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>G_BOX</td>
<td>20</td>
<td>Box</td>
</tr>
<tr>
<td>G_TEXT</td>
<td>21</td>
<td>Formatted Text</td>
</tr>
<tr>
<td>G_BOXTEXT</td>
<td>22</td>
<td>Formatted Text in a Box</td>
</tr>
<tr>
<td>G_IMAGE</td>
<td>23</td>
<td>Monochrome Image</td>
</tr>
<tr>
<td>G_PROGDEF</td>
<td>24</td>
<td>Programmer-Defined Object.</td>
</tr>
</tbody>
</table>
Object Flags

The `ob_flags` field of the `OBJECT` structure is a bitmask of different flags that can be applied to any object as follows:

<table>
<thead>
<tr>
<th>Name</th>
<th>Bit(s)</th>
<th>Mask</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>SELECTABLE</td>
<td>0</td>
<td>0x0001</td>
<td>Object’s selected state may be toggled by clicking on it with the mouse.</td>
</tr>
<tr>
<td>DEFAULT</td>
<td>1</td>
<td>0x0002</td>
<td>An <code>EXIT</code> object with this bit set will have a thicker outline and be triggered when the user presses <code>RETURN</code>.</td>
</tr>
<tr>
<td>EXIT</td>
<td>2</td>
<td>0x0004</td>
<td>Clicking on this <code>OBJECT</code> and releasing the mouse button while still over it will cause the dialog to exit.</td>
</tr>
<tr>
<td>EDITABLE</td>
<td>3</td>
<td>0x0008</td>
<td>Set for <code>FTEXT</code> and <code>FBOXTEXT</code> objects to indicate that they may receive edit focus.</td>
</tr>
<tr>
<td>RBUTTON</td>
<td>4</td>
<td>0x0010</td>
<td>This object is one of a group of radio buttons. Clicking on it will deselect any selected objects at the same tree level that also have the <code>RBUTTON</code> flag set. Likewise, it will be deselected automatically when any other object is selected.</td>
</tr>
<tr>
<td>LASTOB</td>
<td>5</td>
<td>0x0020</td>
<td>This flag signals to the <code>AES</code> that the current <code>OBJECT</code> is the last in the object tree. (Required!)</td>
</tr>
<tr>
<td>TOUCHEXIT</td>
<td>6</td>
<td>0x0040</td>
<td>Setting this flag causes the <code>OBJECT</code> to return an exit state immediately after being clicked on with the mouse.</td>
</tr>
<tr>
<td>HIDETREE</td>
<td>7</td>
<td>0x0080</td>
<td>This <code>OBJECT</code> and all of its children will not be drawn.</td>
</tr>
<tr>
<td>INDIRECT</td>
<td>8</td>
<td>0x0100</td>
<td>This flag cause the <code>ob_spec</code> field to be interpreted as a pointer to the <code>ob_spec</code> value rather than the value itself.</td>
</tr>
<tr>
<td>FL3DIND</td>
<td>9</td>
<td>0x0200</td>
<td>Setting this flag causes the <code>OBJECT</code> to be drawn as a 3D indicator. This is appropriate for radio and toggle buttons. This flag is only recognized as of <code>AES</code> version 3.4.</td>
</tr>
<tr>
<td>FL3DACT</td>
<td>10</td>
<td>0x0400</td>
<td>Setting this flag causes the <code>OBJECT</code> to be drawn as a 3D activator. This is appropriate for <code>EXIT</code> buttons. This flag is only recognized as of <code>AES</code> version 3.4.</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Name</th>
<th>Bit(s)</th>
<th>Mask</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>G_IBOX</td>
<td>25</td>
<td></td>
<td>Invisible Box</td>
</tr>
<tr>
<td>G_BUTTON</td>
<td>26</td>
<td></td>
<td>Push Button w/String</td>
</tr>
<tr>
<td>G_BOXCHAR</td>
<td>27</td>
<td></td>
<td>Character in a Box</td>
</tr>
<tr>
<td>G_STRING</td>
<td>28</td>
<td></td>
<td>Unformatted Text</td>
</tr>
<tr>
<td>G_FTEXT</td>
<td>29</td>
<td></td>
<td>Editable Formatted Text</td>
</tr>
<tr>
<td>G_FBOXTEXT</td>
<td>30</td>
<td></td>
<td>Editable Formatted Text in a Box</td>
</tr>
<tr>
<td>G_ICON</td>
<td>31</td>
<td></td>
<td>Monochrome Icon</td>
</tr>
<tr>
<td>G_TITLE</td>
<td>32</td>
<td></td>
<td>Menu Title</td>
</tr>
<tr>
<td>G_CICON</td>
<td>33</td>
<td></td>
<td>Color Icon (Available as of <code>AES</code> v3.3)</td>
</tr>
</tbody>
</table>
Object States

The *ob_state* field determines the display state of the *OBJECT* as follows:

<table>
<thead>
<tr>
<th>Name</th>
<th>Bit</th>
<th>Mask</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>SELECTED</td>
<td>0</td>
<td>0x0001</td>
<td>The object is selected. An object with this bit set will be drawn in inverse video except for <strong>G_CICON</strong> which will use its 'selected' image.</td>
</tr>
<tr>
<td>CROSSED</td>
<td>1</td>
<td>0x0002</td>
<td>An <strong>OBJECT</strong> with this bit set will be drawn over with a white cross (this state can only usually be seen over a colored or <strong>SELECTED</strong> object).</td>
</tr>
<tr>
<td>CHECKED</td>
<td>2</td>
<td>0x0004</td>
<td>An <strong>OBJECT</strong> with this bit set will be displayed with a checkmark in its upper-left corner.</td>
</tr>
<tr>
<td>DISABLED</td>
<td>3</td>
<td>0x0008</td>
<td>An <strong>OBJECT</strong> with this bit set will ignore user input. Text objects with this bit set will draw in a dithered pattern.</td>
</tr>
<tr>
<td>OUTLINED</td>
<td>4</td>
<td>0x0010</td>
<td><strong>G_BOX</strong>, <strong>G_IBOX</strong>, <strong>G_BOXTEXT</strong>, <strong>G_FBOXTEXT</strong>, and <strong>G_BOXCHAR</strong> <strong>OBJECTs</strong> with this bit set will be drawn with a double border.</td>
</tr>
<tr>
<td>SHADOWED</td>
<td>5</td>
<td>0x0020</td>
<td><strong>G_BOX</strong>, <strong>G_IBOX</strong>, <strong>G_BOXTEXT</strong>, <strong>G_FBOXTEXT</strong>, and <strong>G_BOXCHAR</strong> <strong>OBJECTs</strong> will be drawn with a shadow.</td>
</tr>
</tbody>
</table>

The AES supports the **objc_change()** call which can be used to change the state of an object and (optionally) redraw it.
The Object-Specific Field

The `ob_spec` field contains different data depending on the object type as indicated in the table below:

<table>
<thead>
<tr>
<th>Object</th>
<th>Contents of <code>ob_spec</code></th>
</tr>
</thead>
<tbody>
<tr>
<td>G_BOX</td>
<td>The low 16 bits contain a <code>WORD</code> containing color information for the <code>OBJECT</code>. Bits 23-16 contain a signed <code>BYTE</code> representing the border thickness of the box.</td>
</tr>
<tr>
<td>G_TEXT</td>
<td>The <code>ob_spec</code> field contains a pointer to a <code>TEDINFO</code> structure.</td>
</tr>
<tr>
<td>G_BOXTXT</td>
<td>The <code>ob_spec</code> field contains a pointer to a <code>TEDINFO</code> structure.</td>
</tr>
<tr>
<td>G_IMAGE</td>
<td>The <code>ob_spec</code> field points to a <code>BITBLK</code> structure.</td>
</tr>
<tr>
<td>G_PROGDEF</td>
<td>The <code>ob_spec</code> field points to a <code>APPLBLK</code> structure.</td>
</tr>
<tr>
<td>G_IBOX</td>
<td>The low 16 bits contain a <code>WORD</code> containing color information for the <code>OBJECT</code>. Bits 23-16 contain a signed <code>BYTE</code> representing the border thickness of the box.</td>
</tr>
<tr>
<td>G_BUTTON</td>
<td>The <code>ob_spec</code> field contains a pointer to the text to be contained in the button.</td>
</tr>
<tr>
<td>G_BOXCHAR</td>
<td>The low 16 bits contain a <code>WORD</code> containing color information for the <code>OBJECT</code>. Bits 23-16 contain a signed <code>BYTE</code> representing the border thickness of the box. Bits 31-24 contain the ASCII value of the character to display.</td>
</tr>
<tr>
<td>G_STRING</td>
<td>The <code>ob_spec</code> field contains a pointer to the text to be displayed.</td>
</tr>
<tr>
<td>G_FTEXT</td>
<td>The <code>ob_spec</code> field contains a pointer to a <code>TEDINFO</code> structure.</td>
</tr>
<tr>
<td>G_FBOXTXT</td>
<td>The <code>ob_spec</code> field contains a pointer to a <code>TEDINFO</code> structure.</td>
</tr>
<tr>
<td>G_ICON</td>
<td>The <code>ob_spec</code> field contains a pointer to an <code>ICONBLK</code> structure.</td>
</tr>
<tr>
<td>G_TITLE</td>
<td>The <code>ob_spec</code> field contains a pointer to the text to be used for the title.</td>
</tr>
<tr>
<td>G_CICON</td>
<td>The <code>ob_spec</code> field contains a pointer to a <code>CICONBLK</code> structure.</td>
</tr>
</tbody>
</table>

Object-Specific Structures

Almost all objects reference a `WORD` containing the object color as defined below (note the definition below may need to be altered depending upon the bit ordering of your compiler).

```c
typedef struct objc_colorword
{
    UWORD borderc : 4; /* Bits 15-12 contain the border color */
    UWORD textc : 4; /* Bits 11-8 contain the text color */
    UWORD opaque : 1; /* Bit 7 is 1 if opaque or 0 if transparent */
    UWORD pattern : 3; /* Bits 6-4 contain the fill pattern index */
    UWORD fillc : 4; /* Bits 3-0 contain the fill color */
} OBJC_COLORWORD;
```

Available colors for fill patterns, text, and borders are listed below:
### TEDINFO

G_TEXT, G_BOXTEXT, G_FTEXT, and G_FBOXTEXT objects all reference a TEDINFO structure in their `ob_spec` field. The TEDINFO structure is defined below:

```c
typedef struct text_edinfo
{
  char * te_ptext;
  char * te_ptmplt;
  char * te_pvalid;
  WORD  te_font;
  WORD  te_fontid;
  WORD  te_just;
  WORD  te_color;
  WORD  te_fontsize;
  WORD  te_thickness;
  WORD  te_txtlen;
  WORD  te_tmplen;
} TEDINFO;
```

The three character pointer point to text strings required for G_FTEXT and G_FBOXTEXT objects. `te_ptext` points to the actual text to be displayed and is the only field used by all text objects. `te_ptmplt` points to the text template for editable fields. For each character that the user can enter, the text string should contain a tilde character (ASCII 126). Other characters are displayed but cannot be overwritten by the user. `te_pvalid` contains validation characters for each character the user may enter. The current acceptable validation characters are:

<table>
<thead>
<tr>
<th>Character</th>
<th>Allows</th>
</tr>
</thead>
<tbody>
<tr>
<td>9</td>
<td>Digits 0-9</td>
</tr>
<tr>
<td>A</td>
<td>Uppercase letters A-Z plus SPACE</td>
</tr>
<tr>
<td>a</td>
<td>Upper and lowercase letters plus SPACE</td>
</tr>
</tbody>
</table>
As an example the following diagram shows the correct text, template, and validation strings for obtaining a GEMDOS filename from the user.

<table>
<thead>
<tr>
<th>String</th>
<th>Contents</th>
</tr>
</thead>
<tbody>
<tr>
<td>te_ptext</td>
<td>‘\0’ (NULL char)</td>
</tr>
<tr>
<td>te_ptmplt</td>
<td>___________</td>
</tr>
<tr>
<td>te_pvalid</td>
<td>FFFFFFFFFFFF</td>
</tr>
</tbody>
</table>

*te_font* may be set to any of the following values:

<table>
<thead>
<tr>
<th>Name</th>
<th>te_font</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>GDOS_PROP</td>
<td>0</td>
<td>Use a SpeedoGDOS font (valid only with an AES version of at least 4.0 and SpeedoGDOS installed).</td>
</tr>
<tr>
<td>GDOS_MONO</td>
<td>1</td>
<td>Use a SpeedoGDOS font (valid only with an AES version of at least 4.1 and SpeedoGDOS installed) and force monospaced output.</td>
</tr>
<tr>
<td>GDOS_BITM</td>
<td>2</td>
<td>Use a GDOS bitmap font (valid only with an AES version of at least 4.1 and SpeedoGDOS installed).</td>
</tr>
<tr>
<td>IBM</td>
<td>3</td>
<td>Use the standard monospaced system font.</td>
</tr>
<tr>
<td>SMALL</td>
<td>5</td>
<td>Use the small monospaced system font.</td>
</tr>
</tbody>
</table>

When using a value of GDOS_PROP, GDOS_MONO, or GDOS_BITM, *te_fontsize* specifies the font size in points and *te_fontid* specifies the SpeedoGDOS font identification number. Selecting the IBM or SMALL font will cause *te_fontsize* and *te_fontid* to be ignored.

*te_just* sets the justification of the text output as follows:

<table>
<thead>
<tr>
<th>Name</th>
<th>te_just</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>TE_LEFT</td>
<td>0</td>
<td>Left Justify</td>
</tr>
<tr>
<td>TE_RIGHT</td>
<td>1</td>
<td>Right Justify</td>
</tr>
<tr>
<td>TE_CNTR</td>
<td>2</td>
<td>Center</td>
</tr>
</tbody>
</table>
The thickness sets the border thickness (positive and negative values are acceptable) of the G_BOXTEXT or G_FBOXTEXT object. te_txtlen and te_tmplen should be set to the length of the starting text and template length respectively.

BITBLK
G_IMAGE objects contain a pointer to a BITBLK structure in their ob_spec field. The BITBLK structure is defined as follows:

```
typedef struct bit_block
{
    WORD * bi_pdata;
    WORD  bi_wb;
    WORD  bi_hl;
    WORD  bi_x;
    WORD  bi_y;
    WORD  bi_color;
} BITBLK;
```

bi_pdata should point to a monochrome bit image. bi_wb specifies the width (in bytes) of the image. All BITBLK images must be a multiple of 16 pixels wide therefore this value must be even.

bi_hl specifies the height of the image in scan lines (rows). bi_x and bi_y are used as offsets into bi_pdata. Any data occurring before these coordinates will be ignored. bi_color is a standard color WORD where the fill color specifies the color in which the image will be rendered.

ICONBLK
The ob_spec field of G_ICON objects point to an ICONBLK structure as defined below:

```
typedef struct icon_block
{
    WORD *    ib_pmask;
    WORD *    ib_pdata;
    char *    ib_ptext;
    WORD      ib_char;
    WORD      ib_xchar;
    WORD      ib_ychar;
    WORD      ib_xicon;
    WORD      ib_yicon;
    WORD      ib_wicon;
    WORD      ib_hicon;
    WORD      ib_xtext;
    WORD      ib_ytext;
    WORD      ib_wtext;
    WORD      ib_htext;
} ICONBLK;
```

ib_pmask and ib_pdata are pointers to the monochrome mask and image data respectively. ib_ptext is a string pointer to the icon text. ib_char defines the icon character (used for drive icons) and the icon foreground and background color as follows:
<table>
<thead>
<tr>
<th>ib_char</th>
<th>Bits 15-12</th>
<th>Bits 11-8</th>
<th>Bits 7-0</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Icon Foreground Color</td>
<td>Icon Background Color</td>
<td>ASCII Character (or 0 for no character)</td>
</tr>
</tbody>
</table>

`ib_xchar` and `ib_ychar` specify the location of the icon character relative to `ib_xicon` and `ib_yicon`. `ib_xicon` and `ib_yicon` specify the location of the icon relative to the `ob_x` and `ob_y` of the object. `ib_wicon` and `ib_hicon` specify the width and height of the icon in pixels. As with images, icons must be a multiple of 16 pixels in width.

`ib_xtext` and `ib_ytext` specify the location of the text string relative to the `ob_x` and `ob_y` of the object. `ib_wtext` and `ib_htext` specify the width and height of the icon text area.

**CICONBLK**

The `G_CICON` object (available as of AES version 3.3) defines its `ob_spec` field to be a pointer to a `CICONBLK` structure as defined below:

```c
typedef struct cicon_blk
{
    ICONBLK monoblk;
    CICON * mainlist;
} CICONBLK;
```

`monoblk` contains a monochrome icon which is rendered if a color icon matching the display parameters cannot be found. In addition, the icon text, character, size, and positioning data from the monochrome icon are always used for the color one. `mainlist` points to the first `CICON` structure in a linked list of color icons for different resolutions. `CICON` is defined as follows:

```c
typedef struct cicon_data
{
    WORD num_planes;
    WORD * col_data;
    WORD * col_mask;
    WORD * sel_data;
    WORD * sel_mask;
    struct cicon_data * next_res;
} CICON;
```

`num_planes` indicates the number of bit planes this color icon contains. `col_data` and `col_mask` point to the icon data and mask for the unselected icon respectively. Likewise, `sel_data` and `sel_mask` point to the icon data and mask for the selected icon. `next_res` points to the next color icon definition or `NULL` if no more are available. Bitmap data pointed to by these variables should be in VDI device-dependent format (they are stored as device-independent images in a .RSC file).

The AES searches the `CICONBLK` object for a color icon that has the same number of planes in the display. If none is found, the AES simply uses the monochrome icon.
**APPLBLK**

**G_PROGDEF** objects allow programmers to define custom objects and link them transparently in the resource. The \( ob\_spec \) field of **G_PROGDEF** objects contains a pointer to an **APPLBLK** as defined below:

```c
typedef struct appl_blk
{
    WORD (*ab_code)(PARMBLK *);
    LONG ab_parm;
} APPLBLK;
```

\( ab\_code \) is a pointer to a user-defined routine which will draw the object. The routine will be passed a pointer to a **PARMBLK** structure containing the information it needs to render the object. The routine must be defined with stack checking off and expect to be passed its parameter on the stack. \( ab\_parm \) is a user-defined value which is copied into the **PARMBLK** structure as defined below:

```c
typedef struct parm_blk
{
    OBJECT *tree;
    WORD pb_obj;
    WORD pb_prevstate;
    WORD pb_currstate;
    WORD pb_x;
    WORD pb_y;
    WORD pb_w;
    WORD pb_h;
    WORD pb_xc;
    WORD pb_yc;
    WORD pb_wc;
    WORD pb_hc;
    LONG pb_parm;
} PARMBLK;
```

\( tree \) points to the **OBJECT** tree of the object being drawn. The object is located at index \( pb\_obj \).

The routine is passed the old \( ob\_state \) of the object in \( pb\_prevstate \) and the new \( ob\_state \) of the object in \( pb\_currstate \). If \( pb\_prevstate \) and \( pb\_currstate \) is equal then the object should be drawn completely, otherwise only the drawing necessary to redraw the object from \( pb\_prevstate \) to \( pb\_currstate \) are necessary.

\( pb\_x, pb\_y, pb\_w, \) and \( pb\_h \) give the screen coordinates of the object. \( pb\_xc, pb\_yc, pb\_wc, \) and \( pb\_hc \) give the rectangle to clip to. \( pb\_parm \) contains a copy of the \( ap\_parm \) value in the **APPLBLK** structure.

The custom routine should return a **WORD** containing any remaining \( ob\_state \) bits you wish the AES to draw over your custom object.
Because the drawing routing will be called from the context of the AES, using the stack heavily or defining many local variables is not recommended.

## Dialogs

Dialog boxes are modal forms of user input. This means that no other interaction can occur between the user and applications until the requirements of the dialog have been met and it is exited. A normal dialog box consists of an OBJECT tree with a BOX as its root object and any number of other controls that accept user input. Both alert boxes and the file selector are examples of AES provided dialog boxes.

The AES form_do() function is the simplest method of using a dialog box. Simply construct an OBJECT tree with at least one EXIT or TOUCHEXIT object and call form_do(). All interaction with the dialog like editable fields, radio buttons, and selectable objects will be maintained by the AES until the user strikes an EXIT or TOUCHEXIT object. The proper method for displaying a dialog box is shown in the example below:

```c
WORD do_dialog( OBJECT *tree, WORD first_edit )
{
    GRECT g;
    WORD ret;

    /* Reserve screen/mouse button */
    wind_update( BEG_UPDATE );
    wind_update( BEG_MCTRL );

    /* Center dialog on screen and put clipping rectangle in g */
    form_center( tree, &g.g_x, &g.g_y, &g.g_w, &g.g_h );

    /* Reserve screen space and draw growing box */
    form_dial( FMD_START, 0, 0, 0, 0, g.g_x, g.g_y, g.g_w, g.g_h );
    form_dial( FMD_GROW, g.g_x + g.g_w/2, g.g_y + g.g_h/2, 0, 0, g.g_x, g.g_y, g.g_w, g.g_h );

    /* Draw the dialog box */
    objc_draw( tree, ROOT, MAX_DEPTH, g.g_x, g.g_y, g.g_w, g.g_h );

    /* Handle dialog */
    ret = form_do( tree, first_edit );

    /* Deselect EXIT button */
    tree[ret].ob_state &= ~SELECTED;

    /* Draw shrinking box and release screen area */
    form_dial( FMD_SHRINK, g.g_x + g.g_w/2, g.g_y + g.g_h/2, 0, 0, g.g_x, g.g_y, g.g_w, g.g_h );
    form_dial( FMD_FINISH, 0, 0, 0, 0, g.g_x, g.g_y, g.g_w, g.g_h );

    /* Release screen/mouse control. */
    wind_update( END_MCTRL );
    wind_update( END_UPDATE );
}
```
You may wish to create your own specialized dialog handling routines or place dialog boxes in windows to create modeless input. This can be accomplished by using the `form_button()`, `form_keybd()`, and `objc_edit()` AES calls. Specific information about these calls may be found in the `Function Reference`.

GEM also provides two generic dialog boxes through the `form_alert()` and `form_error()` calls. `form_alert()` displays an alert dialog with a choice between icons and user-defined text and buttons. `form_error()` displays an alert based on predefined system error codes.

**Menus**

Most GEM applications use a menu bar to allow the user to navigate through program options. In addition, newer versions of the AES now allow popup menus and drop-down list boxes (a special form of a popup menu). Menus are simply specially designed OBJECT trees activated using special AES calls.

**The Menu Bar**

The menu bar is a special OBJECT which is usually registered in the beginning stages of a GEM program which contains choices which the user may select to trigger a special menu event (`MN_SELECTED`) to be sent to the application’s message loop. Normally, you will use a resource construction set to create a menu but if you are designing an RCS or must create a menu bar by hand, the format for the OBJECT structure of a GEM menu bar is shown below:

```
+-------------------+-------------------+
| ROOT (G_IBOX)     | DROPDOWNS (G_IBOX) |
|                   | BAR (G_BOX)       |
|                   | ACTIVE (G_IBOX)   |
|                   | G_TITLE           |
|                   | G_TITLE           |
|                   | G_TITLE           |
|                   | G_STRING          |
|                   | G_STRING          |
|                   | G_STRING          |
+-------------------+-------------------+
```

The `ROOT` object is a `G_IBOX` and should be set to the same width and height of the screen. It has two children, the `BAR` object and the `DROPDOWNS` object.
The **BAR** object is a **G_BOX** which should be the width of the screen and the height of the system font plus two pixels for a border line. The **DROPDOWNS** object is a **G_IBOX** and should be of a size large enough to encompass all of the drop-down menu boxes.

The **BAR** object has one child, the **ACTIVE** object, it should be the width of the screen and the height of the system font. It has as many **G_TITLE** children as there are menu titles.

The **DROPDOWNS** object has the same number of **G_BOX** child objects as the **ACTIVE** object has **G_TITLE** children. Each box must be high enough to support the number of **G_STRING** menu items and wide enough to support the longest item. Each **G_BOX** must be aligned so that it falls underneath its corresponding **G_TITLE**. In addition, each **G_STRING** menu item should be the same length as its parent **G_BOX** object.

Each **G_STRING** menu item should be preceded by two spaces. Each **G_TITLE** should be preceded and followed by one space. The first **G_BOX** object should appear under a **G_TITLE** object named ‘Desk’ and should contain eight children. The first child **G_STRING** is application defined (it usually leads to the ‘About...’ program credits), the second item should be a disabled separator (‘-----------’ line). The next six items are dummy objects used by the **AES** to display program and desk accessory titles.

**Utilizing a Menu Bar**

Menu bars can be displayed and their handling initiated by calling `menu_bar()`. In addition, using this command, a menu bar may be turned off or replaced with another menu bar at any time.

Individual menu items may be altered with three **AES** calls. `menu_icheck()` sets or removes a checkmark from in front of menu items. `menu_ienable()` enables or disables a menu item. `menu_itext()` alters the text of a menu item. After receiving a message indicating that a menu item has been clicked, perform the action appropriate to the menu item and then call `menu_tnormal()` to return the menu title text to normal video.

**Hierarchical Menus**

**AES** versions 3.3 and above support hierarchical submenus. When a submenu is attached to a regular menu item, a right arrow is appended to the end of the menu item text and a submenu is displayed whenever the mouse is positioned over the menu item. The user may select submenu items which cause an extended version of the **MN_SELECTED** message to be delivered (containing the menu object tree).

Up to 64 submenu attachments may be in effect at any time per process. Attaching a single submenu to more than one menu item counts as only one attachment.

Submenus should be **G_BOX** objects with as many **G_STRING** (or other) child objects as necessary. One or several submenus may be contained in a single **OBJECT** tree. If the submenu’s scroll flag is set, scroll arrows will appear and the menu will be scrollable if it
contains more items than the currently set system scroll value. Submenus containing user-defined objects should not have their scroll flag set.

Submenus are attached and removed with the `menu_attach()` call. A serious bug exists in AES versions lower than 4.0 which causes `menu_attach()` to crash the system if you use it to remove or inquire the state of an existing submenu. This means that submenus may only be removed in AES versions 4.0 and above. Submenus may be nested to up to four levels though only one level is recommended.

Submenus may not be attached to menu items in the left-most ‘Desk’ menu. Individual submenu items may be aligned with the parent object by using `menu_istart()`.

**Popup Menus**

AES versions 3.3 and above support popup menus. Popup menus share the same `OBJECT` structure as hierarchical menus but are never attached to a parent menu item. They may be displayed anywhere on the screen and are often called in response to selecting a special dialog item (see Chapter 11: *GEM User Interface Guidelines*). Popup menus are displayed with the AES call `menu_popup()`.

**Menu Settings**

The AES call `menu_settings()` may be used to adjust certain global defaults regarding the appearance and timing delays of submenus and popup menus. Because this call affects all system applications it should only be utilized by a system configuration utility and not by individual applications.

**Drop-Down List Boxes**

AES versions 4.0 and later support a special type of popup menu called a drop-down list box. Setting the menu scroll flag to a value of -1 will cause a popup menu to be displayed as a drop-down list instead.

A drop-down list reveals up to eight items from a multiple item list to the user. A slider bar is displayed next to the list and is automatically handled during the `menu_popup()` call. Several considerations must be taken when using a drop-down list box:

- Drop-down lists may only contain `G_STRING` objects.
- If you want to force the AES to always draw scroll bars for the list box, the `OBJECT` tree must contain at least eight `G_STRING` objects. If less than that number of items exist, pad the remaining items with blanks and set the object’s `DISABLED` flag.
- As long as the `OBJECT` tree has at least eight `G_STRING` objects, it should not be padded with any additional objects since the size of the slider is based on the number of objects.
The Menu Buffer
A special memory area is allocated by the AES so that it may reserve the screen area underneath displayed menus. A pointer to this memory and its length may be obtained by calling `wind_get(WF_SCREEN, ...)`. Menu buffer memory may be used as a temporary holding arena for applications as long as the following rules are maintained:

- The application must not use a menu bar or it must be locked with `wind_update(BEG_UPDATE)`.
- Access to the menu buffer in a multitasking environment is not controlled so information stored by one application may be overwritten by another. It is therefore recommended that the menu buffer should not be used under MultiTOS.

Windows

GEM applications usually maintain most user-interaction in windows. Windows are workspaces created with `wind_create()` with any of several predefined gadgets (controls) illustrated in the diagram and table below:
<table>
<thead>
<tr>
<th>Name</th>
<th>Mask</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>NAME</td>
<td>0x0001</td>
<td>Using this mask will cause the AES to display the window with a title bar containing a name that the application should set with <code>wind_set(WF_NAME, ...)</code>.</td>
</tr>
<tr>
<td>CLOSER</td>
<td>0x0002</td>
<td>This mask will attach a closer box to the window which, when pressed, will send a <code>WM_CLOSED</code> message to the application.</td>
</tr>
<tr>
<td>FULLER</td>
<td>0x0004</td>
<td>This mask displays a fuller box with the window which, when pressed, will cause a <code>WM_FULLED</code> message to be sent to the application.</td>
</tr>
<tr>
<td>MOVER</td>
<td>0x0008</td>
<td>This mask allows the user to move the window by clicking and dragging on the window’s title bar. This action will generate a <code>WM_MOVED</code> message.</td>
</tr>
<tr>
<td>INFO</td>
<td>0x0010</td>
<td>This mask creates an information line just below the title bar which can contain any user-defined information as set with <code>wind_set(WF_INFO, ...)</code>.</td>
</tr>
<tr>
<td>SIZER</td>
<td>0x0020</td>
<td>This mask attaches a sizer object to the window which, when clicked and dragged to a new location, will generate a <code>WM_SIZED</code> message.</td>
</tr>
<tr>
<td>UPARROW</td>
<td>0x0040</td>
<td>This mask attaches an up arrow object to the window which, when pressed, will generate a <code>WM_ARROWED</code> message to the application.</td>
</tr>
<tr>
<td>DNARROW</td>
<td>0x0080</td>
<td>This mask attaches a down arrow object to the window which, when pressed, will generate a <code>WM_ARROWED</code> message to the application.</td>
</tr>
<tr>
<td>VSLIDE</td>
<td>0x0100</td>
<td>This mask attaches a vertical slider object to the window which, when clicked and dragged, will generate a <code>WM_VSLID</code> message. Clicking on the exposed area of the slider will also generate this message.</td>
</tr>
<tr>
<td>LFARROW</td>
<td>0x0200</td>
<td>This mask attaches a left arrow object to the window which, when pressed, will generate a <code>WM_ARROWED</code> message to the application.</td>
</tr>
<tr>
<td>RTARROW</td>
<td>0x0400</td>
<td>This mask attaches a right arrow object to the window which, when pressed, will generate a <code>WM_ARROWED</code> message to the application.</td>
</tr>
</tbody>
</table>
HSLIDE 0x0800 This mask attaches a horizontal slider object to the window which, when clicked and dragged, will generate a WM_HSLID message. Clicking on the exposed area of the slider will also generate this message.

SMALLER 0x4000 This mask attaches a smaller object which, when clicked, will generate a WM_ICONIFIED message. If the object is CTRL-clicked, a WM_ALLICONIFY message will be generated.

This object is only valid in AES v4.1 and higher.

\texttt{wind_create()} returns a window handle which should be stored as it must be referenced on any further calls that open, alter, close, or delete the window. \texttt{wind_create()} may fail if too many windows are already open. Different versions of the AES impose different limits on the number of concurrently open windows.

Calling \texttt{wind_create()} does not automatically display the window. \texttt{wind_open()} displays a window named by its window handle. Any calls needed to initialize the window (such as setting the window title, etc.) should be made between the \texttt{wind_create()} and \texttt{wind_open()} calls.

\texttt{wind_set()} and \texttt{wind_get()} can be used to set and retrieve many various window attributes. Look for their documentation in the function reference for further details.

\texttt{wind_close()} may be used to remove a window from the screen. The window itself and its attributes are not deleted as a result of this call, however. A subsequent call to \texttt{wind_open()} will restore a window to the state it was in prior to the \texttt{wind_close()} call. The \texttt{wind_delete()} function is used to physically delete a window and free any memory it was using.

Two other utility functions for use in dealing with windows are provided by the AES. \texttt{wind_calc()} will return the border rectangle of a window given the desired work area or the work area of a window given the desired border area. The call takes into account the sizes of the various window gadgets.

\texttt{wind_find()} returns the handle of the window currently under the mouse.
The Desktop Window
The desktop window encompasses the entire screen. It has a constant window handle of DESK (0) so information about it can be inquired with wind_get(). Calling wind_get() with a parameter of WF_CURRXYWH will return the size of the screen. Calling wind_get() with a parameter of WF_WORKXYWH will return the size of the screen minus the size of the menu bar.

The desktop draws a custom OBJECT tree in its work area. This tree results in the fill pattern and color seen on screen. An application may create its own custom desktop object tree by using wind_set() with a parameter of WF_DESKTOP. The OBJECT tree specified should be the exact size of the desktop work area.

MultiTOS will switch between these object trees as applications are switched. The desktop’s object tree will be visible whenever an application doesn’t specify one of its own.

The Rectangle List
Whenever a window receives a redraw message or needs to update its window because of its reasons, it should always constrain output to its current rectangle list. The AES will calculate the size and position of a group of rectangles that compromise the area of your window not covered by other overlapping windows.

wind_get() with parameters of WF_FIRSTXYWH and WF_NEXTXYWH is used to return the current rectangle list. Redrawing inside a window should also only be attempted when the window semaphore is locked with wind_update(BEG_UPDATE). This prevents the rectangle list from changing during the redraw and prevents the user from dropping down menus which might be overwritten. The following code sample illustrates a routine that correctly steps through the rectangle list:

```
VOID
RedrawWindow( WORD winhandle, GRECT *dirty )
{
  GRECT rect;

  wind_update( BEG_UPDATE );

  wind_get( winhandle, WF_FIRSTXYWH, &rect.g_x, &rect.g_y, &rect.g_w, &rect.g_h);
  while( rect.g_w && rect.g_h )
  {
```

```
if( rc_intersect( dirty, &rect ) )
{
    /*
     * Do your drawing here...constrained to the rectangle in g.
     */
}

wind_get( winhandle, WF_NEXTXYWH, &rect.g_x, &rect.g_y, &rect.g_w,
            &rect.g_h);

wind_update( END_UPDATE );
}

Window Toolbars

AES versions 4.0 and later support window toolbar attachments. Toolbars are OBJECT trees containing a number of TOUCHEXIT objects. They are attached to a window using wind_set() with a parameter of WF_TOOLBAR. The following diagram shows a window with a toolbar:

![Example from Atari Works 2.1](image)

Window toolbars are automatically redrawn whenever necessary and their ROOT objects are automatically repositioned and resized with the window. If any special redrawing is necessary (ex: changing the visual state of an object after a click), the application may obtain a special toolbar rectangle list by using wind_get() with parameters of WF_FTOOLBAR and WF_NTOOLBAR.

If toolbar objects must be modified on WM_SIZED events, simply modify them prior to calling wind_set( handle, WM_CURRXYWH, ... ).

A special note about windows with toolbars concerns the usage of wind_calc(). wind_calc() doesn’t understand the concept of toolbars. The information it returns must be modified by adjusting the height of its output rectangles according to the current height of the toolbar object tree.
The Graphics Library

The Graphics Library contain many functions which can be used to provide visual clues to the user. This library also contains functions to inquire and set information about the mouse pointer.

graf_movebox(), graf_shrinkbox(), and graf_growbox() display animations that can be used to indicate an impending change in the screen display. graf_dragbox(), graf_rubberbox(), and graf_slidebox() display visual effects that are interactively changed by the mouse position.

graf_mkstate() is used to inquire the current state of the mouse buttons and mouse position. graf_mouse() can be used to change the shape of the system mouse. graf_handle() is used to return the physical handle of the screen (needed to open a VDI workstation) and the metrics of the system default text font.

The File Selector Library

Two routines are provided by the AES to display and handle the common system file selector. AES versions less than 1.4 do not support fsel_exinput(). All AES versions support fsel_input().

Both calls take a GEMDOS pathname and filename as parameters. The pathname should include a complete path specification including a drive letter, colon, path, and filemask. The filemask may (and usually does include wildcard characters). The application may also pass a default filename to the selector.

fsel_exinput() allows the application to specify a replacement title for the file selector which reminds the user about the action they are taking such as ‘Select a .DOC file to open...’.

The Scrap Library

The scrp_read() and scrp_write() calls are provided by the AES to return and set the current clipboard path. The clipboard is a global resource in which applications can share data. Applications supporting the clipboard contain an ‘Edit’ menu title which has at least the following four items, ‘Cut’, ‘Copy’, ‘Paste’, and ‘Delete’. An appropriate action for each is listed below:

Implementing ‘Cut’ and ‘Copy’

When the user selects ‘Cut’ or ‘Copy’ from the ‘Edit’ menu and an object is selected (‘Cut’ and ‘Copy’ should only be enabled in the menu when an object is selected which may be transferred to the clipboard) the following steps may be used to transfer the data to the system clipboard:

1. Call scrp_read() to return the name of the current scrap directory. If the returned string is empty, no clipboard directory has been defined since the computer has
been started. The directory string returned may need to be reformatted. A proper
directory string ends in a backslash, however some applications incorrectly
append a filename to this string.

2. If no clipboard directory was returned or the one specified is invalid, create a
directory in the user’s boot drive called ‘\CLIPBRD’ and write the pathname back
using `scrp_write()`. For example, if the user’s boot drive was ‘C:’ then your
parameter to `scrp_write()` would be ‘C:\CLIPBRD’.

3. Search and delete files in the current clipboard directory with the mask
   ‘SCRAP.*’.

4. Now write a disk file for the selected data to a file named SCRAP.??? where ‘???’
is the proper file extension for an object of its type. If the object can be
represented in more than one format by your application, write as many formats as
possible all named ‘SCRAP’ with the proper file extension.

5. If the menu choice was ‘Cut’ rather than ‘Copy,’ delete the object from your data
structures and update your application as necessary.

**Implementing ‘Paste’**

‘Paste’ is used to read a file and insert it appropriately into an application that supports data of
its type. To implement ‘Paste’ follow the steps below:

1. Call `scrp_read()` to obtain the current system clipboard directory. If the returned
   string is empty, no data is in the clipboard.

2. Format the string returned by `scrp_read()` into a usable pathname and search for
   files called ‘SCRAP’ in that path having a file extension of data that your
   application supports. Remember, more than one SCRAP.??? file may be present.

3. Load the data and insert it in your application as appropriate.

**MultiTOS Notes**

The AES, when running under MultiTOS, will create a MiNT semaphore named ‘_SCP’ which
should be used to provide negotiated access to the scrap directory. Access to this semaphore
should be obtained from MiNT prior to any clipboard operation and must be released as soon as
it is complete. Applications should not attempt to destroy this semaphore.
The Shell Library

The Shell Library was originally intended to provide AES support to the Desktop application. Many of the routines, however, are useful to other GEM applications. Some functionality of the Shell Library was discussed earlier in this chapter in ‘The Environment String’.

The Shell Buffer

The Desktop application loads the DESKTOP.INF or NEWDESK.INF file (depending on the TOS version) into the shell buffer. Prior to TOS 2.00, the shell buffer was 1024 bytes long meaning that was the maximum length of the DESKTOP.INF file. AES versions 2.00 to 3.30 allocate a buffer 4096 bytes long. AES versions 3.30 and above support variable-length buffers.

The shell buffer contains the ‘working’ copy of the above mentioned system files. The information in this buffer may be copied by using shel_get(). Likewise, information can be written to this buffer using shel_put(). Extreme care must be used with these functions as their misuse can confuse or possibly even crash the Desktop.

Miscellaneous Shell Library Functions

shel_find() is used to locate data files associated with an application. The AES uses this call to locate application resource files during rsrcl_load().

shel_read() returns information about the process which called the application (usually the Desktop).

shel_write() was originally used only to spawn new applications. With newer AES versions, though, shel_write() has taken on an enormous functionality and its documentation should be consulted for more information.

The GEM.CNF File

When running under MultiTOS, the AES will load and process an ASCII text file called ‘GEM.CNF’ which contains command lines that set environment and AES system variables and may run GEM programs. In addition, a replacement shell program may be specified in this file (see Chapter 9: Desktop for more information).

AES environment variables may be set in the ‘GEM.CNF’ file with the command ‘setenv’ as in the following example:

```
setenv TOSRUN=c:\multitos\miniwin.app
```

Several AES system variables may also be set in this file as shown in the following example:

```
AE_FONTID=3
```
Currently recognized AES system variables that may be set are shown in the following table:

<table>
<thead>
<tr>
<th>Variable</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>AE_FONIDI</td>
<td>This variable may be set to any valid Speedo outline font ID which will be used as the AES default text font.</td>
</tr>
<tr>
<td></td>
<td>This feature is only valid as of AES version 4.1.</td>
</tr>
<tr>
<td>AE_PNTSIZE</td>
<td>This variable defines the size of the AES default text font in points.</td>
</tr>
<tr>
<td></td>
<td>This feature is only valid as of AES version 4.1.</td>
</tr>
<tr>
<td>AE_SREDRAW</td>
<td>Setting this variable to 1 causes the AES to send a full-screen redraw message whenever an application starts. Setting it to 0 disables this feature. The default is 1.</td>
</tr>
<tr>
<td>AE_TREDRAW</td>
<td>Setting this variable to 1 causes the AES to send a full-screen redraw message whenever an application terminates. Setting it to 0 disables this feature. The default is 1.</td>
</tr>
</tbody>
</table>

The ‘GEM.CNF’ file may also be used to automatically start applications as shown in the following example:

```
run c:\multitos\tclock.prg
```

**AES Function Calling Procedure**

The GEM AES is accessed through a 680x0 TRAP #2 statement. Upon calling the TRAP, register d0 should contain the magic number 0xC8 and register d1 should contain a pointer to the AES parameter block. The global data array member of the parameter block is filled in with information about the AES after an appl_init() call (see appl_init() for more details). The AES parameter block is a structure containing pointers to several arrays defined as follows:

```c
struct aespb
{
    WORD *ctrl;
    WORD *global;
    WORD *intin;
    WORD *intout;
    LONG *addrin;
    LONG *addrout;
};
```

The control array is filled in prior to an AES call with information about the number of parameters the function is being passed, the number of return values the function expects, and the opcode of the function itself as follows:
The \textit{intin} array and \textit{addrin} arrays are used to pass integer and address parameters respectively (consult each individual binding for details).

Upon return from the call, the \textit{intout} and \textit{addrout} arrays will be filled in with any appropriate output values.

To add a binding for the AES to your compiler you will usually write a short procedure that provides an interface to the AES arrays. The following example illustrates the binding to \texttt{graf\_dragbox()} in this manner:

\begin{verbatim}
WORD
graf_dragbox( WORD width, WORD height, WORD start_x, WORD start_y,
            WORD box_x, WORD box_y, WORD box_w, WORD box_h,
            WORD *end_x, WORD *end_y )
{
  contrl[0] = 71;
  contrl[1] = 8;
  contrl[2] = 3;
  contrl[3] = 0;
  contrl[4] = 0;
  intin[0] = width;
  intin[1] = height;
  intin[2] = start_x;
  intin[3] = start_y;
  intin[4] = box_x;
  intin[5] = box_y;
  intin[6] = box_w;
  intin[7] = box_h;
  aes();
  *end_x = intout[1];
  *end_y = intout[2];
  return intout[0];
}
\end{verbatim}
The following code is the assembly language function `aes()` used by the function above:

```assembly
.globl _aes
.text
_aes:
    lea _aespb,a0
    move.l a0,d1
    move.w #$C8,d0
    trap #2
    lea _intout,a0
    move.w (a0),d0
    rts
.data
_aespb: .dc.l _contrl, _global, _intin, _intout, _addrin, _addrout
.bss
    _contrl: .ds.w 5
    _global: .ds.w 15
    _intin: .ds.w 16
    _intout: .ds.w 7
    _addrin: .ds.l 2
    _addrout: .ds.l 1
.end
```

The bindings in the **AES Function Reference** call a specialized function called `crys_if()` to actually call the AES. Many compilers use this method as well (Lattice C calls the function `_AESif()`).

`crys_if()` properly fills in the `contrl` array and calls the AES. It is passed one WORD parameter in d0 which contains the opcode of the function minus ten multiplied by four (for quicker table indexing). This gives an index into a table from which the `contrl` array data may be loaded. The `crys_if()` function is listed below:

```assembly
* Note that this binding depends on the fact that no current AES call utilizes
* the addrout array
.globl _crys_if
.globl _aespb
.globl _contrl
.globl _global
.globl _intin
.globl _addrin
.globl _intout
.globl _addrout
.text
_crys_if:
    lea table(pc),a0 ; Table below
```

---

**THE ATARI COMPENDIUM**
move.l 0(a0,d0.w),d0 ; Load four packed bytes into d0
lea _aespb,a0 ; Load address of _aespb into a0
movea.l (a0),a1 ; Move address of _aespb into a1
movep.l d0,1(a1) ; Move four bytes into WORDs at 1(contl)
move.l a0,d1 ; Move address of _aespb into d1
move.w #$C8,d0 ; AES magic number
trap #2 ; Call GEM
lea _intout,a0 ; Get return value
move.w (a0),d0 ; Put it into d0
rts

* Table of AES opcode/control values
* Values are: opcode, intin, intout, addrin
* As stated before, addrout is left at 0 since no AES calls use it

table:

<table>
<thead>
<tr>
<th>Opcode</th>
<th>Intin</th>
<th>Intout</th>
<th>Addrin</th>
</tr>
</thead>
<tbody>
<tr>
<td>10</td>
<td>0</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>11</td>
<td>2</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>12</td>
<td>2</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>13</td>
<td>0</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>14</td>
<td>2</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>15</td>
<td>1</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>16</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>17</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>18</td>
<td>1</td>
<td>3</td>
<td>1</td>
</tr>
<tr>
<td>19</td>
<td>0</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>20</td>
<td>0</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>21</td>
<td>3</td>
<td>5</td>
<td>0</td>
</tr>
<tr>
<td>22</td>
<td>5</td>
<td>5</td>
<td>0</td>
</tr>
<tr>
<td>23</td>
<td>0</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>24</td>
<td>2</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>25</td>
<td>16</td>
<td>7</td>
<td>1</td>
</tr>
<tr>
<td>26</td>
<td>2</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>27</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>28</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>29</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>30</td>
<td>1</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>31</td>
<td>2</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>32</td>
<td>2</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>33</td>
<td>2</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>34</td>
<td>1</td>
<td>1</td>
<td>2</td>
</tr>
<tr>
<td>35</td>
<td>1</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>36</td>
<td>2</td>
<td>1</td>
<td>2</td>
</tr>
<tr>
<td>37</td>
<td>2</td>
<td>1</td>
<td>2</td>
</tr>
<tr>
<td>38</td>
<td>3</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>39</td>
<td>1</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>40</td>
<td>2</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>41</td>
<td>1</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>42</td>
<td>6</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>43</td>
<td>4</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>44</td>
<td>1</td>
<td>3</td>
<td>1</td>
</tr>
<tr>
<td>45</td>
<td>2</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>46</td>
<td>4</td>
<td>2</td>
<td>1</td>
</tr>
<tr>
<td>47</td>
<td>8</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>48</td>
<td>4</td>
<td>3</td>
<td>0</td>
</tr>
<tr>
<td>49</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>50</td>
<td>1</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>51</td>
<td>9</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>52</td>
<td>1</td>
<td>1</td>
<td>1</td>
</tr>
</tbody>
</table>

The Atari Compendium
.dc.b 53, 1, 1, 0 ; form_error
.dc.b 54, 0, 5, 1 ; form_center
.dc.b 55, 3, 3, 1 ; form_keybd
.dc.b 56, 2, 2, 1 ; form_button
.dc.b 57, 0, 0, 0 ;
.dc.b 58, 0, 0, 0 ;
.dc.b 59, 0, 0, 0 ;
.dc.b 60, 0, 0, 0 ;
.dc.b 61, 0, 0, 0 ;
.dc.b 62, 0, 0, 0 ;
.dc.b 63, 0, 0, 0 ;
.dc.b 64, 0, 0, 0 ;
.dc.b 65, 0, 0, 0 ;
.dc.b 66, 0, 0, 0 ;
.dc.b 67, 0, 0, 0 ;
.dc.b 68, 0, 0, 0 ;
.dc.b 69, 0, 0, 0 ;
.dc.b 70, 4, 3, 0 ; graf_rubberbox
.dc.b 71, 8, 3, 0 ; graf_dragbox
.dc.b 72, 6, 1, 0 ; graf_movebox
.dc.b 73, 8, 1, 0 ; graf_growbox
.dc.b 74, 8, 1, 0 ; graf_shrinkbox
.dc.b 75, 4, 1, 1 ; graf_watchbox
.dc.b 76, 3, 1, 1 ; graf_slidebox
.dc.b 77, 0, 5, 0 ; graf_handle
.dc.b 78, 1, 1, 1 ; graf_mouse
.dc.b 79, 0, 5, 0 ; graf_mkstate
.dc.b 80, 0, 1, 1 ; scrp_read
.dc.b 81, 0, 1, 1 ; scrp_write
.dc.b 82, 0, 0, 0 ;
.dc.b 83, 0, 0, 0 ;
.dc.b 84, 0, 0, 0 ;
.dc.b 85, 0, 0, 0 ;
.dc.b 86, 0, 0, 0 ;
.dc.b 87, 0, 0, 0 ;
.dc.b 88, 0, 0, 0 ;
.dc.b 89, 0, 0, 0 ;
.dc.b 90, 0, 2, 2 ; fsel_input
.dc.b 91, 0, 2, 3 ; fsel_exinput
.dc.b 92, 0, 0, 0 ;
.dc.b 93, 0, 0, 0 ;
.dc.b 94, 0, 0, 0 ;
.dc.b 95, 0, 0, 0 ;
.dc.b 96, 0, 0, 0 ;
.dc.b 97, 0, 0, 0 ;
.dc.b 98, 0, 0, 0 ;
.dc.b 99, 0, 0, 0 ;
.dc.b 100, 5, 1, 0 ; wind_create
.dc.b 101, 5, 1, 0 ; wind_open
.dc.b 102, 1, 1, 0 ; wind_close
.dc.b 103, 1, 1, 0 ; wind_delete
.dc.b 104, 2, 5, 0 ; wind_get
.dc.b 105, 6, 1, 0 ; wind_set
.dc.b 106, 2, 1, 0 ; wind_find
.dc.b 107, 1, 1, 0 ; wind_update
.dc.b 108, 6, 5, 0 ; wind_calc
.dc.b 109, 0, 0, 0 ; wind_new
.dc.b 110, 0, 1, 1 ; rsrct_load
.dc.b 111, 0, 1, 0 ; rsrct_free

THE ATARI COMpendium
.dc.b 112, 2, 1, 0 ; rsr_gaddr
.dc.b 113, 2, 1, 1 ; rsr_saddr
.dc.b 114, 1, 1, 1 ; rsr_obfix
.dc.b 115, 0, 0, 0 ; rsr_rcfix (v4.0)
.dc.b 116, 0, 0, 0 ;
.dc.b 117, 0, 0, 0 ;
.dc.b 118, 0, 0, 0 ;
.dc.b 119, 0, 0, 0 ;
.dc.b 120, 0, 1, 2 ; shel_read
.dc.b 121, 3, 1, 2 ; shel_write
.dc.b 122, 1, 1, 1 ; shel_get
.dc.b 123, 1, 1, 1 ; shel_put
.dc.b 124, 0, 1, 1 ; shel_find
.dc.b 125, 0, 1, 2 ; shel_envrn
.dc.b 126, 0, 0, 0 ;
.dc.b 127, 0, 0, 0 ;
.dc.b 128, 0, 0, 0 ;
.dc.b 129, 0, 0, 0 ;
.dc.b 130, 1, 5, 0 ; appl_getinfo (v4.0)

.data

_aespb: .dc.l _contrl, _global, _intin, _intout, _addrin, _addrout
_contrl: .dc.l 0, 0, 0, 0, 0

.bss

* _contrl = opcode
* _contrl+2 = num_intin
* _contrl+4 = num_addrin
* _contrl+6 = num_intout
* _contrl+8 = num_addrout

_global .ds.w 15
_intin .ds.w 16
_intout .ds.w 7
_addrin .ds.l 2
_addrout .ds.l 1

.end
The Application Services Library provides general use functions used in locating and working with other resident applications in addition to providing AES initialization and termination code. The members of the Application Services Library are:

- appl_exit()
- appl_find()
- appl_getinfo()
- appl_init()
- appl_read()
- appl_search()
- appl_tplay()
- appl_trecord()
- appl_write()
appl_exit()

WORD appl_exit( VOID )

appl_exit() should be called at the termination of any program initialized with appl_init().

Opcode 19 (0x13)

Availability All AES versions.

Binding return crys_if(0x13);

Return Value appl_exit() returns 0 if an error occurred or non-zero otherwise.

Comments The proper procedure for handling an error from this function is currently undefined.

See Also appl_init()

appl_find()

WORD appl_find( fname )
CHAR *fname;

appl_find() searches the AES’s current process list for a program named fname and, if present, returns the application identifier of the process.

Opcode 13 (0x0D)

Availability All AES versions.

Parameters fname is a pointer to a null-terminated ASCII string containing a valid GEMDOS filename (not including an extension) padded with blanks to be exactly 8 characters long (not including the NULL).

Binding

addrin[0] = fname;
return crys_if(0x0D);

Return Value appl_find() returns the application identifier of the process if it is found or -1 otherwise.
The Atari Compendium

**Version Notes**

AES versions from 4.0 add several extensions to this call for the benefit of MultiTOS as follows:

- If the upper word of the `CHAR *` is 0xFFFF, the lower word is assumed to be the MiNT id and `appl_find()` will return the AES application identifier.
- If the upper word of the `CHAR *` is 0xFFFE, the lower word is assumed to be the AES application identifier and the MiNT id is returned.
- If the upper word of the `CHAR *` is 0x0000, the current processes’ application identifier is returned.

This functionality only exists if the AES version is 4.0 and above and `appl_getinfo()` indicates that it is available.

**See Also**

`appl_write()`, `appl_init()`

---

**appl_getinfo()**

```c
WORD appl_getinfo(ap_gtype, ap_gout1, ap_gout2, ap_gout3, ap_gout4 )
WORD ap_gtype;
WORD *ap_gout1, *ap_gout2, *ap_gout3, *ap_gout4;
```

`appl_getinfo()` returns information about the AES.

**Opcode**

130 (0x82)

**Availability**

Available as of AES version 4.00.

**Parameters**

`ap_gtype` specifies the type of information to be returned in the shorts pointed to by `ap_gout1`, `ap_gout2`, `ap_gout3`, and `ap_gout4` as follows:

<table>
<thead>
<tr>
<th>Name</th>
<th>Value</th>
<th>Returns</th>
</tr>
</thead>
<tbody>
<tr>
<td>AES_LARGEFONT</td>
<td>0</td>
<td>AES Large Font Information</td>
</tr>
<tr>
<td></td>
<td></td>
<td>*ap_gout1 is filled in with the AES font’s point size.</td>
</tr>
<tr>
<td></td>
<td></td>
<td>*ap_gout2 is filled in with the font id.</td>
</tr>
<tr>
<td></td>
<td></td>
<td>*ap_gout3 is a code indicating the type of font:</td>
</tr>
<tr>
<td></td>
<td></td>
<td>SYSTEM_FONT (0) is the system font</td>
</tr>
<tr>
<td></td>
<td></td>
<td>OUTLINE_FONT (1) is an outline font</td>
</tr>
<tr>
<td></td>
<td></td>
<td>*ap_gout4 is unused.</td>
</tr>
<tr>
<td>AES_SMALLFONT</td>
<td>1</td>
<td>AES Large Font Information</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Same as above for the current small font.</td>
</tr>
</tbody>
</table>

---

**The Atari Compendium**
<table>
<thead>
<tr>
<th>AES_SYSTEM</th>
<th>2</th>
<th>AES System Specifics</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td><em>ap_gout1</em> is filled in with the resolution number (as would be returned by <em>Getrez()</em>).</td>
</tr>
<tr>
<td></td>
<td></td>
<td><em>ap_gout2</em> is filled in with the number of colors supported by the AES object library.</td>
</tr>
<tr>
<td></td>
<td></td>
<td><em>ap_gout3</em> is 0 if color icons are not supported or 1 if they are.</td>
</tr>
<tr>
<td></td>
<td></td>
<td><em>ap_gout4</em> is 0 to indicate that the extended resource file format is not supported or 1 if it is.</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>AES_LANGUAGE</th>
<th>3</th>
<th>AES Globalization</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td><em>ap_gout1</em> is filled in with the current AES language code as follows:</td>
</tr>
<tr>
<td></td>
<td></td>
<td><strong>Name</strong></td>
</tr>
<tr>
<td></td>
<td>AESLANG_ENGLISH</td>
<td>0</td>
</tr>
<tr>
<td></td>
<td>AESLANG_GERMAN</td>
<td>1</td>
</tr>
<tr>
<td></td>
<td>AESLANG_FRENCH</td>
<td>2</td>
</tr>
<tr>
<td></td>
<td>—</td>
<td>3</td>
</tr>
<tr>
<td></td>
<td>AESLANG_SPANISH</td>
<td>4</td>
</tr>
<tr>
<td></td>
<td>AESLANG_ITALIAN</td>
<td>5</td>
</tr>
<tr>
<td></td>
<td>AESLANG_SWEDISH</td>
<td>6</td>
</tr>
<tr>
<td></td>
<td><em>ap_gout2</em>, <em>ap_gout3</em>, and <em>ap_gout4</em> are unused.</td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>AES_PROCESS</th>
<th>4</th>
<th>AES Multiple Process Support</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td><em>ap_gout1</em> is 0 to indicate the use of non-pre-emptive multitasking and 1 to indicate the use of pre-emptive multitasking.</td>
</tr>
<tr>
<td></td>
<td></td>
<td><em>ap_gout2</em> is 0 if <em>appl_find()</em> cannot convert between MiNT and AES id’s and 1 to indicate that it can.</td>
</tr>
<tr>
<td></td>
<td></td>
<td><em>ap_gout3</em> is 0 if <em>appl_search()</em> is not implemented and 1 if it is.</td>
</tr>
<tr>
<td></td>
<td></td>
<td><em>ap_gout4</em> is 0 if <em>rsrc_rctfix()</em> is not implemented and 1 if it is.</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>AES_PCGEM</th>
<th>5</th>
<th>AES PC-GEM Features</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td><em>ap_gout1</em> is 0 if <em>objc_xfind()</em> is not implemented and 1 if it is.</td>
</tr>
<tr>
<td></td>
<td></td>
<td><em>ap_gout2</em> is currently reserved.</td>
</tr>
<tr>
<td></td>
<td></td>
<td><em>ap_gout3</em> is 0 if <em>menu_click()</em> is not implemented and 1 if it is.</td>
</tr>
<tr>
<td></td>
<td></td>
<td><em>ap_gout4</em> is 0 if <em>shel_rdef()</em> and <em>shel_wdef()</em> are not implemented and 1 if they are.</td>
</tr>
</tbody>
</table>
### AES_INQUIRE

<table>
<thead>
<tr>
<th>Code</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>6</td>
<td>AES Extended Inquiry Functions</td>
</tr>
<tr>
<td></td>
<td><strong>ap_gout1</strong> is 0 if -1 is not a valid <strong>ap_id</strong> parameter to <strong>appl_read()</strong> or 1 if it is.</td>
</tr>
<tr>
<td></td>
<td><strong>ap_gout2</strong> is 0 if -1 is not a valid length parameter to <strong>shel_get()</strong> or 1 if it is.</td>
</tr>
<tr>
<td></td>
<td><strong>ap_gout3</strong> is 0 if -1 is not a valid <strong>mode</strong> parameter to <strong>menu_bar()</strong> or 1 if it is.</td>
</tr>
<tr>
<td></td>
<td><strong>ap_gout4</strong> is 0 if <strong>MENU_INSTL</strong> is not a valid <strong>mode</strong> parameter to <strong>menu_bar()</strong> or 1 if it is.</td>
</tr>
</tbody>
</table>

#### AES_MOUSE

<table>
<thead>
<tr>
<th>Code</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>8</td>
<td>AES Mouse Support</td>
</tr>
<tr>
<td></td>
<td><strong>ap_gout1</strong> is 0 to indicate that <strong>mode</strong> parameters of 258-260 are not supported by <strong>graf_mouse()</strong> and 1 if they are.</td>
</tr>
<tr>
<td></td>
<td><strong>ap_gout2</strong> is 0 to indicate that the application has control over the mouse form and 1 to indicate that the mouse form is maintained by the AES on a per-application basis.</td>
</tr>
<tr>
<td></td>
<td><strong>ap_gout3</strong> and <strong>ap_gout4</strong> are currently unused.</td>
</tr>
</tbody>
</table>

#### AES_MENU

<table>
<thead>
<tr>
<th>Code</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>9</td>
<td>AES Menu Support</td>
</tr>
<tr>
<td></td>
<td><strong>ap_gout1</strong> is 0 to indicate that sub-menus are not supported and 1 if <strong>MultiTOS</strong> style sub-menus are.</td>
</tr>
<tr>
<td></td>
<td><strong>ap_gout2</strong> is 0 to indicate that popup menus are not supported and 1 if <strong>MultiTOS</strong> style popup menus are.</td>
</tr>
<tr>
<td></td>
<td><strong>ap_gout3</strong> is 0 to indicate that scrollable menus are not supported and 1 if <strong>MultiTOS</strong> style scrollable menus are.</td>
</tr>
<tr>
<td></td>
<td><strong>ap_gout4</strong> is 0 to indicate that the <strong>MN_SELECTED</strong> message does not contain object tree information in <strong>msg[5-7]</strong> and 1 to indicate that it does.</td>
</tr>
</tbody>
</table>
### AES_Shell Support

`ap_gout1 & 0x00FF` indicates the highest legal value for the `mode` parameter of `shel_write()`. `ap_gout1 & 0xFF00` indicate which extended `shel_write()` `mode` bits are supported.

`ap_gout2` is 0 if `shel_write()` with a `mode` parameter of 0 launches an application or 1 if it cancels the previous `shel_write()`.

`ap_gout3` is 0 if `shel_write()` with a `mode` parameter of 1 launches an application immediately or 1 if it takes effect when the current application exits.

`ap_gout4` is 0 if ARGV style parameter passing is not supported or 1 if it is.

### AES_Window Features

`ap_gout1` is a bitmap of extended modes supported by `wind_get()` and `wind_set()` (if a bit is set, it is supported) as follows:

<table>
<thead>
<tr>
<th>Bit</th>
<th>Mode</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td><code>WF_TOP</code> returns window below the top also.</td>
</tr>
<tr>
<td>1</td>
<td><code>wind_get(WF_NEWDESK, ...) supported.</code></td>
</tr>
<tr>
<td>2</td>
<td><code>WF_COLOR</code> get/set.</td>
</tr>
<tr>
<td>3</td>
<td><code>WF_DCOLOR</code> get/set.</td>
</tr>
<tr>
<td>4</td>
<td><code>WF_OWNER</code> get/set.</td>
</tr>
<tr>
<td>5</td>
<td><code>WF_BEVENT</code> get/set.</td>
</tr>
<tr>
<td>6</td>
<td><code>WF_BOTTOM</code> set.</td>
</tr>
<tr>
<td>7</td>
<td><code>WF_ICONIFY</code> set.</td>
</tr>
<tr>
<td>8</td>
<td><code>WF_UNICONIFY</code> set.</td>
</tr>
<tr>
<td>9-15</td>
<td>Unused</td>
</tr>
</tbody>
</table>

`ap_gout2` is currently unused.

`ap_gout3` is a bitmap of supported window behaviors (if a bit is set, it is supported) as follows:

<table>
<thead>
<tr>
<th>Bit</th>
<th>Behaviour</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>Iconifier gadget present.</td>
</tr>
<tr>
<td>1</td>
<td>Bottomer gadget present.</td>
</tr>
<tr>
<td>2</td>
<td>SHIFT-click sends window to bottom.</td>
</tr>
<tr>
<td>3</td>
<td>“hot” close box supported.</td>
</tr>
<tr>
<td>4-15</td>
<td>Unused</td>
</tr>
</tbody>
</table>

`ap_gout4` is currently unused.
### AES_MESSAGE

<table>
<thead>
<tr>
<th>Bit</th>
<th>Message</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>WM_NEWTOP is meaningful.</td>
</tr>
<tr>
<td>1</td>
<td>WM_UNTOPPED is sent.</td>
</tr>
<tr>
<td>2</td>
<td>WM_ONTOP is sent.</td>
</tr>
<tr>
<td>3</td>
<td>AP_TERM is sent.</td>
</tr>
<tr>
<td>4</td>
<td>Shutdown and resolution change messages.</td>
</tr>
<tr>
<td>5</td>
<td>CH_EXIT is sent.</td>
</tr>
<tr>
<td>6</td>
<td>WM_BOTTOM is sent.</td>
</tr>
<tr>
<td>7</td>
<td>WM_ICONIFY is sent.</td>
</tr>
<tr>
<td>8</td>
<td>WM_UNICONIFY is sent.</td>
</tr>
<tr>
<td>9</td>
<td>WM_ALLICONIFY is sent.</td>
</tr>
<tr>
<td>10-15</td>
<td>Unused</td>
</tr>
</tbody>
</table>

*ap_gout1* is a bitmap of extra messages supported (if a bit is set, it is supported) as follows:

*ap_gout2* is a bitmap of extra messages supported. Current all bits are unused.

*ap_gout3* is a bitmap indicating message behaviour (if a bit is set, the behaviour exists) as follows:

<table>
<thead>
<tr>
<th>Bit</th>
<th>Message</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>WM_ICONIFY message gives coordinates.</td>
</tr>
<tr>
<td>1-15</td>
<td>Unused</td>
</tr>
</tbody>
</table>

*ap_gout4* is currently unused.

### AES_OBJECT

*ap_gout1* is 0 if 3D objects are not supported or 1 if they are.

*ap_gout2* is 0 if `objc_sysvar()` is not present, 1 if MultiTOS v1.01 `objc_sysvar()` is present, or 2 if extended `objc_sysvar()` is present.

*ap_gout3* is 0 if the system font is the only font supported or 1 if GDOS fonts are also supported.

*ap_gout4* is reserved for OS extensions.

### AES_FORM

*ap_gout1* is 0 if ‘flying dialogs’ are not supported or 1 if they are.

*ap_gout2* is 0 if keyboard tables are not supported or 1 if Mag!X style keyboard tables are supported.

*ap_gout3* is 0 if the last cursor position from `objc_edit()` is not returned or 1 if it is.

*ap_gout4* is currently reserved.
intin[0] = ap_gtype;
crys_if(0x82);
*ap_gout1 = inout[1];
*ap_gout2 = inout[2];
*ap_gout3 = inout[3];
*ap_gout4 = inout[4];
return inout[0];

appl_getinfo() returns 1 if an error occurred or 0 otherwise.

Using an ap_gtype value of 4 and above is only supported as of AES version 4.1.

Many of the ap_gtype return values identify features of TOS not supported by Atari but for the benefit of third-party vendors. You should contact the appropriate third-party for documentation on these functions.

appl_init()

WORD appl_init( VOID )

appl_init() should be the first function called in any application that intends to use GEM calls.

10 (0x0A)

All AES versions.

The function as prototyped accepts no parameters, however, all ‘C’ compilers use this call to set up internal information as well as to update the applications’ global array.

return crys_if(0x0A);

appl_init() returns the applications’ global identifier if successful or -1 if the AES cannot register the application. If successful, the global identifier should be stored in a global variable for later use.

Besides the return value, the AES fills in the application’s global array (to reference the global array see your programming languages’ manual).

<table>
<thead>
<tr>
<th>Name</th>
<th>global[x]</th>
<th>Meaning</th>
</tr>
</thead>
</table>

The Atari Compendium
**_AESversion_** 0  
AES version number.

**_AESnumapps_** 1  
Number of concurrent applications possible (normally 1).  
**MultiTOS** will return -1.

**_AESapid_** 2  
Application identifier (same as **appl_init()** return value).

**_AESappglobal_** 3-4  
LONG global available for use by the application.

**_AESrscfile_** 5-6  
Pointer to the base of the resource loaded via **rsrc_load()**.

— 7-12  
Reserved

**_AESmaxchar_** 13  
Current maximum character used by the AES to do **vst_height()** prior to writing to the screen. This entry is only present as of AES version 0x0400.

**_AESminchar_** 14  
Current minimum character used by the AES to do **vst_height()** prior to writing to the screen. This entry is only present as of AES version 0x0400.

**VERSION NOTES**  
See above.

**SEE ALSO**  
**appl_exit()**

---

### **appl_read()**

**WORD appl_read( ap_id, length, message )**

**WORD ap_id, length;**

**VOIDP message;**

**appl_read()** is designed to facilitate inter-process communication between processes running under the **AES**. The call will halt the application until a message of sufficient length is available (see version notes below).

**OPCODE**  
11 (0x0B)

**AVAILABILITY**  
All AES versions.

**PARAMETERS**  
*ap_id* is your application identifier as returned by **appl_init()**. *length* is the length (in bytes) of the message to read. *message* is a pointer to a memory buffer where the incoming message should be copied to.

**BINDING**

```c
intin[0] = ap_id;
intin[1] = length;
addrin[0] = message;
return crys_if(0x0B);
```

**RETURN VALUE**  
**appl_read()** returns 0 if an error occurred or non-zero otherwise.
If the AES version is 4.0 or higher and `appl_getinfo()` indicates that this feature is supported, `ap_id` takes on an additional meaning. If `APR_NOWAIT` (-1) is passed instead of `ap_id`, `appl_read()` will return immediately if no message is currently waiting.

Normally this call is not used. `evnt_multi()` or `evnt_mesag()` is used instead for standard message reception. `appl_read()` is required for reading messages that are long and/or of variable length.

It is recommended that message lengths in multiples of 16 bytes be used.

**See Also**

`appl_write()`

---

**appl_search()**

```c
WORD appl_search( mode, fname, type, ap_id )
WORD mode;
CHAR *fname;
WORD *type,*ap_id;
```

`appl_search()` provides a method of identifying all of the currently running processes.

**Opcode**

18 (0x12)

**Availability**

Available only in AES versions 4.0 and above when `appl_getinfo()` indicates its presence.

**Parameters**

`mode` specifies the search mode as follows:

<table>
<thead>
<tr>
<th>Name</th>
<th>mode</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>APP_FIRST</td>
<td>0</td>
<td>Return the filename of the first process</td>
</tr>
<tr>
<td>APP_NEXT</td>
<td>1</td>
<td>Return the filename of subsequent processes</td>
</tr>
</tbody>
</table>

`fname` should point to a memory location at least 9 bytes long to hold the 8 character process filename found and the NULL byte. `type` is a pointer to a `WORD` into which will be placed the process type as follows:

<table>
<thead>
<tr>
<th>Name</th>
<th>type</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>APP_SYSTEM</td>
<td>0x01</td>
<td>System process</td>
</tr>
<tr>
<td>APP_APPLICATION</td>
<td>0x02</td>
<td>Application</td>
</tr>
<tr>
<td>APP_ACCESSORY</td>
<td>0x04</td>
<td>Accessory</td>
</tr>
<tr>
<td>APP_SHELL</td>
<td>0x08</td>
<td></td>
</tr>
</tbody>
</table>
The `type` parameter is actually a bit mask so it is possible that a process containing more than one characteristic will appear. The currently running shell process (usually the desktop) will return a value of `APP_APPLICATION | APP_SHELL` (0x0A).

`ap_id` is a pointer to a word into which will be placed the processes’ application identifier.

**BINDING**

```
intin[0] = mode;
addrin[0] = fname;
addrin[1] = type;
addrin[2] = ap_id;
return crys_if(0x12);
```

**RETURN VALUE** `appl_search()` returns 0 if no more applications exist or 1 when more processes exist that meet the search criteria.

---

### appl_tplay()

**WORD** `appl_tplay(mem, num, scale)`

**VOIDP** `mem;`

**WORD** `num, scale;`

`appl_tplay()` plays back events originally recorded with `appl_trecord()`.

**OPCODE** 14 (0x0E)

**AVAILABILITY** All AES versions.

**PARAMETERS** `mem` is a pointer to an array of `EVNTREC` structures (see `appl_trecord()`). `num` indicates the number of `EVNTREC`’s to play back.

`scale` indicates on a scale of 1 to 10000 how fast the AES will attempt to play back your recording. A value of 100 will play it back at recorded speed. A value of 200 will play the events back at twice the recorded speed, and 50 will play back the events at half of the recorded speed. Other values will respond accordingly.

**BINDING**

```
intin[0] = num;
intin[1] = scale;
addrin[0] = mem;
return crys_if(0x0E);
```
appl_trecord() always returns 1 meaning no error occurred.

**Caveats**

This function does not work correctly on AES versions less than 1.40 without a patch program available from Atari Corp.

**See Also**

appl_trecord()

---

**appl_trecord()**

WORD appl_trecord( mem, num )

VOIDP mem;

WORD num;

appl_trecord() records AES events for later playback.

**Opcode**

15 (0xF)

**Availability**

All AES versions.

**Parameters**

mem points to an array of num EVNTREC structures into which the AES will record events as indicated here:

```c
typedef struct pEvntrec
{
    WORD ap_event;
    LONG ap_value;
} EVNTREC;
```

ap_event defines the required interpretation of ap_value as follows:

<table>
<thead>
<tr>
<th>Name</th>
<th>ap_event</th>
<th>Event</th>
<th>ap_value</th>
</tr>
</thead>
<tbody>
<tr>
<td>APPEVNT_TIMER</td>
<td>0</td>
<td>Timer</td>
<td>Elapsed Time (in milliseconds)</td>
</tr>
<tr>
<td>APPEVNT_BUTTON</td>
<td>1</td>
<td>Button</td>
<td>low word = state (1 = down) high word = # of clicks</td>
</tr>
<tr>
<td>APPEVNT_MOUSE</td>
<td>2</td>
<td>Mouse</td>
<td>low word = X pos high word = Y pos</td>
</tr>
<tr>
<td>APPEVNT_KEYBOARD</td>
<td>3</td>
<td>Keyboard</td>
<td>bits 0-7: ASCII code bits 8-15: scan code bits 16-31: shift key state</td>
</tr>
</tbody>
</table>

**Binding**

```c
intin[0] = num;
addrin[0] = mem;
return crys_if(0x0F);
```
appl_write()

WORD appl_write(ap_id, length, msg)
WORD ap_id, length;
VOIDP msg;

appl_write() can be used to send a message to a valid message pipe.

OPCODE 12 (0x0C)

AVAILABILITY All AES versions.

PARAMETERS ap_id is the application identifier of the process to which you wish to send the message. length specifies the number of bytes present in the message. msg is a pointer to a memory buffer with at least length bytes available.

BINDING

intin[0] = ap_id;
intin[1] = length;
addrin[0] = msg;
return crys_if(0x0C);

RETURN VALUE appl_write() returns 0 if an error occurred or greater than 0 if the message was sent successfully.

VERSION NOTES As of AES version 1.40, desk accessories may send MN_SELECTED messages to the desktop to trigger desktop functions.

As of AES version 4.00 you can use shel_write(7,...) to ‘broadcast’ a message to all processes running with the exception of the AES itself, the desktop, and your own application. See shel_write() for details.

COMMENTS It is recommended that you always send messages in 16 byte blocks using a WORD array of 8 elements as the AES does.

SEE ALSO appl_read(), shel_write()
The Event Library consists of a group of system calls which are used to monitor system messages including mouse clicks, keyboard usage, menu bar interaction, timer calls, and mouse tracking. The library consists of the following calls:

- evnt_button()
- evnt_dclick()
- evnt_keybd()
- evnt_mesag()
- evnt_mouse()
- evnt_multi()
- evnt_timer()
- evnt_button()
evnt_button()

WORD evnt_button( clicks, mask, state, mx, my, button, kstate )
WORD clicks, mask, state;
WORD *mx, *my, *button, *kstate;

evnt_button() releases control to the operating system until the specified mouse button event has occurred.

**Opcode**
21 (0x15)

**Availability**
All AES versions.

**Parameters**

*clicks* specifies the number of mouse-clicks that must occur before returning.

*mask* specifies the mouse buttons to wait for as follows:

<table>
<thead>
<tr>
<th>Name</th>
<th>mask</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>LEFT_BUTTON</td>
<td>0x01</td>
<td>Left mouse button</td>
</tr>
<tr>
<td>RIGHT_BUTTON</td>
<td>0x02</td>
<td>Right mouse button</td>
</tr>
<tr>
<td>MIDDLE_BUTTON</td>
<td>0x04</td>
<td>Middle button (this button would be the first button to the left of the rightmost button on the device).</td>
</tr>
<tr>
<td>—</td>
<td>0x08</td>
<td>Other buttons (0x08 is the mask for the button to the immediate left of the middle button. Masks continue leftwards).</td>
</tr>
</tbody>
</table>

*state* specifies the button state that must occur before returning as follows:

<table>
<thead>
<tr>
<th>mask</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>0x00</td>
<td>All buttons released</td>
</tr>
<tr>
<td>0x01</td>
<td>Left button depressed</td>
</tr>
<tr>
<td>0x02</td>
<td>Right button depressed</td>
</tr>
<tr>
<td>0x04</td>
<td>Middle button depressed</td>
</tr>
<tr>
<td>0x08</td>
<td>etc...</td>
</tr>
</tbody>
</table>

*mx* is a pointer to a WORD which upon return will contain the x-position of the mouse pointer at the time of the event.

*my* is a pointer to a WORD which upon return will contain the y-position of the mouse pointer at the time of the event.

*button* is a pointer to a WORD which upon return will contain the mouse button state as defined in *state*.

*kstate* is a pointer to a WORD which upon return will contain the current status...
of the keyboard shift keys. The value is a bit-mask defined as follows:

<table>
<thead>
<tr>
<th>Name</th>
<th>Mask</th>
<th>Key</th>
</tr>
</thead>
<tbody>
<tr>
<td>K_RSHIFT</td>
<td>0x01</td>
<td>Right Shift</td>
</tr>
<tr>
<td>K_LSHIFT</td>
<td>0x02</td>
<td>Left Shift</td>
</tr>
<tr>
<td>K_CTRL</td>
<td>0x04</td>
<td>Control</td>
</tr>
<tr>
<td>K_ALT</td>
<td>0x08</td>
<td>Alternate</td>
</tr>
</tbody>
</table>

**BINDING**

```c
int in[0] = clicks;
int in[1] = mask;
int in[2] = state;
crys_if(0x15);
*mx = intout[1];
*my = intout[2];
*button = intout[3];
*kstate = intout[4];
return intout[0];
```

**RETURN VALUE**

Upon exit, `evnt_button()` returns a `WORD` indicating the number of times the mouse button state matched `state`.

**COMMENTS**

A previously undocumented feature of this call is accessed by logically OR’ing the `mask` parameter with 0x100. This causes the call to return when independent buttons are depressed. For example, a `mask` value of 0x03 will return when both the left and right mouse buttons are depressed. A `mask` value of 0x103 will cause the call to return when either button is depressed.

**SEE ALSO**

`evnt_multi()`

---

**evnt_dclick()**

```c
WORD evnt_dclick( new, flag )
WORD new, flag;
```

`evnt_dclick()` sets the mouse double-click response rate. This call is global, and thus, affects all applications.

**OPCODE**

26 (0x1A)

**AVAILABILITY**

All AES versions.

**PARAMETERS**

If `flag` is `EDC_INQUIRE` (0), `new` is ignored and the current double-click rate is returned. If `flag` is `EDC_SET` (1), `new` specifies the new double-click rate as
follows:

<table>
<thead>
<tr>
<th>flag</th>
<th>Response</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>Slowest</td>
</tr>
<tr>
<td>1</td>
<td></td>
</tr>
<tr>
<td>2</td>
<td></td>
</tr>
<tr>
<td>3</td>
<td></td>
</tr>
<tr>
<td>4</td>
<td>Fastest</td>
</tr>
</tbody>
</table>

BINDING

```c
int in[0] = new;
in[1] = flag;
return crys_if(0x1A);
```

RETURN VALUE evnt_dclick() returns the newly set or current double-click rate based on flag.

COMMENTS Because this setting is global for all applications, Atari has strongly recommended that developers use this call only where appropriate (such as in a configuration CPX like the General Setup CPX included with XCONTROL).

---

**evnt_keybd()**

WORD evnt_keybd( VOID )

`evnt_keybd()` relinquishes program control to the operating system until a valid keypress is available in the applications’ message pipe.

OPCODE 20 (0x14)

AVAILABILITY All AES versions.

PARAMETERS None

BINDING

```c
return crys_if(0x14);
```

RETURN VALUE `evnt_keybd()` returns a 16-bit value containing the ASCII code of the key entered in the lower eight bits and the scan code in the upper 8-bits.

VERSION NOTES TOS versions released at or above 2.06 and 3.06 disabled reception of keys 1 through 9 on the numeric keypad when used in conjunction with the alternate key. Users may now enter the full range of ASCII values by holding down ALT, typing in the decimal ASCII code, and then releasing the ALT key. These keys, therefore, should not be used by applications. The standard numeric keypad is still available.

SEE ALSO evnt_multi()
evnt_mesag()

WORD evnt_mesag( msg )
WORD *msg;

`evnt_mesag()` releases control to the operating system until a valid system message is available in the applications’ message pipe.

**OPCODE**  
23 (0x17)

**AVAILABILITY**  
All AES versions.

**PARAMETERS**  
`msg` is a pointer to an array of 8 WORD’s to be used as a message buffer.

**BINDING**  
`addrin[0] = msg`  
`return crys_if(0x17);`

**RETURN VALUE**  
The return value is currently reserved by Atari and currently is defined as 1. The array `msg` is filed in with the following values:
<table>
<thead>
<tr>
<th>Index</th>
<th>Description</th>
<th>Possible Values</th>
<th>#</th>
</tr>
</thead>
<tbody>
<tr>
<td>msg[0]</td>
<td>Message Type</td>
<td>MN_SELECTED</td>
<td>10</td>
</tr>
<tr>
<td></td>
<td></td>
<td>WM_REDRAW</td>
<td>20</td>
</tr>
<tr>
<td></td>
<td></td>
<td>WM_TOPPED</td>
<td>21</td>
</tr>
<tr>
<td></td>
<td></td>
<td>WM_CLOSED</td>
<td>22</td>
</tr>
<tr>
<td></td>
<td></td>
<td>WM_FULLED</td>
<td>23</td>
</tr>
<tr>
<td></td>
<td></td>
<td>WM_ARROWED</td>
<td>24</td>
</tr>
<tr>
<td></td>
<td></td>
<td>WM_HSLID</td>
<td>25</td>
</tr>
<tr>
<td></td>
<td></td>
<td>WM_VSLID</td>
<td>26</td>
</tr>
<tr>
<td></td>
<td></td>
<td>WM_SIZED</td>
<td>27</td>
</tr>
<tr>
<td></td>
<td></td>
<td>WM_MOVED</td>
<td>28</td>
</tr>
<tr>
<td></td>
<td></td>
<td>WM_UNTOPPED</td>
<td>30</td>
</tr>
<tr>
<td></td>
<td></td>
<td>WM_ONTOP</td>
<td>31</td>
</tr>
<tr>
<td></td>
<td></td>
<td>WM_BOTTOM</td>
<td>33</td>
</tr>
<tr>
<td></td>
<td></td>
<td>WM_ICONIFY</td>
<td>34</td>
</tr>
<tr>
<td></td>
<td></td>
<td>WM_UNICONIFY</td>
<td>35</td>
</tr>
<tr>
<td></td>
<td></td>
<td>WM_ALLICONIFY</td>
<td>36</td>
</tr>
<tr>
<td></td>
<td></td>
<td>WM_TOOLBAR</td>
<td>37</td>
</tr>
<tr>
<td></td>
<td></td>
<td>AC_OPEN</td>
<td>40</td>
</tr>
<tr>
<td></td>
<td></td>
<td>AC_CLOSE</td>
<td>41</td>
</tr>
<tr>
<td></td>
<td></td>
<td>AP_TERM</td>
<td>50</td>
</tr>
<tr>
<td></td>
<td></td>
<td>AP_TFAIL</td>
<td>51</td>
</tr>
<tr>
<td></td>
<td></td>
<td>AP_RESCHG</td>
<td>57</td>
</tr>
<tr>
<td></td>
<td></td>
<td>SHUT_COMPLETED</td>
<td>60</td>
</tr>
<tr>
<td></td>
<td></td>
<td>RESCH_COMPLETED</td>
<td>61</td>
</tr>
<tr>
<td></td>
<td></td>
<td>AP_DRAGDROP</td>
<td>63</td>
</tr>
<tr>
<td></td>
<td></td>
<td>SH_WDRAW</td>
<td>72</td>
</tr>
<tr>
<td></td>
<td></td>
<td>CH_EXIT</td>
<td>90</td>
</tr>
</tbody>
</table>

msg[1] The application identifier of the sending application.
Any valid ap_id.

msg[2] The length of the message beyond 16 bytes (use appl_read() to read the excess).
Currently all system messages return 0 in this slot. Only user-defined messages utilize a higher value.
Each system message can be interpreted as follows:

<table>
<thead>
<tr>
<th>Message</th>
<th>Extended Information</th>
</tr>
</thead>
</table>
| MN_SELECTED     | A menu item has been selected by the user. msg[3] contains the object number of the menu title and msg[4] contains the object number of the menu item.  

As of AES version 4.0 (and when indicated by appl_getinfo()), msg[5] and msg[6] contain the high and low word, respectively, of the object tree of the menu item. msg[7] contains the parent object index of the menu item.  |
| WM_REDRAW       | This message alerts an application that a portion of the screen needs to be redrawn. msg[3] contains the handle of the window to redraw. msg[4-7] are the x, y, w, and h respectively of the ‘dirtied’ area.  

When the message is received the window contents should be drawn (or a representative icon if the window is iconified).  |
| WM_TOPPED       | This message is sent when an application window which is currently not the top window is clicked on by the user. msg[3] contains the handle of the window.  

You should use wind_set( handle, WF_TOP, msg[3], 0, 0, 0) to actually cause the window to be topped.  |
| WM_CLOSED       | This message is sent when the user clicks on a windows’ close box. msg[3] contains the handle of the window to close.  

You should react to this message with wind_close().  |
| WM_FULLED       | This message is sent when the user clicks on a windows’ full box. If the window is not at full size, the window should be resized using wind_set(handle, WF_CURRXYWH,... to occupy the entire screen minus the menu bar (see wind_set()).  

If the window was previously ‘fulled’ and has not been resized since, the application should return the window to its previous size.  |
**WM_ARROWED**

This message is sent to inform an application that one of its slider gadgets has been clicked on.

A row or column message is sent when a slider arrow is selected. A ‘page’ message is sent when a darkened area of the scroll bar is clicked. This usually indicates that the application should adjust the window’s contents by a larger amount than with the row or column messages.

$msg[3]$ indicates which action was actually selected as follows:

<table>
<thead>
<tr>
<th>Name</th>
<th>Value</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>WA_UPPAGE</td>
<td>0</td>
<td>Page Up</td>
</tr>
<tr>
<td>WA_DNPAGE</td>
<td>1</td>
<td>Page Down</td>
</tr>
<tr>
<td>WA_UPLINE</td>
<td>2</td>
<td>Row Up</td>
</tr>
<tr>
<td>WA_DNLINE</td>
<td>3</td>
<td>Row Down</td>
</tr>
<tr>
<td>WA_LFPAGE</td>
<td>4</td>
<td>Page Left</td>
</tr>
<tr>
<td>WA_RTPAGE</td>
<td>5</td>
<td>Page Right</td>
</tr>
<tr>
<td>WA_LFLINE</td>
<td>6</td>
<td>Column Left</td>
</tr>
<tr>
<td>WA_RTLINE</td>
<td>7</td>
<td>Column Right</td>
</tr>
</tbody>
</table>

**WM_HSLID**

This message indicates that the horizontal slider has been moved. $msg[3]$ contains the new slider position ranging from 0 to 1000.

Note: Slider position is relative and not related to slider size.

**WM_VSLID**

This message indicates that the vertical slider has been moved. $msg[3]$ contains the new slider position ranging from 0 to 1000.

Note: Slider position is relative and not related to slider size.

**WM_SIZED**

This message occurs when the user drags the window sizing gadget. $msg[3]$ contains the window handle. $msg[4-7]$ indicate the x, y, w, and h respectively of the new window location.

Use `wind_set(handle, WF_CURRXYWH, ...)` to actually size the window.

**WM_SIZED** and **WM_MOVED** usually share common handling code.

**WM_MOVED**

This message occurs when the user moves the window by dragging the windows’ title bar. $msg[3]$ contains the handle of the window being moved. $msg[4-7]$ indicate the x, y, w, and h respectively of the new window location.

Use `wind_set(handle, WF_CURRXYWH, ...)` to actually move the window.

**WM_MOVED** and **WM_SIZED** usually share common handling code.

**WM_UNTOPPED**

This message is sent when the current window is sent behind one or more windows as the result of another window being topped. $msg[3]$ contains the handle of the window being untopped.

The application need take no action. The message is for informational use only.
<table>
<thead>
<tr>
<th>Event</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>WM_ONTOP</strong></td>
<td>This message is sent when an applications’ window is brought to the front on a multitasking AES. <em>msg</em>[3] is the handle of the window being brought to the front. This message requires no action, it is for informational purposes only.</td>
</tr>
<tr>
<td><strong>WM_BOTTOM</strong></td>
<td>This message is sent when the user shift-clicks on the window’s (specified in <em>msg</em>[3]) mover bar to indicate that the window should be sent to the bottom of the window stack by using wind_set() with a parameter of WF_BOTTOM.</td>
</tr>
<tr>
<td><strong>WM_ICONIFY</strong></td>
<td>This message is sent when the user clicks on the SMALLER window gadget. <em>msg</em>[3] indicates the handle of the window to be iconified. <em>msg</em>[4-7] indicate the x, y, w, and h of the iconified window. If the iconified window represents a single window this message should be responded to by using wind_set() with a parameter of WF_ICONIFY.</td>
</tr>
<tr>
<td><strong>WM_UNICONIFY</strong></td>
<td>This message is sent when the user double-clicks on an iconified window. <em>msg</em>[3] indicates the handle of the window to be iconified. <em>msg</em>[4-7] indicate the x, y, w, and h of the original window. This message should be responded to by using wind_set() with a parameter of WF_UNICONIFY.</td>
</tr>
<tr>
<td><strong>WM_ALLICONIFY</strong></td>
<td>This message is sent when the user CTRL-clicks on the SMALLER window gadget. <em>msg</em>[3] indicates which window’s gadget was clicked. <em>msg</em>[4-7] indicates the position at which the new iconified window should be placed. The application should respond to this message by closing all open windows and opening a new iconified window at the position indicated which represents the application.</td>
</tr>
<tr>
<td><strong>WM_TOOLBAR</strong></td>
<td>This message is sent when a toolbar object is clicked. <em>msg</em>[3] contains the handle of the window containing the toolbar. <em>msg</em>[4] contains the object index of the object clicked. <em>msg</em>[5] contains the number of clicks. <em>msg</em>[6] contains the state of the keyboard shift keys at the time of the click (as in evnt_keybd()).</td>
</tr>
<tr>
<td><strong>AC_OPEN</strong></td>
<td>This message is sent when the user has selected a desk accessory to open. <em>msg</em>[4] contains the application identifier (as returned by appl_init()) of the accessory to open.</td>
</tr>
<tr>
<td><strong>AC_CLOSE</strong></td>
<td>This message is sent to a desk accessory when the accessory should be closed. <em>msg</em>[3] is the application identifier (as returned by appl_init()) of the accessory to close. Do not close any windows your accessory had open, the system will do this for you. Also, do not require any feedback from the user when this is received. Treat this message as a ‘Cancel’ from the user.</td>
</tr>
</tbody>
</table>
This message is sent when the system requests that the application terminate. This is usually the result of a resolution change but may also occur if another application sends this message to gain total control of the system.

The application should shut down immediately after closing windows, freeing resources, etc... msg[5] indicates the reason for the shut down as follows:

- **AP_TERM** (50) = Just shut down.
- **AP_RESCHG** (57) = Resolution Change.

If for some reason, your process can not shut down you must inform the AES by sending an **AP_TFAIL** (51) message by using `shel_write()` mode 10 (see `shel_write()`).

Note: Desk Accessories will always be sent **AC_CLOSE** messages, not **AP_TERM**.

**AP_TFAIL**

This message should be sent to the system (see `shel_write()`) when an application has received an **AP_TERM** (50) message and cannot shut down.

`msg[0]` should contain **AP_TFAIL** and `msg[1]` should contain the application error code.

**AP_RESCHG**

This message is actually a sub-command and is only found as a possible value in the **AP_TERM** (50) message (see above).

**SHUT_COMPLETED**

This message is sent to the application which requested a shutdown when the shutdown is complete and was successful.

**RESCH_COMPLETED**

This message is sent to an application when a resolution change it requested is completed. `msg[3]` contains 1 if the resolution change was successful and 0 if an error occurred.

**AP_DRAGDROP**

This message indicates that another application wishes to initiate a drag and drop session. `msg[3]` indicates the handle of the window which had an object dropped on it or -1 if no specific window was targeted.

`msg[4-5]` contains the X and Y position of the mouse when the object was ‘dropped’. `msg[6]` indicates the keyboard shift state at the time of the drop (as in `evnt_keybd()`).

`msg[7]` is a two-byte ASCII packed pipe identifier which gives the file extension of the pipe to open.

For more information about the drag & drop protocol, see Chapter 2: **GEMDOS**.

**SH_WDRAW**

This message is sent to the Desktop to ask it to update an open drive window. `msg[3]` should contain the drive number to update (0 = A:, 1 = B:) or -1 to update all windows.

**CH_EXIT**

This message is sent when a child process that the application has started, returns. `msg[3]` contains the child’s application identifier and `msg[4]` contains its exit code.

**VERSION NOTES**

**WM_UNTopped, WM_ONtop, AP_TERM, AP_TFAIL, AP_RESCHG, SHUT_COMPLETED, RESCH_COMPLETED, and CH_EXIT** are new as of
AES version 4.0.

WM_BOTTOM, WM_ICONIFY, WM_UNICONIFY, WM_ALLICONIFY, and WM_TOOLBAR are new as of AES version 4.1.

No lower version AES will send these messages.

The existence (or acceptance) of these messages should also be checked for by using appl_getinfo().

SEE ALSO evnt_multi()

---

**evnt_mouse()**

WORD evnt_mouse(flag, x, y, w, h, mx, my, button, kstate)

WORD flag, x, y, w, h;

WORD *mx, *mx, *button, *kstate;

`evnt_mouse()` releases control to the operating system until the mouse enters or leaves a specified area of the screen.

**OPCODE** 22 (0x16)

**AVAILABILITY** All AES versions.

**PARAMETERS** `flag` specifies the event to wait for as follows:

<table>
<thead>
<tr>
<th>Name</th>
<th>Value</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>MO_ENTER</td>
<td>0</td>
<td>Wait for mouse to enter rectangle.</td>
</tr>
<tr>
<td>MO_LEAVE</td>
<td>1</td>
<td>Wait for mouse to leave rectangle.</td>
</tr>
</tbody>
</table>

The rectangle to watch is specified in `x, y, w, h, mx` and `my` are WORD pointers which will be filled in with the final position of the mouse.

`button` is a WORD pointer which will be filled in upon return with the final state of the mouse button as defined in `evnt_button()`.

`kstate` is a WORD pointer which will be filled in upon return with the final state of the keyboard shift keys as defined in `evnt_button()`.

**BINDING**

```c
intin[0] = flag;
intin[1] = x;
intin[2] = y;
intin[3] = w;
intin[4] = h;
```
evnt_multi()

WORD evnt_multi( events, bclicks, bmask, bstate, m1flag, m1x, m1y, m1w, m1h, m2flag, m2x, m2y, m2w, m2h, msg, locount, hicount, mx, my, ks, kc, mc )

This call combines the functionality of evnt_button(), evnt_keybd(), evnt_message(), evnt_mouse(), and evnt_timer() into one call.

This call is usually the cornerstone of all GEM applications that must process system events.

Opcode 25 (0x19)

Availability All AES versions.

Parameters events is a bit mask which tells the function which events your application is interested in. You should logically ‘OR’ any of the following values together:

<table>
<thead>
<tr>
<th>Name</th>
<th>Mask</th>
<th>Function</th>
</tr>
</thead>
<tbody>
<tr>
<td>MU_KEYBD</td>
<td>0x01</td>
<td>Wait for a user keypress.</td>
</tr>
<tr>
<td>MU_BUTTON</td>
<td>0x02</td>
<td>Wait for the specified mouse button state.</td>
</tr>
<tr>
<td>MU_M1</td>
<td>0x04</td>
<td>Wait for a mouse/rectangle event as specified.</td>
</tr>
<tr>
<td>MU_M2</td>
<td>0x08</td>
<td>Wait for a mouse/rectangle event as specified.</td>
</tr>
</tbody>
</table>

Return Value The return value of this function is reserved. Currently it always returns 1.

Comments The evnt_multi() function can be used to watch two mouse/rectangle events as opposed to one.

See also evnt_multi()
**MU_MESAG** 0x10  |  Wait for a message.
---|---
**MU_TIMER** 0x20  |  Wait the specified amount of time.

For usage of `bclicks`, `bmask`, `bstate`, `mx`, `my`, `kc`, and `ks`, you should consult `evnt_button()`.

For usage of `m1flag`, `m1x`, `m1y`, `m1w`, `m1h`, `m2flag`, `m2x`, `m2y`, `m2w`, and `m2h`, consult `evnt_mouse()`.

For usage of `msg`, see `evnt_mesag()`.

For usage of `locount` and `hicount`, see `evnt_timer()`.

**BINDING**

```c
int in[0] = events;
int in[1] = bclicks;
int in[2] = bmask;
int in[3] = bstate;
int in[4] = m1flag;
int in[5] = m1x;
int in[6] = m1y;
int in[7] = m1w;
int in[8] = m1h;
int in[9] = m2flag;
int in[10] = m2x;
int in[11] = m2y;
int in[12] = m2w;
int in[13] = m2h;
int in[14] = locount;
int in[15] = hicount;

addrin[0] = msg;

crys_if(0x19);

*mx = intout[1];
*my = intout[2];
*mb = intout[3];
*ks = intout[4];
*kc = intout[5];
*mc = intout[6];

return intout[0];
```

**RETURN VALUE**
The function returns a bit mask of which events actually happened as in `events`. This may be one or more events and your application should be prepared to handle each.

**VERSION NOTES**
The only facet of `evnt_multi()` which has changed from AES version 4.0 is that which relates to `evnt_mesag()`. For further information you should consult that section.

**CAVEATS**
Under TOS 1.0, calling this function from a desk accessory with the **MU_TIMER**
mask and \textit{locount} and \textit{hicount} being equal to 0 could hang the system.

\textbf{SEE ALSO} \textit{evnt_button()}, \textit{evnt_keybd()}, \textit{evnt_mesag()}, \textit{evnt_mouse()}, \textit{evnt_timer()}

\textbf{evnt_timer()}

\textbf{WORD \textit{evnt_timer( locount, hicount )}}
\textbf{WORD \textit{locount, hicount;}}

\textit{evnt_timer()} releases control to the operating system until a specified amount of time has passed.

\textbf{OPCODE} \hfill 24 (0x18)
\textbf{AVAILABILITY} \hfill All AES versions.
\textbf{PARAMETERS} \hfill \textit{locount} is the low word of a 32-bit time value specified in milliseconds.
\hfill \textit{hicount} is the high portion of that 32-bit value.
\textbf{BINDING} \hfill \texttt{intin[0] = locount;}
\hfill \texttt{intin[1] = hicount;}
\hfill \texttt{return crys_if(0x18);}
\textbf{RETURN VALUE} \hfill The return value is reserved and is currently always 1.
\textbf{CAVEATS} \hfill Under \textit{TOS} 1.0, calling this function from a desk accessory with both parameters having a value of 0 will hang the system.
\textbf{COMMENTS} \hfill This function should not be relied on as an accurate clock. The time specified is used as a minimum time value only and the function will return at some point after that duration has passed.

\textbf{SEE ALSO} \textit{evnt_multi()}
The Form Library contains utility functions for the use and control of dialog boxes, alert boxes, and user input. The members of the Form Library are:

- form_alert()
- form_button()
- form_center()
- form_dial()
- form_do()
- form_error()
- form_keybd()
form_alert()

WORD form_alert( default, alertstr )
WORD default;
CHAR *alertstr;

form_alert() displays a standardized alert box and returns the user’s selection.

Opcode 52 (0x34)

Availability All AES versions.

Parameters default contains the number of the exit button which is to be made default (1-3).
alertstr contains a formatted string as follows: “[#][Alert Text][Buttons]”.

# specifies the icon to display in the alert as follows:

<table>
<thead>
<tr>
<th>#</th>
<th>Icon Displayed</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>No Icon</td>
</tr>
<tr>
<td>1</td>
<td>!</td>
</tr>
<tr>
<td>2</td>
<td>?</td>
</tr>
<tr>
<td>3</td>
<td>STOP</td>
</tr>
<tr>
<td>4</td>
<td>i</td>
</tr>
<tr>
<td>5</td>
<td></td>
</tr>
</tbody>
</table>

‘Alert Text’ is a text string of as many as 5 lines composed of up to 30 characters each. Each line is separated by a ‘|’ character.

‘Buttons’ is a text string to define as many as 3 buttons up to 10 characters each. If only one button is used, its text may be as long as 30 characters. Again, each button is separated by a ‘|’ character.
**BINDING**

```c
intin[0] = default;
addrin[0] = alertstr;
return crys_if(0x34);
```

**RETURN VALUE**

`form_alert()` returns a `WORD` indicating which button was used to exit by the user (A possible value of 1-3).

**VERSION NOTES**

Icons #4-5 are only available as of AES version 4.1.

**CAVEATS**

Several versions of the AES have special quirks related to this function. By following the guidelines below you should avoid any difficulty:

1. All AES versions below 1.06 have some difficulty formatting alert strings padded with spaces. If you want your alerts to look right on all AES versions, do not pad any button or line with spaces with the exception below.

2. Add one space to the end of the longest text line on an alert. This will prevent the right edge from touching the border in some AES versions.

---

### form_button()

**WORD form_button( tree, obj, clicks, newobj )**

**OBJECT *tree;**

**WORD obj, clicks, newobj;**

`form_button()` is a utility function designed to aid in the creation of a custom `form_do()` handler.

**OPCODE**

56 (0x38)

**AVAILABILITY**

All AES versions.

**PARAMETERS**

`tree` is a pointer to a valid object tree in memory you wish to process button events for. `obj` is the object index into `tree` which was clicked on and which needs to be processed.

`clicks` is the number of times the mouse button was clicked.

`newobj` returns the next object to gain edit focus or 0 if there are no editable objects. If the top bit of `newobj` is set, this indicates that a TOUCHEXIT object was double-clicked.

**BINDING**

```c
intin[0] = obj;
intin[1] = clicks;
```
addrin[0] = tree;
crys_if(0x38);
*newobj = intout[1];
return intout[0];

RETURN VALUE
form_button() returns a 0 if it exits finding an EXIT or TOUCHEXIT object selected or 1 otherwise.

COMMENTS
To use this function properly, the application should take the following steps:

1. Monitor mouse clicks with evnt_multi() or evnt_button().
2. When a click occurs, use objc_find() to determine if the click occurred over the object.
3. If so, call form_button() with the appropriate values.

This function was not originally documented by Atari. You may have to add bindings for this function to some earlier ‘C’ compilers.

SEE ALSO
form_do(), form_keybd()

form_center()

WORD form_center( tree, x, y, w, h )
OBJECT *tree;
WORD *x, *y, *w, *h;

form_center() is used to modify an object’s coordinates so that it will appear in the center of the display screen.

OPCODE
54 (0x36)

AVAILABILITY
All AES versions.

PARAMETERS
tree points to a valid OBJECT structure (see discussion of resources) which the application wishes to have centered. x, y, w, and h, return a clipping rectangle suitable for use in objc_draw().

BINDING
addrin[0] = tree;
crys_if(0x36);
*x = intout[1];
*y = intout[2];
*w = intout[3];
*h = intout[4];
return intout[0];

**RETURN VALUE**
The return value is currently reserved. Currently it equals 1.

**COMMENTS**
The values that *form_center()* returns in x, y, w, and h, are not necessarily the same as the object’s. These values take into account negative borders, outlining, and shadowing. This is meant to provide a suitable clipping rectangle for *objc_draw()*

**SEE ALSO**
objc_draw()

---

**form_dial()**

WORD form_dial( *mode, x1, y1, w1, h1, x2, y2, w2, h2 *)

WORD *mode, x1, y1, w1, h1, x2, y2, w2, h2;

*form_dial()* is used to reserve and release screen space for dialog usage. In addition, it also optionally provides grow/shrink box effects.

**OPCODE**
51 (0x33)

**AVAILABILITY**
All AES versions.

**PARAMETERS**
*mode* specifies the action to take and the meaning of remaining parameters as follows:

<table>
<thead>
<tr>
<th>Name</th>
<th>#</th>
<th>Action</th>
</tr>
</thead>
<tbody>
<tr>
<td>FMD_START</td>
<td>0</td>
<td>This mode reserves the screen space for a dialog. x2, y2, w2, and h2, contain the coordinates of the dialog to be used (usually obtained through <em>form_center()</em>).</td>
</tr>
<tr>
<td>FMD_GROW</td>
<td>1</td>
<td>This mode draws an expanding box from the coordinates specified in x1, y1, w1, and h1 to the coordinates specified in x2, y2, w2, and h2. This call is optional and is not required to display a dialog.</td>
</tr>
<tr>
<td>FMD_SHRINK</td>
<td>2</td>
<td>This mode draws a shrinking box from the coordinates specified in x2, y2, w2, and h2 to the coordinates specified in x1, y1, w1, and h1. This call is optional and is not required to display a dialog.</td>
</tr>
<tr>
<td>FMD_FINISH</td>
<td>3</td>
<td>This mode releases the screen space for a dialog (previously reserved with mode 0). x2, y2, w2, and h2 contain the coordinates of the space to release. One of the side-effects of this call is a WM_REDRAW message sent to any window which the dialog was covering.</td>
</tr>
</tbody>
</table>
form_do()

WORD form_do( tree, editobj )
OBJECT *tree;
WORD editobj;

form_do() provides an automated dialog handling function to the calling application. It suspends program control, handling all radio buttons, selectable objects, etc... until an object with the TOUCHEXIT or EXIT flag is selected.

_OPCODE_ 50 (0x32)

_AVAILABILITY_ All AES versions.

_PARAMETERS_ tree is a pointer to a valid object tree (see the discussion on objects in this chapter) which contains a dialog with at least one EXIT or TOUCHEXIT button or object.

editobj is the object index into tree which specifies the desired initial location of the edit cursor (the object must be flagged as EDITABLE). If the form has no text editable fields, you should use 0.

_BINDING_ intin[0] = editobj;
addrin[0] = tree;
return crys_if(0x32);

RETURN VALUE form_do() returns the object index of the EXIT or TOUCHEXIT button which
was selected. If the object was double clicked, bit 15 will be set. This means that to obtain the actual object number you should mask off the result with 0x7FFF.

---

**form_error()**

WORD form_error( error )

WORD error;

form_error() displays a pre-defined error alert box to the user.

**OPCODE**

53 (0x35)

**AVAILABILITY**

All AES versions.

**PARAMETERS**

error specifies a MS-DOS error code as follows:

<table>
<thead>
<tr>
<th>Name</th>
<th>GEMDOS Error #</th>
<th>error</th>
<th>Message</th>
</tr>
</thead>
<tbody>
<tr>
<td>FERR_FILENOTFOUND</td>
<td>-33</td>
<td>2</td>
<td>File Not Found</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>The application can not find the folder or file that you tried to access.</td>
</tr>
<tr>
<td>FERR_PATHNOTFOUND</td>
<td>-34</td>
<td>3</td>
<td>Path Not Found</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>The application cannot find the folder or file that you tried to access.</td>
</tr>
<tr>
<td>FERR_NOHANDLES</td>
<td>-35</td>
<td>4</td>
<td>No More File Handles</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>The application does not have room to open another document. To make room, close any open document that you do not need.</td>
</tr>
<tr>
<td>FERR_ACCESSDENIED</td>
<td>-36</td>
<td>5</td>
<td>Access Denied</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>An item with this name already exists in the directory, or this item is set to read-only status.</td>
</tr>
<tr>
<td>FERR_LOWMEM</td>
<td>-39</td>
<td>8</td>
<td>Insufficient Memory</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>There is not enough memory for the application you just tried to run.</td>
</tr>
<tr>
<td>FERR_BADENVIRON</td>
<td>-41</td>
<td>10</td>
<td>Invalid Environment</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>There is not enough memory for the application you just tried to run.</td>
</tr>
<tr>
<td>FERR_BADFORMAT</td>
<td>-42</td>
<td>11</td>
<td>Invalid Format</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>There is not enough memory for the application you just tried to run.</td>
</tr>
</tbody>
</table>
The GEMDOS error number can be translated into a MS-DOS code by subtracting 31 from the absolute value of the error code.

**BINDING**

```c
int in[0] = error;
return crys_if(0x35);```

**RETURN VALUE**
The function returns the exit button clicked as in form_alert(). It is, however, insignificant as all of the error alerts have only one button.

**CAVEATS**
Not every GEMDOS error code has a matching alert box.

**SEE ALSO**
form_alert()

---

### form_keybd()

WORD form_keybd( tree, obj, nextobj, kc, newobj, keyout )  
OBJECT *tree;  
WORD obj, nextobj, kc;  
WORD *newobj, *keyout;

form_keybd() processes keyboard input for dialog box control. It handles special keys such as return, escape, tab, etc... It is only of real use if you are writing a customized form_do() routine.

**OPCODE**
55 (0x37)

**AVAILABILITY**
All AES versions.

**PARAMETERS**
- `tree` points to a valid OBJECT tree containing the dialog you wish to process. `obj` is the object index of the object which currently has edit focus (0 if none). `nextobj` is reserved and should be 1.
kc is the value returned from `evnt_keybd()` or `evnt_multi()` which represents the keypresses’ scan code and ASCII value.

`newobj` is a `WORD` pointer which is filled in on function exit to be the new object with edit focus unless the RETURN key was pressed with a default object present in which case it equals the object index of the object that was the default.

`keyout` is the value ready to be passed on to `objc_edit()` if no processing was required or 0 if the key was processed and handled by the call.

**BINDING**

```c
intin[0] = obj;
intin[1] = nextobj;
intin[2] = kc;
addrin[0] = tree;
crys_if(0x37);
*newobj = intout[1];
*keyout = intout[2];
return intout[0];
```

**RETURN VALUE**

`form_keybd()` returns 0 if a default EXIT object was triggered by this call or 1 if the dialog should continue to be processed.

**COMMENTS**

This function was not originally documented by Atari. You may need to add bindings for this function into some older ‘C’ compilers.

**SEE ALSO**

`objc_edit()`, `form_do()`, `form_button()`
The File Selector Library contains two functions for displaying the system file selector (or currently installed alternate file selector) and prompting the user to select a file. The members of this library are:

- `f.sel_exinput()`
- `f.sel_input()`
fsel_exinput() displays the system file selector and offers the user an opportunity to choose a complete GEMDOS path specification.

**OPCODE**
91 (0x5B)

**AVAILABILITY**
Available from AES version 1.40.

**PARAMETERS**
- `path` should be a pointer to a character buffer at least 128 bytes long (applications wishing to access CD-ROM’s should allocate at least 200 bytes). On input the buffer should contain a complete GEMDOS path specification including a drive specifier, path string, and wildcard mask as follows: ‘drive:\path\mask’. The mask can be any valid GEMDOS wildcard (usually *.*).

On function exit, `path` contains final path of the selected file (you will have to strip the mask).

- `file` should point to a character buffer 13 bytes long (12 character filename plus NULL). On input its contents will be placed on the filename line of the selector (usually this value can simply be an empty string). On function exit, `file` contains the filename which the user selected.

- `button` is a short pointer which upon function exit will contain `FSELCANCEL` (0) if the user selected CANCEL or `FSEL_OK` (1) if OK.

- `title` should be a pointer to a character string up to 30 characters long which contains the title to appear in the file selector (usually indicates which action the user is about to take).

**BINDING**

```c
addrin[0] = path;
addrin[1] = file;
addrin[2] = label;
crys_if(0x5B);
*button = intout[1];
return intout[0];
```

**RETURN VALUE**
`fsel_exinput()` returns 0 if an error occurred and 1 otherwise.
VERSION NOTES
Some ‘C’ compilers (Lattice for example) provide a special function which allows `fsel_exinput()` to be used even on earlier AES versions.

COMMENTS
The path parameter to this function should be validated to ensure that the path actually exists prior to calling this function to prevent confusing the user.

This call should always be used as opposed to `fsel_input()` when it is available. Otherwise, the user has no reminder as to what function s/he is actually undertaking.

SEE ALSO
`fsel_input()`

---

**fsel_input()**

WORD `fsel_input(path, file, button)`

CHAR *`path, *file;`

WORD *`button;`

`fsel_input()` displays the system file selector and allows the user to select a valid GEMDOS path and file.

OPCODE
90 (0x5A)

AVAILABILITY
All AES versions.

PARAMETERS
All parameters are consistent with `fsel_exinput()` with the notable lack of title.

BINDING
```
addrin[0] = path;
addrin[1] = file;
crys_if(0x5A);
*button = intout[1];
return intout[0];
```

RETURN VALUE
`fsel_input()` returns a 0 if an error occurred or 1 otherwise.

COMMENTS
You should never use this function in place of `fsel_exinput()` when `fsel_exinput()` is available.

SEE ALSO
`fsel_exinput()`
The *Graphics Library* provides applications with a variety of utility functions which serve to provide common screen effects, mouse control, and the obtaining of basic screen attributes. The functions of the *Graphics Library* are as follows:

- `graf_dragbox()`
- `graf_growbox()`
- `graf_handle()`
- `graf_mkstate()`
- `graf_mouse()`
- `graf_movebox()`
- `graf_rubberbox()`
- `graf_shrinkbox()`
- `graf_slidebox()`
- `graf_watchbox()`
graf_dragbox()

`WORD graf_dragbox( w, h, sx, sy, bx, by, bw, bh, endx, endy )`
`WORD w, h, sx, sy, bx, by, bw, bh;`
`WORD *endx, *endy;`

`graf_dragbox()` allows the user to move a box frame within the constraints of a bounding rectangle. This call is most often used to give the user a visual ‘clue’ when an object is being moved on screen.

**OPCODE**

71 (0x47)

**AVAILABILITY**

All AES versions.

**PARAMETERS**

`w` and `h` specify the initial width and height of the box to draw. `sx` and `sy` specify the starting x and y screen coordinates.

`bx`, `by`, `bw`, and `bh`, give the coordinates of the bounding rectangle.

`endx` and `endy` are `WORD` pointers which, on function exit, will be filled in with the ending x and y position of the box.

**BINDING**

```c
intin[0] = w;
intin[1] = h;
intin[2] = sx;
intin[3] = sy;
intin[4] = bx;
intin[5] = by;
intin[6] = bw;
intin[7] = bh;

crys_if(0x47);

*endx = intout[1];
*endy = intout[2];

return intout[0];
```

**RETURN VALUE**

`graf_dragbox()` returns a 0 if an error occurred during execution or greater than zero otherwise.

**COMMENTS**

This call should be made only when the mouse button is depressed. The call returns when the mouse button is released.

**SEE ALSO**

`graf_slidebox()`
graf_growbox()

WORD graf_growbox(x1, y1, w1, h1, x2, y2, w2, h2)
WORD x1, y1, w2, h2, x2, y2, w2, h2;

graf_growbox() is used to provide a visual ‘clue’ to a user by animating an outline of a box from one set of coordinates to another. It is the complement function to graf_shrinkbox().

OPCODE 73 (0x49)

AVAILABILITY All AES versions.

PARAMETERS

x1, y1, w1, and h1 are the screen coordinates of the starting rectangle (where the outline will grow from).

x2, y2, w2, and h2 are the screen coordinates of the ending rectangle (where the outline will grow to).

BINDING

int in[0] = x1;
int in[1] = y1;
int in[2] = w1;
int in[3] = h1;
int in[4] = x2;
int in[5] = y2;
int in[6] = w2;
int in[7] = h2;

return crys_if(0x49);

RETURN VALUE graf_growbox() returns 0 if an error occured or non-zero otherwise.

CAVEATS There is currently no defined method of handling an error generated by this function.

COMMENTS This function is what is called by GEM’s form_dial(FMD_GROW,...

SEE ALSO form_dial(), graf_shrinkbox()

graf_handle()

WORD graf_handle(wcell, hcell, wbox, hbox);
WORD *wcell, *hcell, *wbox, *hbox;

graf_handle() returns important information regarding the physical workstation.
currently in use by the AES.

**Opcode** 77 (0x4D)

**Availability** All AES versions.

**Parameters**

- `wcell` and `hcell` are **WORD** pointers which on function exit will be filled in with the width and height, respectively, of the current system character set.

- `wbox` and `hbox` are **WORD** pointers which on function exit will be filled in with the width and height, respectively, of the minimum bounding box of a **BOXCHAR** character.

**Binding**

```c
crys_if(0x4D);
*charw = intout[1];
*charh = intout[2];
*boxw = intout[3];
*boxh = intout[4];
return intout[0];
```

**Return Value** This function returns the **VDI** handle for the current physical workstation used by the AES.

**Caveats** There is currently no defined method of handling an error generated by this function.

**Comments** The return value of this function is required to open a virtual screen workstation.

**See Also** `v_opnvwk()`

---

**graf_mkstate()**

WORD `graf_mkstate(mx, my, mb, ks)`

WORD `*mx, *my, *mb, *ks`;

`graf_mkstate()` returns information about the current state of the mouse pointer, buttons, and keyboard shift-key state.

**Opcode** 79 (0x4F)

**Availability** All AES versions.

**Parameters**

- `mx` and `my` are **WORD** pointers, which, on function exit will be filled in with the current x and y coordinates of the mouse pointer. `mb` is a **WORD** pointer, which,
on function exit will be filled in with the current button state of the mouse as defined in `evnt_button()`.

**BINDING**

```
crys_if(0x4F);

*mx = intout[1];
*my = intout[2];
*mb = intout[3];
*ks = intout[4];

return intout[0];
```

**RETURN VALUE**
The function return is currently reserved and currently equals 1.

**SEE ALSO**
`evnt_button()`, `vq_mouse()`

---

**graf_mouse()**

```
WORD graf_mouse( mode, formptr )
WORD mode;
VOIDP formptr;
```

`graf_mouse()` alters the appearance of the mouse form and can be used to hide and display the mouse pointer from the screen.

**OPCODE**

78 (0x4E)

**AVAILABILITY**

All AES versions.

**PARAMETERS**

`mode` is defined as follows:

<table>
<thead>
<tr>
<th>mode</th>
<th>#</th>
<th>Meaning</th>
<th>Shape</th>
</tr>
</thead>
<tbody>
<tr>
<td>ARROW</td>
<td>0</td>
<td>Change the current mouse cursor shape.</td>
<td>🍁</td>
</tr>
<tr>
<td>TEXT_CRSR</td>
<td>1</td>
<td>Change the current mouse cursor shape.</td>
<td>🎨</td>
</tr>
<tr>
<td>BUSY_BEE</td>
<td>2</td>
<td>Change the current mouse cursor shape.</td>
<td>🐣</td>
</tr>
<tr>
<td>POINT_HAND</td>
<td>3</td>
<td>Change the current mouse cursor shape.</td>
<td>🖐️</td>
</tr>
<tr>
<td>FLAT_HAND</td>
<td>4</td>
<td>Change the current mouse cursor shape.</td>
<td></td>
</tr>
<tr>
<td>---------------</td>
<td>---</td>
<td>--------------------------------------</td>
<td></td>
</tr>
<tr>
<td>THIN_CROSS</td>
<td>5</td>
<td>Change the current mouse cursor shape.</td>
<td></td>
</tr>
<tr>
<td>THICK_CROSS</td>
<td>6</td>
<td>Change the current mouse cursor shape.</td>
<td></td>
</tr>
<tr>
<td>OUTLN_CROSS</td>
<td>7</td>
<td>Change the current mouse cursor shape.</td>
<td></td>
</tr>
<tr>
<td>USER_DEF</td>
<td>255</td>
<td>Change the current mouse cursor shape.</td>
<td></td>
</tr>
<tr>
<td>M_OFF</td>
<td>256</td>
<td>Remove the mouse cursor from the screen.</td>
<td></td>
</tr>
<tr>
<td>M_ON</td>
<td>257</td>
<td>Display the mouse cursor.</td>
<td></td>
</tr>
<tr>
<td>M_SAVE</td>
<td>258</td>
<td>Save the current mouse form in an AES provided buffer. Check appl_getinfo() for the presence of this feature.</td>
<td></td>
</tr>
<tr>
<td>M_LAST</td>
<td>259</td>
<td>Restore the most recently saved mouse form. Check appl_getinfo() for the presence of this feature.</td>
<td></td>
</tr>
<tr>
<td>M_RESTORE</td>
<td>260</td>
<td>Restore the mouse form to its last shape. Check appl_getinfo() for the presence of this feature.</td>
<td></td>
</tr>
</tbody>
</table>

If `mode` is equal to USER_DEF, `formptr` must point to a MFORM structure as defined below (if `mode` is different than USER_DEF, `formptr` should be NULL):

```c
typedef struct {
    short mf_xhot;
    short mf_yhot;
    short mf_nplanes;
    short mf_fg;
    short mf_bg;
    short mf_mask[16];
    short mf_data[16];
} MFORM;
```

`mf_xhot` and `mf_yhot` are the location of the mouse ‘hot-spot’. These values should be in the range 0 to 15 and define what offset into the bitmap is actually the ‘point’.

`mf_nplanes` specifies the number of bit-planes used by the mouse pointer. Currently, the value of 1 is the only legal value.

`mf_fg` and `mf_bg` are the mask and data colors of the mouse specified as palette...
indexes. Usually these values will be 0 and 1 respectively.

*mf_mask* is an array of 16 *WORD’s* which define the mask portion of the mouse form. *mf_data* is an array of 16 *WORD’s* which define the data portion of the mouse form.

As of AES 4.0 and beyond, the **AES** may not allow a mouse form to change to benefit another application. If it is absolutely necessary for the application to display its mouse form, logically OR the mode parameter with **M_FORCE** (0x8000) and make the call.

This will force the **AES** to change to your mouse form. It should, however, be done within the scope of a **wind_update()** sequence.

**BINDING**

```c
intin[0] = mode;
addrin[0] = formptr;
return crys_if(0x4E);
```

**RETURN VALUE**

*graf_mouse()* returns a 0 if an error occurred or non-zero otherwise.

**CAVEATS**

There is currently no defined method of handling an error generated by this function.

**SEE ALSO**

*vsc_form()*

---

**graf_movebox()**

WORD *graf_movebox( bw, bh, sx, sy, ex, ey )*

**WORD bw, bh, sx, sy, ex, ey;**

*graf_movebox()* animates a moving box between two points on the screen. It is used to give the user a visual ‘clue’ to an action undertaken by the application.

**OPCODE**

72 (0x48)

**AVAILABILITY**

All AES versions.

**PARAMETERS**

*bw* and *bh* specify the width and height, respectively, of the box to animate. *sx* and *sy* specify the starting coordinates of the box. *ex* and *ey* specify the ending coordinates of the box.

**BINDING**

```c
intin[0] = bw;
intin[1] = bh;
intin[2] = sx;
```
intin[3] = sy;
intin[4] = ex;
intin[5] = ey;

return crys_if(0x48);

**RETURN VALUE**
The return value is 0 if an error occurred or non-zero otherwise.

**CAVEATS**
There is currently no defined method for handling an error generated by this call.

**COMMENTS**
Some older ‘C’ bindings referred to this call as **graf_mbox()**. If your compiler still uses this call you should update it.

---

**graf_rubberbox()**

**WORD graf_rubberbox( bx, by, minw, minh, endw, endh )**

**WORD bx, by, minw, minh;**
**WORD *endw, *endh;**

**graf_rubberbox()** allows the user to change the size of a box outline with a fixed starting point.

**OPCODE**
70 (0x46)

**AVAILABILITY**
All AES versions.

**PARAMETERS**
* bx and by define the fixed upper-left corner of the box to stretch or shrink.

* minw and minh specify the minimum width and height that the rectangle can be shrunk to.

* endw and endh are WORD pointers which will be filled in with the ending width and height of the box when the mouse button is released.

**BINDING**
intin[0] = bx;
intin[1] = by;
intin[2] = minw;
intin[3] = minh;

crys_if(0x46);

*endw = intout[1];
*endh = intout[2];

return intout[0];

**RETURN VALUE**
**graf_rubberbox()** returns 0 if an error occurred or non-zero otherwise.
C A V E A T S  
There is currently no defined method for handling an error generated by this call.

C O M M E N T S  
This function should only be entered when the user has depressed the mouse button as it returns when the mouse button is released.

S E E A L S O  
graf_dragbox(), graf_slidebox()

graf_shrinkbox()

WORD graf_shrinkbox( x1, y1, w1, h1, x2, y2, w2, h2 )

WORD x1, y1, w1, h1, x2, y2, w2, h2;

graf_shrinkbox() displays an animated box shrinking from one rectangle to another. It should be used to provide the user with a visual ‘clue’ to an action. It is the complement function to graf_growbox().

O P C O D E  
74 (0x4A)

A V A I L A B I L I T Y  
All AES versions.

P A R A M E T E R S  
x1, y1, w1, and h1 are the coordinates of the rectangle to shrink to.

x2, y2, w2, and h2 are the coordinates of the rectangle to shrink from.

B I N D I N G  
intin[0] = x1;
intin[1] = y1;
intin[2] = w1;
intin[3] = h1;
intin[4] = x2;
intin[5] = y2;
intin[6] = w2;
intin[7] = h2;

return crys_if(0x4A);

R E T U R N  V A L U E  
The function returns 0 if an error occurred or non-zero otherwise

C A V E A T S  
There is currently no defined method of handling an error from this call.

C O M M E N T S  
This function is essentially the same as form_dial(FMD_SHRINK,...

S E E A L S O  
form_dial(), graf_growbox()
**graf_slidebox()**

WORD graf_slidebox( tree, parent, obj, orient )
OBJECT *tree;
WORD parent, obj, orient;

`graf_slidebox()` allows the user to slide a child object within the bounds of its parent. It is often used to implement slider controls.

**Opcode**

76 (0x4C)

**Availability**

All AES versions.

**Parameters**

tree

`tree` is pointer to the object tree containing the child and parent objects.

parent

`parent` is the object index of an object which bounds the movement of the child.

child

`child` is the object index of the object which can be moved within the bounds of `parent`.

orient

`orient` specifies the orientation of the allowed movement. 0 is horizontal (left-right), 1 is vertical (up-down).

**Binding**

intin[0] = parent;
intin[1] = child;
intin[2] = orient;

addrin[0] = tree;
return crys_if(0x4C);

**Return Value**

The function returns a value specifying the relative offset of the child within the parent as a number between 0 and 1000.

**Comments**

This call can be used easily with sliders built into dialogs by making the slider bar a **TOUCHEXIT** and calling this function when it is clicked. This call should only be made when the mouse button is depressed as it returns when it is released.

**See Also**

`graf_movebox()`
**graf_watchbox()**

WORD graf_watchbox( tree, obj, instate, outstate )
OBJECT *tree;
WORD obj, instate, outstate;

*graf_watchbox()* modifies the given state of a specified object depending on whether the pointer is within the bounds of the object or outside the bounds of the object as long as the left mouse button is held down.

**OPCDE**

75 (0x4B)

**AVAILABILITY**

All AES versions.

**PARAMETERS**

tree is a pointer to the **ROOT** object of the tree which contains the object you wish to watch. *obj* is the object index of the object to watch.

instate is the *ob_state* (see *objc_change()*)) to apply while the mouse is inside of the bounds of the object.

outstate is the *ob_state* to apply while the mouse is outside of the bounds of the object.

**BINDING**

```c
intin[0] = obj;
intin[1] = instate;
intin[2] = outstate;
addrin[0] = tree;
```

```c
return crys_if(0x4B);
```

**RETURN VALUE**

*graf_watchbox()* returns a 0 if the mouse button was released outside of the object or a 1 if the button was released inside of the object.

**COMMENTS**

As this call returns when the mouse button is released, it should only be made when the mouse button is depressed. This call is used internally by *form_button()* and *form_do()* and is usually only necessary if you are replacing one of these handlers.

**SEE ALSO**

*form_button()*
The *Menu Library* assists in the handling of system menu bars and popup menus. In addition, individual control of menu items can also be handled through these functions. The members of the *Menu Library* are:

- `menu_attach()`
- `menu_bar()`
- `menu_icleck()`
- `menu_ienable()`
- `menu_istart`
- `menu_popup()`
- `menu_register()`
- `menu_settings()`
- `menu_text()`
- `menu_tnormal()`
menu_attach()

WORD menu_attach(flag, tree, item, mdata )

WORD flag;
OBJECT *tree;
WORD item;
MENU *mdata;

menu_attach() allows an application to attach, change, or remove a sub-menu. It also allows the application to inquire information regarding a currently defined sub-menu.

O P C O D E

37 (0x25)

A V A I L A B I L I T Y

This function is only available from AES version 3.30 and above. In AES versions 4.0 and greater, appl_getinfo() should be used to determine its exact functionality.

P A R A M E T E R S

flag indicates the action the application desires as follows:

<table>
<thead>
<tr>
<th>#</th>
<th>Define</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>ME_INQUIRE</td>
<td>Return information on a sub-menu attached to the menu item designated by tree and item in mdata.</td>
</tr>
<tr>
<td>1</td>
<td>ME_ATTACH</td>
<td>Attach or change a sub-menu. mdata should be initialized by the application. tree and item should be the OBJECT pointer and index to the menu which is to have the sub-menu attached. If mdata is NULLPTR, any sub-menu attached will be removed.</td>
</tr>
<tr>
<td>2</td>
<td>ME_REMOVE</td>
<td>Remove a sub-menu. tree and item should be the OBJECT pointer and index to the menu item which a sub-menu was attached to. mdata should be NULLPTR.</td>
</tr>
</tbody>
</table>

In all cases except ME_REMOVE, mdata should point to a MENU structure as defined here:

typedef struct {
    OBJECT  *mn_tree;
    WORD    mn_menu;
    WORD    mn_item;
    WORD    mn_scroll;
    WORD    mn_keystate;
} MENU;

The MENU structure members are defined as follows:
### Member | Meaning
--- | ---
\texttt{mn\_tree} | Points to the \texttt{OBJECT} tree of the sub-menu.
\texttt{mn\_menu} | Is an index to the parent object of the menu items.
\texttt{mn\_item} | Is the starting menu item.
\texttt{mn\_scroll} | If \texttt{SCROLL\_NO} (0), the menu will not scroll. If \texttt{SCROLL\_YES} (1), and the number of menu items exceed the menu scroll height, arrows will appear which allow the user to scroll selections.
\texttt{mn\_keystate} | This member is unused and should be 0 for this call.

#### BINDING

\begin{verbatim}
intin[0] = flag;
intin[1] = item;
addrin[0] = tree;
addrin[1] = mdata;
return crys_if(0x25);
\end{verbatim}

#### RETURN VALUE

\texttt{menu\_attach()} returns 0 if an error occurred and the sub-menu could not be attached or 1 if the operation was successful.

#### CAVEATS

AES versions supporting \texttt{menu\_attach()} less than 4.1 contain a bug which causes the AES to crash when changing or removing a sub-menu attachment.

At present, if you wish to attach a scrolling menu, the menu items must be \texttt{G\_STRING\_S}.

#### COMMENTS

If a menu bar having attachments is removed with \texttt{menu\_bar( NULL, MENU\_REMOVE )} those attachments are removed by the system and must be reattached with this call if the menu is redisplayed at a later time.

Several recommendations regarding sub-menus should be adhered to:

1. Menu items which will have sub-menus attached to them should be padded with blanks to the end of the menu.
2. Menu items which will have sub-menus attached to them should not have a keyboard equivalent.
3. Sub-menus will display faster if a byte-boundary is specified.
4. Sub-menus will be shifted vertically to align the start object with the main menu item which it is attached to.
5. Sub-menus will always be adjusted to automatically fit on the screen.
6. There can be a maximum of 64 sub-menu attachments per process (attaching a sub-menu to more than one menu item counts as only one attachment).
7. Do not attach a sub-menu to itself.
8. As a user-interface guideline, there should only be one level of sub-menus, though it is possible to have up to four levels currently.
9. \texttt{menu\_istart()} works only on sub-menus attached with \texttt{menu\_attach()}.
**menu_bar()**

WORD menu_bar( tree, mode )

OBJECT *tree;

WORD mode;

**menu_bar()** displays a specialized **OBJECT** tree on the screen as the application menu. It can also be used to determine the owner of the currently displayed menu bar in a multitasking **AES**.

**OPCODE**

30 (0x1E)

**AVAILABILITY**

All AES versions.

**PARAMETERS**

*tree* is a pointer to an **OBJECT** tree which has been formatted for use as a system menu (for more information on the **OBJECT** format of a menu see the discussion on objects in this chapter).

*mode* is a flag indicating the action to take as follows:

<table>
<thead>
<tr>
<th>Name</th>
<th>mode</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>MENU_REMOVE</td>
<td>0</td>
<td>Erase the menu bar specified in <em>tree</em>.</td>
</tr>
<tr>
<td>MENU_INSTALL</td>
<td>1</td>
<td>Display the menu bar specified in <em>tree</em>.</td>
</tr>
<tr>
<td>MENU_INQUIRE</td>
<td>-1</td>
<td>Return the <strong>AES</strong> application identifier of the process which owns the currently displayed system menu. <em>tree</em> can be set to <strong>NULL</strong>. The <strong>AES</strong> version must be greater than 4.0 and <strong>appl_getinfo()</strong> must indicate that this feature is supported.</td>
</tr>
</tbody>
</table>

**BINDING**

intin[0] = mode;

addrin[0] = tree;

return crys_if(0x1E);

**RETURN VALUE**

If *mode* is **MENU_REMOVE** (0) or **MENU_INSTALL** (1), the return value indicates an error condition where >0 means no error and 0 means an error occurred. In inquiry mode (*mode = **MENU_INQUIRE** (-1)), **menu_bar()** returns the application identified of the process which owns the currently displayed menu bar.

**COMMENTS**

The safest way to redraw an application’s menu bar is to redraw it only if you are
sure it is currently the active menu bar. In a non-multitasking AES, this is a certainty, however, in a multitasking AES you should first inquire the menu bar’s owner within the scope of a wind_update( BEG_UPDATE ) call to prevent the system from swapping active menu bars while in the process of redrawing.

**See Also**

menu_ienable(), menu_icheck()

---

### menu_icheck()

**WORD menu_icheck( tree, obj, check )**

**OBJECT *tree;**

**WORD obj, check;**

*menu_icheck() adds/removes a checkmark in front of a menu item.*

#### OPCODE

31 (0x1F)

#### AVAILABILITY

All AES versions.

#### PARAMETERS

*tree* specifies the object tree of the current menu. *obj* should be the object index of a menu item. If *check* is UNCHECK (0), no checkmark will be displayed next to this item whereas if *check* is CHECK (1), a checkmark will be displayed.

#### BINDING

```c
intin[0] = obj;
intin[1] = check;

addrin[0] = obj;
return crys_if(0x1F);
```

#### RETURN VALUE

*menu_icheck()* returns 0 if an error occurred or non-zero otherwise.

**See Also**

objc_change()

---

### menu_ienable()

**WORD menu_ienable( tree, obj, flag )**

**OBJECT *tree;**

**WORD obj, flag;**

*menu_ienable() enables/disables menu items.*

#### OPCODE

32 (0x20)
menu_istart()

WORD menu_istart( flag, tree, imenu, item )
WORD flag;
OBJECT *tree;
WORD imenu, item;

menu_istart() shifts a sub-menu that is attached to a menu item to align vertically with the specified object in the sub-menu.

Opcode 38 (0x26)

Availability This function is only available with AES versions 3.30 and above.

Parameters flag should be set to MIS_SETALIGN (1) to modify the alignment of a sub-menu and its parent menu item. If flag is set to MIS_GETALIGN (0), no modifications will be made, however the sub-menu item index which is currently aligned with its parent menu item is returned.

tree points to the object tree of the menu to alter. imenu specifies the object within the submenu which will be aligned with menu item item.

Binding intin[0] = flag;
intin[1] = imenu;
intin[2] = item;
addrin[0] = tree;
return crys_if(0x26);
**RETURN VALUE**

`menu_istart()` returns 0 if an error occurred or the positive object index of the sub-menu item which is currently aligned with its parent menu item.

**COMMENTS**

Generally, a sub-menu is aligned so that the currently selected sub-menu item is aligned with its parent menu.

**SEE ALSO**

`menu_attach()`

---

### `menu_popup()`

**WORD menu_popup( menu, xpos, ypos, mdata )**

**MENU *menu;**

**WORD xpos, ypos;**

**MENU *menu;**

`menu_popup()` displays a popup menu and returns the user’s selection.

**OPCODE**

36 (0x24)

**AVAILABILITY**

This function is only available with AES versions 3.30 and above.

**PARAMETERS**

`menu` points to a `MENU` structure (defined under `menu_attach()`) containing the popup menu. `xpos` and `ypos` specify the location at which the upper-left corner of the starting object will be placed.

If the function returns a value of 1, the `MENU` structure pointed to by `mdata` will be filled in with the ending state of the menu (including the object the user selected).

As of AES version 4.1, if `menu.mn_scroll` is set to `SCROLL_LISTBOX` (-1) when this function is called, a drop-down list box will be displayed instead of a popup menu.

Drop-down list boxes will only display a scroll bar if at least eight entries exist. If you want to force the scroll bar to appear, pad the object with empty `G_STRING` objects with their `DISABLED` flag set.

**BINDING**

```c
intin[0] = xpos;
intin[1] = ypos;
addrin[0] = menu;
addrin[1] = mdata;
return crys_if(0x24);
```

**RETURN VALUE**

`menu_popup()` returns 0 if an error occurred or 1 if successful.
menu_register()

WORD menu_register( ap_id, title )
WORD ap_id;
char *title;

menu_register() registers desk accessories in the ‘Desk’ menu and renames MultiTOS applications which appear there.

_OPCODE_ 35 (0x23)

_AVAILABILITY_ All AES versions.

_PARAMETERs_ ap_id specifies the application identifier of the application to register. title points to a NULL-terminated string containing the title which is to appear in the ‘Desk’ menu for the accessory or application.

If ap_id is set to REG_NEWNAME (-1) then the process name given in title will be used as the new process name. The new process name should be exactly eight characters terminated with a NULL. Pad the string with space characters if necessary.

_BINDING_ intin[0] = ap_id;
addrin[0] = title;
return crys_if(0x23);

_RETURN VALUE_ menu_register() returns a -1 if an error occurred or the menu identifier otherwise.

_VERSION NOTES_ Applications other than desk accessories should not call this function unless they are running under MultiTOS.

_COMMENTS_ Desk accessories should store the return value as this is the value that will be included with future AC_OPEN messages to identify the accessory.

Applications running under MultiTOS may use this function to provide a more functional title for the ‘Desk’ menu than the program’s filename.

Calling menu_register() with a parameter of REG_NEWNAME is used to change the internal process name of the application returned by appl_find() and appl_search(). This is useful if you know another process will attempt to find your
application as a specific process name and the user may have renamed your application filename (normally used as the process name).

---

**menu_settings()**

```c
WORD menu_settings(flag, set )
WORD flag;
MN_SET *set;
```

*menu_settings()* changes the global settings for popup and scrollable menus.

**Opcode**

39 (0x27)

**Availability**

This function is only available with AES versions 3.30 and above.

**Parameters**

If *flag* is 0, current settings are read into the *MN_SET* structure pointed to by *set*. If *flag* is 1, current settings are set from the *MN_SET* structure pointed to by *set*. *MN_SET* is defined as follows:

```c
typedef struct {
    /* Submenu-display delay in milliseconds */
    LONG display;

    /* Submenu-drag delay in milliseconds */
    LONG drag;

    /* Single-click scroll delay in milliseconds*/
    LONG delay;

    /* Continuous-scroll delay in milliseconds */
    LONG speed;

    /* Menu scroll height (in items) */
    WORD height;
} MN_SET;
```

**Binding**

```c
intin[0] = flag;
addrin[0] = set;
return crys_if(0x27);
```

**Return Value**

*menu_settings()* always returns 1.

**Comments**

The defaults set by *menu_settings()* are global and not local to an application. You should therefore limit your use of this function to system applications like CPX’s and so forth.
**menu_text()**

```c
WORD menu_text( tree, obj, text )
OBJECT *tree;
WORD obj;
char *text;
```

*menu_text()* changes the text of a menu item.

**OPCODE**
34 (0x22)

**AVAILABILITY**
All AES versions.

**PARAMETERS**
*tree* specifies the object tree of the menu bar. *obj* specifies the object index of the menu item to change. *text* points to a **NULL**-terminated character string containing the new text.

```c
intin[0] = obj;
addrin[0] = tree;
addrin[1] = text;
return crys_if(0x22);
```

**RETURN VALUE**
*menu_text()* returns a 0 if an error occurred or non-zero otherwise.

**COMMENTS**
The new menu item text must be no larger than the original menu item text.

---

**menu_tnormal()**

```c
WORD menu_tnormal( tree, obj, flag )
OBJECT *tree;
WORD obj, flag;
```

*menu_tnormal()* highlights/un-highlights a menu-title.

**OPCODE**
33 (0x21)

**AVAILABILITY**
All AES versions.

**PARAMETERS**
*tree* specifies the object tree of the menu. *obj* specifies the object index of the title to change. *flag* should be set to **HIGHLIGHT** (0) to display the title in reverse (highlighted) or **UNHIGHLIGHT** (1) to display it normally.
BINDING

\[ \begin{align*}
\text{intin}[0] &= \text{obj} \\
\text{intin}[1] &= \text{flag} \\
\text{addrin}[1] &= \text{tree} \\
\text{return} &= \text{crys_if}(0x21);
\end{align*} \]

RETURN VALUE

\text{menu_tnormal()} returns 0 if an error occurred or non-zero otherwise.

COMMENTS

This call is usually called by an application after a \text{MN_SELECTED} message is received and processed to return the menu title to normal.
The *Object Library* is responsible for the drawing and manipulation of *AES* objects such as boxes, strings, icons, etc. See earlier in this chapter for a complete discussion of *AES* objects. The *Object Library* includes the following functions:

- `objc_add()`
- `objc_change()`
- `objc_delete()`
- `objc_draw()`
- `objc_edit()`
- `objc_find()`
- `objc_offset()`
- `objc_order()`
- `objc_sysvar()`
objc_add()

WORD objc_add( tree, parent, child )
OBJECT *tree;
WORD parent, child;

objc_add() establishes a child object’s relationship to its parent.

OPCODE 40 (0x28)

AVAILABILITY All AES versions.

PARAMETERS tree specifies the object tree to modify. parent and child specify the parent and child object to update.

BINDING
intin[0] = parent;
intin[1] = child;
addrin[0] = tree;
return crys_if(0x28);

RETURN VALUE objc_add() returns a 0 if an error occurred or non-zero otherwise.

COMMENTS In order for this function to work, the object to be added must be already be a member of the OBJECT array. This function simply updates the ob_next, ob_head, and ob_tail structure members of OBJECTs in the object tree. These fields should be initialized to NIL (0) in the child to be added.

SEE ALSO objc_order(), objc_delete()

objc_change()

WORD objc_change( tree, obj, rsvd, ox, oy, ow, oh, newstate, drawflag )
OBJECT *tree;
WORD obj, rsvd, ox, oy, ow, oh, newstate, drawflag;

objc_change() changes the display state of an object.

OPCODE 47 (0x2F)

AVAILABILITY All AES versions.

PARAMETERS tree specifies the object tree of the object to modify. obj specifies the object to
modify.

\textit{rsvd} is reserved and should be 0.

\textit{ox}, \textit{oy}, \textit{ow}, and \textit{oh} specify the clipping rectangle if the object is to be redrawn.

\textit{newstate} specifies the new state of the object (same as \textit{ob_state}).

If \textit{drawflag} is \textbf{NO_DRAW} (0) the object is not redrawn whereas if \textit{drawflag} is \textbf{REDRAW} (1) the object is redrawn.

\textbf{BINDING}

\begin{verbatim}
intin[0] = obj;
intin[1] = rsvd;
intin[2] = ox;
intin[3] = oy;
intin[4] = ow;
intin[5] = oh;
intin[6] = newstate;
intin[7] = drawflag;
addrin[0] = tree;
return crys_if(0x2F);
\end{verbatim}

\textbf{RETURN VALUE}

\textbf{objc_change()} returns 0 if an error occurred and non-zero otherwise.

\textbf{COMMENTS}

In general, if not redrawing the object, it is usually quicker to manipulate the object tree directly.

\textbf{SEE ALSO}

\textbf{objc_draw()}

---

\textbf{objc_delete()}

\textbf{WORD }\textbf{objc_delete(} \textbf{tree, obj )}

\textbf{OBJECT *tree;}

\textbf{WORD }\textbf{obj;}

\textbf{objc_delete()} removes an object from an object tree.

\textbf{OPCODE}

41 (0x29)

\textbf{AVAILABILITY}

All AES versions.

\textbf{PARAMETERS}

\textit{tree} specifies the object tree of the object to delete. \textit{obj} is the object to be deleted.

\textbf{BINDING}

\begin{verbatim}
intin[0] = obj;
addrin[0] = tree;
\end{verbatim}
objc_draw() – 6.117

return crys_if(0x29);

**RETURN VALUE**
objc_delete() returns 0 if an error occurred or non-zero otherwise.

**COMMENTS**
This function does not move other objects in the tree structure, it simply unlinks the specified object from the object chain by updating the other object’s ob_next, ob_head, and ob_tail structure members.

**SEE ALSO**
objc_add()

---

objc_draw()

WORD objc_draw( tree, obj, depth, ox, oy, ow, oh )
OBJECT *tree;
WORD obj, depth, ox, oy, ow, oh;

objc_draw() renders an AES object tree on screen.

**OPCODE**
42 (0x2A)

**AVAILABILITY**
All AES versions.

**PARAMETERS**
tree specifies the object tree to draw. obj specifies the object index at which drawing is to begin.

depth specifies the maximum object depth to draw (a value of 1 searches only first generation objects, a value of 2 searches up to second generation objects, up to a maximum of 7 to search all objects).

ox, oy, ow, and oh specify an AES style rectangle which defines the clip rectangle to enforce during drawing.

**BINDING**
intin[0] = obj;
intin[1] = depth;
intin[2] = ox;
intin[3] = oy;
intin[4] = ow;
intin[5] = oh;

addrin[0] = tree;
return crys_if(0x2A);

**RETURN VALUE**
objc_draw() returns 0 if an error occurred or non-zero otherwise.
objc_edit()

WORD objc_edit( tree, obj, kc, idx, mode )
OBJECT *tree;
WORD obj, kc;
WORD *idx
WORD mode;

objc_edit() allows manual control of an editable text field.

OPCODE 46 (0x2E)

AVAILABILITY All AES versions.

PARAMETERS

tree specifies the object tree containing the editable object obj to modify. mode specifies the action of the call and the meaning of the other parameters as follows:

<table>
<thead>
<tr>
<th>mode</th>
<th>Value</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>ED_START</td>
<td>0</td>
<td>Reserved for future use. Do not call.</td>
</tr>
<tr>
<td>ED_INIT</td>
<td>1</td>
<td>Display the edit cursor in the object specified. kc is ignored. The WORD pointed to by idx is filled in with the current index of the edit cursor in the field.</td>
</tr>
<tr>
<td>ED_CHAR</td>
<td>2</td>
<td>A key has been pressed that needs special processing. kc contains the keyboard scan code in the high byte and ASCII code in the low byte. idx points to the current index of the text cursor in the field. idx will be updated as a result of this call.</td>
</tr>
<tr>
<td>ED_END</td>
<td>3</td>
<td>Turn off the text cursor.</td>
</tr>
</tbody>
</table>

BINDING

intin[0] = obj;
intin[1] = kc;
intin[2] = *idx;
intin[3] = mode;

addrin[0] = tree;
crys_if(0x2E);

*idx = intout[1];
return intout[0];

RETURN VALUE objc_edit() returns 0 if an error occurred or non-zero otherwise.

COMMENTS This function is usually used in conjunction with form_keybd() in a custom form_do() handler.
objc_find()

WORD objc_find( tree, obj, depth, ox, oy )
OBJECT *tree;
WORD obj, depth, ox, oy;

objc_find() determines which object is found at a given coordinate.

_OPCODE 43 (0x2B)

_AVAILABILITY All AES versions.

_PARAMETERS tree specifies the object tree containing the objects to search. The search starts from object index obj forward in the object tree.

depth specifies the depth in the tree to search (a value of 1 searches only first generation objects, a value of 2 searches up to second generation objects, up to a maximum of 7 to search all objects).

ox and oy specify the coordinate to search at.

_BINDING

intin[0] = obj;
intin[1] = depth;
intin[2] = ox;
intin[3] = oy;

addrin[0] = tree;
return crys_if(0x2B);

_RETURN VALUE_ objc_find() returns the object index of the object found at coordinates (ox, oy) or -1 if no object is found.

objc_offset()

WORD objc_offset( tree, obj, ox, oy )
OBJECT *tree;
WORD obj;
WORD *ox, *oy;

objc_offset() calculates the true screen coordinates of an object.

_OPCODE 44 (0x2C)

SEE ALSO form_keybd()
objc_offset()

WORD objc_offset( tree, obj )
OBJECT *tree;
WORD obj;

objc_offset() returns 0 if an error occurred or non-zero otherwise.

objc_order()

WORD objc_order( tree, obj, pos )
OBJECT *tree;
WORD obj, pos;

objc_order() changes the position of an object relative to other child objects of the same parent.

OPCODE 45 (0x2D)

AVAILABILITY All AES versions.

PARAMETERS tree specifies the object tree of object obj which is to be moved. pos specifies the new position of the object as follows:
objc_sysvar() – 6.121

<table>
<thead>
<tr>
<th>Name</th>
<th>pos</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>OO_LAST</td>
<td>-1</td>
<td>Make object the last child.</td>
</tr>
<tr>
<td>OO_FIRST</td>
<td>0</td>
<td>Make object the first child.</td>
</tr>
<tr>
<td>—</td>
<td>1</td>
<td>Make object the second child.</td>
</tr>
<tr>
<td>—</td>
<td>2</td>
<td>etc...</td>
</tr>
</tbody>
</table>

**BINDING**

```c
intin[0] = obj;
intin[1] = pos;
addrin[0] = tree;
return crys_if(0x2D);
```

**RETURN VALUE**
objc_order() returns 0 if an error occurred or non-zero otherwise.

**COMMENTS**
objc_order() does not actually move structure elements in memory. It works by updating the OBJECT tree's ob_head, ob_tail, and ob_next fields to ‘move’ the OBJECT in the tree hierarchy.

---

objc_sysvar()

WORD objc_sysvar( mode, which, in1, in2, out1, out2 )

WORD mode, which, in1, in2;
WORD *out1, *out2;

objc_sysvar() returns/modifies information about the color and placement of 3D object effects.

**OPCODE**

48 (0x30)

**AVAILABILITY**

Available as of AES version 3.40.

**PARAMETERS**

mode determines whether attributes should be read or modified. A value of SV_INQUIRE (0) will read the current values whereas a value of SV_SET (1) will modify the current values. which determines what attribute you wish to read or modify.

When reading values, in1 and in2 are unused. The two return values are placed in the WORDs pointed to by out1 and out2. When modifying values, out1 and out2 are unused. in1 and in2 specify the new values for the attribute.

The meanings of the two input/output values referred to as val1 and val2 are as follows:
<table>
<thead>
<tr>
<th>Name</th>
<th>which</th>
<th>Values</th>
</tr>
</thead>
<tbody>
<tr>
<td>LK3DIND</td>
<td>1</td>
<td>If val1 is 1, the text of indicator objects does move when selected, otherwise, if 0, it does not. If val2 is 1, the color of indicator objects does change when selected, otherwise, if 0, it does not.</td>
</tr>
<tr>
<td>LK3DACT</td>
<td>2</td>
<td>Same as LK3DIND for activator objects.</td>
</tr>
<tr>
<td>INDBUTCOL</td>
<td>3</td>
<td>val1 specifies the default color for indicator objects. val2 is unused.</td>
</tr>
<tr>
<td>ACTBUTCOL</td>
<td>4</td>
<td>val1 specifies the default color for activator objects. val2 is unused.</td>
</tr>
<tr>
<td>BACKGRCOL</td>
<td>5</td>
<td>val1 specifies the default color for background objects. val2 is unused.</td>
</tr>
<tr>
<td>AD3DVAL</td>
<td>6</td>
<td>val1 specifies the number of extra pixels on each horizontal side of an indicator or activator object needed to accommodate 3D effects. val2 specifies the number of extra pixels on each vertical side of an indicator or activator object needed to accommodate 3D effects. This setting may only be read, not modified.</td>
</tr>
</tbody>
</table>

**BINDING**

```c
intin[0] = mode;
intin[1] = which;
intin[2] = in1;
intin[3] = in2;

cryx_if(0x30);

*out1 = intout[1];
*out2 = intout[2];

return intout[0];
```

**RETURN VALUE**
objc_sysvar() returns 0 if unsuccessful or non-zero otherwise.

**COMMENTS**
Applications should not use objc_sysvar() to change these settings since all changes are global. Only CPXs or Desk Accessories designed to modify these parameters should.
The *Resource Library* is responsible for the loading/unloading of resource files and the manipulation of resource objects in memory. The members of the *Resource Library* are:

- `rsrch_free()`
- `rsrch_gaddr()`
- `rsrch_load()`
- `rsrch_obfix()`
- `rsrch_rcfix()`
- `rsrch_saddr()`
rsrcl_free()

WORD rsrc_free( VOID )

rsrcl_free() releases memory allocated by rsrc_load() for an application’s resource.

OPCODE 111 (0x6F)

AVAILABILITY All AES versions.

BINDING

return crys_if(0x6F);

RETURN VALUE rsrc_free() returns 0 if an error occurred or non-zero otherwise.

COMMENTS rsrc_free() should be called before an application which loaded a resource using rsrc_load() exits.

SEE ALSO rsrc_load()

rsrcl_gaddr()

WORD rsrc_gaddr( type, index, addr )
WORD type, index;
VOIDPP addr;

rsrcl_gaddr() returns the address of an object loaded with rsrc_load().

OPCODE 112 (0x70)

AVAILABILITY All AES versions.

PARAMETERS The pointer pointed to by addr will be filled in with the address of the index\textsuperscript{th} resource object of type type. Valid values for type are as follows:

<table>
<thead>
<tr>
<th>Name</th>
<th>type</th>
<th>Resource Object</th>
</tr>
</thead>
<tbody>
<tr>
<td>R_TREE</td>
<td>0</td>
<td>Object tree</td>
</tr>
<tr>
<td>R_OBJECT</td>
<td>1</td>
<td>Individual object</td>
</tr>
<tr>
<td>R_TEDINFO</td>
<td>2</td>
<td>TEDINFO structure</td>
</tr>
<tr>
<td>R_ICONBLK</td>
<td>3</td>
<td>ICONBLK structure</td>
</tr>
<tr>
<td>R_BITBLK</td>
<td>4</td>
<td>BITBLK structure</td>
</tr>
<tr>
<td>R_STRING</td>
<td>5</td>
<td>Free String data</td>
</tr>
</tbody>
</table>
6.126 – Resource Library - AES Function Reference

| R_IMAGEDATA | 6 | Free Image data |
| R_OBSPEC    | 7 | ob_spec field within OBJECTs |
| R_TEPTTEXT  | 8 | te_ptext within TEDINFOs |
| R_TEPTMPLT  | 9 | te_ptmplt within TEDINFOs |
| R_TEPVALID  | 10 | te_pvalid within TEDINFOs |
| R_IBPMASK   | 11 | ib_pmask within ICONBLKs |
| R_IBPDATA   | 12 | ib_pdata within ICONBLKs |
| R_IBPTEXT   | 13 | ib_ptext within ICONBLKs |
| R_BIPDATA   | 14 | bi_pdata within BITBLKs |
| R_FRSTR     | 15 | Free string |
| R_FRIMG     | 16 | Free image |

**BINDING**

```c
intin[0] = type;
intin[1] = index;
crys_if(0x70);
*addr = addrout[0];
return intout[0];
```

**RETURN VALUE**

`rsrc_gaddr()` returns a 0 if the address in `addr` is valid or non-zero if the object did not exist.

**COMMENTS**

This function is most often used to obtain the address of OBJECT trees, ‘free’ strings, and ‘free’ images after loading a resource file.

**SEE ALSO**

`rsrc_saddr()`

---

**rsrcl_load()**

`WORD rsrcl_load(fname )`

`char *fname;`

`rsrcl_load()` loads and allocates memory for the named resource file.

**OPCODE**

110 (0x6E)

**AVAILABILITY**

All AES versions.

**PARAMETERS**

`fname` is a character pointer to a NULL-terminated GEMDOS file specification of the resource to load.

**BINDING**

```c
addrin[0] = fname;
```
rsrc_obfix() – 6.127

return crys_if(0x6E);

RETURN VALUE
rsrcreturns 0 if successful or non-zero if an error occurred.

COMMENTS
In addition to loading the resource, all OBJECT coordinates are converted from character based coordinates to screen coordinates.

SEE ALSO
rsrcrefree()
rsrc_rcfix()

WORD rsrc_rcfix( rc_header )
VOID *rc_header;

rsrc_rcfix() fixes up coordinates and memory pointers of raw resource data in memory.

OPCODE 115 (0x73)

AVAILABILITY Available only in AES versions 4.0 and greater. The presence of this call should also be checked for using appl_getinfo().

PARAMETERS rc_header is a pointer to an Atari Resource Construction Set (or compatible) resource file header in memory.

BINDING addrin[0] = rc_header;
return crys_if(0x73);

RETURN VALUE rsrc_rcfix() returns a 0 if successful or non-zero otherwise.

COMMENTS If a resource has already been loaded with rsrc_load() it must be freed by rsrc_free() prior to this call. In addition, resources identified with this call must likewise be freed before program termination or another resource file is needed.

SEE ALSO rsrc_obfix()

rsrc_saddr()

WORD rsrc_saddr( type, index, addr )
WORD type, index;
VOID *addr;

rsrc_saddr() sets the address of a resource element.

OPCODE 113 (0x71)

AVAILABILITY All AES versions.

PARAMETERS type specifies the type of resource element to set as defined under rsrc_gaddr(). index specifies the index of the element to modify (0 based). addr specifies the actual address that will be placed in the appropriate data structure.
**BINDING**

```c
intin[0] = type;
intin[1] = index;

addrin[0] = addr;

return crys_if(0x71);
```

**RETURN VALUE**  
rsr_saddr() returns 0 if an error occurred or non-zero otherwise.

**COMMENTS**  
In most cases, direct manipulation of the structures involved is quicker and easier than using this call.

**SEE ALSO**  
rsr_gaddr(), rsr_load()
The *Scrap Library* is used to maintain the location of the clipboard directory used for interprocess data exchange. The members of the *Scrap Library* are:

- `scrp_read()`
- `scrp_write()`
scrp_read()

WORD scrp_read( cpath )
char *cpath;

scrp_read() returns the location of the current clipboard directory.

**Opcode**

80 (0x50)

**Availability**

All AES versions.

**Parameters**

cpath is a pointer to a character buffer of at least 128 bytes into which the clipboard path will be placed.

**Binding**

addrin[0] = cpath;
return crys_if(0x50);

**Return Value**

scrp_read() returns 0 if the clipboard path had not been set or non-zero if cpath was properly updated.

**Caveats**

The system scrap directory is a global resource. Some programs incorrectly call scrp_write() with a path and filename when only a pathname should be used. The following is an example of a correctly formatted cpath argument:

```
C:\CLIPBRD
```

Unfortunately, not all programs adhere exactly to this standard. For this reason, programs reading this information from scrp_read() should be especially careful that the information returned is parsed correctly. In addition, don’t count on a trailing backslash or the existence of a drive specification.

**Comments**

If a value of 0 is returned and the application wishes to write a scrap to the clipboard you should follow these steps:

- Create a folder ‘\CLIPBRD\’ on the root directory of the user’s boot drive (‘C:’ or ‘A:’).
- Write your scrap to the directory as ‘SCRAP.???’ where ‘???’ signifies the type of information contained in the file.
- Allow other applications to access this information by calling scrp_write() with the new clipboard path. For example “C:\CLIPBRD\”.

A detailed discussion of the proper clipboard data exchange protocol, including information about a scrap directory semaphore useful with MultiTOS, is given earlier in this chapter.
scrp_write(

WORD scrp_write( cpath )
char *cpath;

scrp_write() sets the location of the clipboard directory.

OPCODE 81 (0x51)

AVAILABILITY All AES versions.

PARAMETERS cpath points to a NULL-terminated path string containing a valid drive and path specification with a closing backslash. The following is an example of a correctly formatted cpath argument:

C:\CLIPBRD\n
BINDING

addrin[0] = cpath;
return crys_if(0x51);

RETURN VALUE scrp_write() returns 0 if an error occurred or non-zero otherwise.

COMMENTS The scrap directory is a global resource. This call should only be used in two circumstances as follows:

- when used to set the default location of the scrap directory using a CPX or accessory at bootup or by the user’s request.
- when scrp_read() returns an error value and you need to create the clipboard to write information to it.

The clipboard data exchange protocol is discussed in greater detail earlier in this chapter.

SEE ALSO scrp_read()
The Shell Library contains several miscellaneous functions most often used by the GEM Desktop and other ‘Desktop-like’ applications. Other applications may, however, need specific functions of the Shell Library for various tasks. The members of the Shell Library are:

- shel_envrn()
- shel_find()
- shel_get()
- shel_put()
- shel_read()
- shel_write()
shel_envrn()

WORD shel_envrn( value, name )
char **value;
char *name;

shel_envrn() searches the current environment string for a specific variable.

OPCODE 125 (0x7D)

AVAILABILITY All AES versions.

PARAMETERS value points to a character pointer which will be filled in with the address of the first character in the environment string following the string given by name. If the string given by name is not found, value will be filled in with NULL. For instance, suppose the current environment looked like this:

PATH=C:\;D:\;E:\

A call made to shel_envrn() with name pointing to the string ‘PATH=’ would set the pointer pointed to by value to the string ‘C:\;D:\;E:\’ above.

BINDING addrin[0] = value;
addrin[1] = name;

return crys_if(0x7D);

RETURN VALUE shel_envrn() currently always returns 1.

VERSION NOTES AES versions prior to 1.4 only accepted semi-colons as separators between multiple ‘PATH=’arguments. Newer versions accept commas as well.

COMMENTS The character string pointed to by name should include the name of the variable and the equals sign.

shel_find()

WORD shel_find( buf )
char *buf;

shel_find() searches for a file along the AES’s current path, any paths specified by the ‘PATH’ environmental variable, and the calling application’s path.

OPCODE 124 (0x7C)
**shel_find()**

WORD shel_find( buf )
char *buf;

shel_find() returns 0 if the file was not found or non-zero otherwise.

**shel_get()**

WORD shel_get( buf, length )
char *buf;
WORD length;

shel_get() copies the contents of the AES’s shell buffer (normally the ‘DESKTOP.INF’ or ‘NEWDESK.INF’ file) into the specified buffer.

**Availability**

All AES versions.

**Parameters**

buf should point to a character buffer of at least 128 characters and contain the filename of the file to search for on entry. If the function was able to find the file, the buffer pointed to by buf will be filled in with the full pathname of the file upon return.

**Binding**

addrin[0] = buf;

return crys_if(0x7C);

**Return Value**

shel_get() returns 0 if an error occurred or non-zero otherwise.

**Version Notes**

AES versions prior to version 1.4 had a shell buffer size of 1024 bytes. Versions 1.4 to 3.0 had a shell buffer size of 4192 bytes.

In AES versions 4.0 or greater the shell buffer is no longer of a fixed size. When appl_getinfo() indicates that this feature is supported, length can be specified as SHEL_BUFSIZE (-1) to return the size of the current shell buffer.

**See Also**

shel_write()
shel_put()

WORD shel_put( buf, length )
char *buf;
WORD length;

shel_put() copies information into the AES’s shell buffer.

**OPCODE**
123 (0x7B)

**AVAILABILITY**
All AES versions.

**PARAMETERS**
*buf* points to a user memory buffer from which *length* bytes are to be copied into the shell buffer.

**BINDING**

```c
intin[0] = length;
addrin[0] = buf;
return crys_if(0x7B);
```

**RETURN VALUE**
shel_put() returns 0 if an error occurred or non-zero otherwise.

**VERSION NOTES**
Prior to AES version 4.0 this function would only copy as many bytes as would fit into the current buffer. As of version 4.0, the AES will dynamically allocate more memory as needed (up to 32767 bytes) for the shell buffer.

**COMMENTS**
The Desktop uses the information in the shell buffer for several purposes. Applications should not use the shell buffer for their own purposes.

**SEE ALSO**
shel_get()

shel_read()

WORD shel_read( name, tail )
char *name, *tail;

shel_read() is used to determine the current application’s parent and the command tail used to call it.

**OPCODE**
120 (0x78)
AVAILABILITY  All AES versions.

PARAMETERS  

(name) points to a buffer which upon exit will be filled in with the complete file specification of the application which launched the current process.

(tail) will likewise be filled in with the initial command line. The first BYTE of the command line indicates the length of the string which actually begins at &tail[1].

BINDING  

addrin[0] = name;
addrin[1] = tail;

return crys_if(0x78);

RETURN VALUE  

shel_read() returns 0 if an error occurred or non-zero otherwise.

CAVEATS  

shel_read() actually returns the arguments to the last shel_write() so if a process was Pexec()’ed, the information returned will be incorrect.

shel_write()

WORD shel_write( mode, wisgr, wiscr, cmd, tail )
WORD mode, wisgr, wiscr;
char *cmd, *tail;

shel_write() is a multi-purpose function which handles the manipulation and launching of processes.

OPCODE  

121 (0x79)

AVAILABILITY  

All AES versions. In AES versions 4.0 and above, appl_getinfo() can be used to determine the highest legal value for mode as well as the functionality of extended mode bits.

PARAMETERS  

(mode) specifies the meaning of the rest of the parameters as follows:

<table>
<thead>
<tr>
<th>Name</th>
<th>mode</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>SWM_LAUNCH</td>
<td>0</td>
<td>Launch a GEM or TOS application or GEM desk accessory depending on the extension of the file. This mode is only available as of AES version 4.0. wisgr is not used in mode SWM_LAUNCH (0). When the lower eight bits of mode are SWM_LAUNCH (0), SWM_LAUNCHNOW (1), or SWM_LAUNCHACC (3), appropriate bits in the upper byte may be set to enter ‘extended’ mode. The bits in the upper byte are assigned as follows:</td>
</tr>
</tbody>
</table>
If the upper byte is empty, extended mode is not entered and *cmd* specifies the filename (to search for the file with *shel_find()*) or the complete file specification. Otherwise, if any extended bits are set, *cmd* points to a structure as shown below.

```c
typedef struct _shelw {
    char *newcmd;
    LONG psetlimit;
    LONG prenice;
    char *defdir;
    char *env;
} SHELW;
```

_*shelw.newcmd* points to the filename formatted in the manner indicated above.

If bit 8 (*SW_PSETLIMIT*) of *mode* is set, *shelw.psetlimit* contains the maximum memory size available to the process.

If bit 9 of *mode* is (*SW_PRENICE*) set, *shelw.prenice* contains the process priority of the process to launch.

If bit 10 of *mode* (*SW_DEFDIR*) is set, *shelw.defdir* points to a character string containing the default directory for the application begin launched.

If bit 11 of *mode* (*SW_ENVIRON*) is set, *shelw.env* points to a valid environment string for the process.

tail points to a buffer containing the command tail to pass to the process. If *wiscr* is set to CL_NORMAL (0), tail is passed normally, otherwise, if *wiscr* is set to CL_PARSE (1), the AES will parse tail and set up an ARGV environment string.

*modes* SWM_LAUNCH (0), SWM_LAUNCHNOW (1), and SWM_LAUNCHACC (3) return the AES id of the started process. If a 0 is returned, then the process was not launched.

Under MultiTOS, processes are launched concurrently with their parent. An exit code is returned in a CH_EXIT message when the child terminates. See evnt_mesag().

In AES versions 4.0 and above, appl_getinfo() should be used to determine the exact result of this call.
<table>
<thead>
<tr>
<th>Function</th>
<th>Value</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>SWM_LAUNCHNOW</td>
<td>1</td>
<td>Launch a GEM or TOS application based on the value of wisgr. If wisgr is TOSAPP (0), the application will be launched as a TOS application, otherwise if wisgr is GEMAPP (1), the application will be launched as a GEM application. For the meaning of other parameters, see mode SWM_LAUNCH (0). The extended bits in mode are only supported by AES versions of at least 4.0. Parent applications which launch children using this mode are suspended under MultiTOS. In AES versions 4.0 and above, appl_getinfo() should be used to determine the exact result of this call.</td>
</tr>
<tr>
<td>SWM_LAUNCHACC</td>
<td>3</td>
<td>Launch a GEM desk accessory. For the meaning of other parameters, see mode SWM_LAUNCH (0). This mode is only supported by AES versions of at least 4.0.</td>
</tr>
<tr>
<td>SWM_SHUTDOWN</td>
<td>4</td>
<td>Manipulate 'Shutdown' mode. Shutdown mode is usually used prior to a resolution change to cause system processes to terminate. wisgr, cmd, and tail are ignored by this call. The value of wiscr determines the action this call takes as follows:</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Name</td>
</tr>
<tr>
<td></td>
<td></td>
<td>SD_ABORT</td>
</tr>
<tr>
<td></td>
<td></td>
<td>SD_PARTIAL</td>
</tr>
<tr>
<td></td>
<td></td>
<td>SD_COMPLETE</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>SWM_REZCHANGE</td>
<td>5</td>
<td>Change screen resolution. wisgr is the work station ID (same as in AES global[13]) of the new resolution. No other parameters are utilized. This mode is only recognized as of AES version 4.0.</td>
</tr>
<tr>
<td>SWM_BROADCAST</td>
<td>7</td>
<td>Broadcast an AES message to all processes. cmd should point to an 8 WORD message buffer containing the message to send. All other parameters are ignored. This mode is only recognized as of AES version 4.0.</td>
</tr>
</tbody>
</table>
| Binding          | intin[0] = mode;  
|                 | intin[1] = wisgr; 
|                 | intin[2] = wiscr; 
|                 | addrin[0] = cmd;  
|                 | addrin[1] = tail; 
|                 | return crys_if(0x79); |
| Return Value    | The value `shel_write()` differs depending on the mode which was invoked. See above for details. |
| Version Notes   | Many new features were added as of AES version 4.0. For details of each, see above. |

**SWM_ENVIRON**  8  Manipulate the AES environment. If `wisgr` is `ENVIRON_SIZE` (0), the current size of the environment string is returned.

If `wisgr` is `ENVIRON_CHANGE` (1), `cmd` should point to a environment variable to modify. If `cmd` points to “TOSEXT=TOS,TTP”, that string will be added. Likewise, “TOSEXT=” will remove that environment variable.

If `wisgr` is `ENVIRON_COPY` (2), the AES will copy as many as `wiscr` bytes of the current environment string into a buffer pointer to by `cmd`. The function will return the number of bytes not copied.

This mode is only recognized as of AES version 4.0.

**SWM_NEWMSG**  9  Inform the AES of a new message the current application understands. `wisgr` is a bit mask which specifies which new messages the application understands. Currently only bit 0 (`B_UNTIPPABLE`) has a meaning. Setting this bit when calling this function will inform the AES that the application understands `AP_TERM` messages. No other parameters are used.

This mode is only recognized as of AES version 4.0.

**SWM_AESMSG**  10  Send a message to the AES. `cmd` points to an 8 WORD message buffer containing the message to send. No other parameters are needed.

This mode is only recognized as of AES version 4.0.
Window Library

The Window Library is responsible for the displaying and maintenance of AES windows. The members of the Window Library are:

- wind_calc()
- wind_close()
- wind_create()
- wind_delete()
- wind_find()
- wind_get()
- wind_new()
- wind_open()
- wind_set()
- wind_update()
wind_calc()

WORD wind_calc( request, kind, x1, y1, w1, h1, x2, y2, w2, h2 )
WORD request, kind, x1, y1, w1, h1;
WORD *x2, *y2, *w2, *h2;

wind_calc() returns size information for a specific window.

OPCODE 108 (0x6C)

AVAILABILITY All AES versions.

PARAMETERS request specifies the mode of this call.

If request is WC_BORDER (0), x1, y1, w1, and h1 specify the work area of a window of type kind. The call then fills in the WORDs pointed to by x2, y2, w2, and h2 with the full extent of the window.

If request is WC_WORK (1), x1, y1, w1, and h1 specify the full extent of a window of type kind. The call fills in the WORDs pointed to by x2, y2, w2, and h2 with the work area of the window.

kind is a bit mask of window ‘widgets’ present with the window. For a detailed listing of these elements see wind_create().

BINDING

intin[0] = request;
intin[1] = kind;
intin[2] = x1;
intin[3] = y1;
intin[4] = w1;
intin[5] = h1;
crysi_if(0x6C);

*x2 = intout[1];
*y2 = intout[2];
*w2 = intout[3];
*h2 = intout[4];

return intout[0];

RETURN VALUE wind_calc() returns 0 if an error occurred or non-zero otherwise.

COMMENTS wind_calc() is unable to calculate correct values when a toolbar is attached to a window. This can be corrected, though, by adjusting the values output by this function with the height of the toolbar.

SEE ALSO wind_create()
wind_close()

WORD wind_close( handle )
WORD handle;

wind_close() removes a window from the display screen.

OPCODE 102 (0x66)

AVAILABILITY All AES versions.

PARAMETERS handle specifies the window handle of the window to close.

BINDING

intin[0] = handle;
return crys_if(0x66);

RETURN VALUE wind_close() returns 0 if an error occurred or non-zero otherwise.

COMMENTS

Upon calling wind_close() a redraw message for the portion of the screen changed will be sent to all applications.

Calling wind_close() does not release the memory allocated to the window structure. wind_delete() must be called to permanently destroy the window and free any memory allocated by the AES for the window. Until wind_delete() is called, the window may be re-opened at any time with wind_open().

SEE ALSO wind_create(), wind_open(), wind_delete()

wind_create()

WORD wind_create( kind, x, y, w, h )
WORD kind, x, y, w, h;

wind_create() initializes a new window structure and allocates any necessary memory.

OPCODE 100 (0x64)

AVAILABILITY All AES versions.

PARAMETERS kind is a bit array whose elements determine the presence of any ‘widgets’ on the
window as follows:

<table>
<thead>
<tr>
<th>Name</th>
<th>Mask</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>NAME</td>
<td>0x01</td>
<td>Window has a title bar.</td>
</tr>
<tr>
<td>CLOSER</td>
<td>0x02</td>
<td>Window has a close box.</td>
</tr>
<tr>
<td>FULLER</td>
<td>0x04</td>
<td>Window has a fuller box.</td>
</tr>
<tr>
<td>MOVER</td>
<td>0x08</td>
<td>Window may be moved by the user.</td>
</tr>
<tr>
<td>INFO</td>
<td>0x10</td>
<td>Window has an information line.</td>
</tr>
<tr>
<td>SIZER</td>
<td>0x20</td>
<td>Window has a sizer box.</td>
</tr>
<tr>
<td>UPARROW</td>
<td>0x40</td>
<td>Window has an up arrow.</td>
</tr>
<tr>
<td>DNARROW</td>
<td>0x80</td>
<td>Window has a down arrow.</td>
</tr>
<tr>
<td>VSLIDE</td>
<td>0x100</td>
<td>Window has a vertical slider.</td>
</tr>
<tr>
<td>LFARROW</td>
<td>0x200</td>
<td>Window has a left arrow.</td>
</tr>
<tr>
<td>RTARROW</td>
<td>0x400</td>
<td>Window has a right arrow.</td>
</tr>
<tr>
<td>HSLIDE</td>
<td>0x800</td>
<td>Window has a horizontal slider.</td>
</tr>
<tr>
<td>SMALLER</td>
<td>0x4000</td>
<td>Window has an iconifier.</td>
</tr>
</tbody>
</table>

The parameter kind is created by OR’ing together any desired elements.

$x$, $y$, $w$, and $h$, specify the maximum extents of the window. Normally this is the entire screen area minus the menu bar (to find this area use wind_get() with a parameter of WF_WORKXYWH). The area may be smaller to bound the window to a particular size and location.

```
BINDING
int in[0] = kind;
int in[1] = x;
int in[2] = y;
int in[3] = w;
int in[4] = h;
return crys_if(0x64);
```

RETURN VALUE

wind_create() returns a window handle if successful or a negative number if it was unable to create the window.

VERSION NOTES

The SMALLER gadget is only available as of AES version 4.1.

COMMENTS

A window is not actually displayed on screen with this call, you need to call wind_open() to do that.

TOS version 1.00 and 1.02 limited applications to four windows. In TOS version 1.04 that limit was raised to seven. As of MultiTOS the number of open windows is limited only by memory and the capabilities of an application.

You should ensure that your application calls a wind_delete() for each wind_create(), otherwise memory may not be deallocated when your application
wind_delete()

WORD wind_delete(handle)
WORD handle;

wind_delete() destroys the specified window and releases any memory allocated for it.

OPCODE 103 (0x67)

AVAILABILITY All AES versions.

PARAMETERS
handle specifies the window handle of the window to destroy.

BINDING
intin[0] = handle;
return crys_if(0x67);

RETURN VALUE wind_delete() returns 0 if an error occurred or non-zero otherwise.

COMMENTS A window should by closed with wind_close() before deleting it.

SEE ALSO wind_create(), wind_open(), wind_close(), wind_new()

wind_find()

WORD wind_find(x,y)
WORD x,y;

wind_find() returns the handle of the window found at the given coordinates.

OPCODE 106 (0x6A)

AVAILABILITY All AES versions.

PARAMETERS x and y specify the coordinates to search for a window at.

BINDING
intin[0] = x;
intin[1] = y;
wind_get() – 6.151

return crys_if(0x6A);

**RETURN VALUE**
wind_find() returns the handle of the uppermost window found at location \( x, y \). If no window is found, the function returns 0 meaning the **Desktop** window.

**COMMENTS**
This function is useful for tracking the mouse pointer and changing its shape depending upon what window it falls over.

**wind_get()**

WORD wind_get( handle, mode, parm1, parm2, parm3, parm4 )

WORD handle, mode;

WORD *parm1, *parm2, *parm3, *parm4;

**OPCODE**
104 (0x68)

**AVAILABILITY**
All AES versions.

**PARAMETERS**
*handle* specifies the handle of the window to return information about (0 is the desktop window). *mode* specifies the information to return and the values placed into the **WORDs** pointed to by *parm1, parm2, parm3, and parm4* as follows:

<table>
<thead>
<tr>
<th>Name</th>
<th>mode</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>WF_WORKXYWH</td>
<td>4</td>
<td>*parm1, parm2, parm3, and parm4 are filled in with the x, y, w, and h of the current coordinates of the window's work area.</td>
</tr>
<tr>
<td>WF_CURRXYWH</td>
<td>5</td>
<td>*parm1, parm2, parm3, and parm4 are filled in with the x, y, w, and h of the current coordinates of the full extent of the window.</td>
</tr>
<tr>
<td>WF_PREVXYWH</td>
<td>6</td>
<td>*parm1, parm2, parm3, and parm4 are filled in with the x, y, w, and h of the previous coordinates of the full extent of the window prior to the last wind_set() call.</td>
</tr>
<tr>
<td>WF_FULLXYWH</td>
<td>7</td>
<td>*parm1, parm2, parm3, and parm4 are filled in with the x, y, w, and h values specified in the wind_create() call.</td>
</tr>
<tr>
<td>WF_HSLIDE</td>
<td>8</td>
<td>*parm1 is filled in with the current position of the horizontal slider between 1 and 1000. A value of one indicates that the slider is in its leftmost position.</td>
</tr>
<tr>
<td>WF_VSLIDE</td>
<td>9</td>
<td>*parm1 is filled in with the current position of the vertical slider between 1 and 1000. A value of one indicates that the slider is in its uppermost position.</td>
</tr>
<tr>
<td>WF_TOP</td>
<td>10</td>
<td>*parm1 is filled in with the window handle of the window currently on top. As of AES version 4.0 (and when appl_getinfo() indicates), parm2 is filled in with the owners AES id, and parm3 is filled in with the handle of the window directly below it.</td>
</tr>
<tr>
<td>Function</td>
<td>Description</td>
<td></td>
</tr>
<tr>
<td>-------------------</td>
<td>-----------------------------------------------------------------------------</td>
<td></td>
</tr>
<tr>
<td>WF_FIRSTXYWH</td>
<td><code>parm1, parm2, parm3, and parm4</code> are filled in with the x, y, w, and h of the first AES rectangle in the window's rectangle list. If <code>parm3</code> and <code>parm4</code> are both 0, the window is completely covered.</td>
<td></td>
</tr>
<tr>
<td>WF_NEXTXYWH</td>
<td><code>parm1, parm2, parm3, and parm4</code> are filled in with subsequent AES rectangles for each time this function is called until <code>parm3</code> and <code>parm4</code> are 0 to signify the end of the list.</td>
<td></td>
</tr>
<tr>
<td>WF_NEWDESK</td>
<td>As of AES versions 4.0 (and when <code>appl_getinfo()</code> indicates), this mode returns a pointer to the current desktop background OBJECT tree. <code>parm1</code> contains the high WORD of the address and <code>parm2</code> contains the low WORD.</td>
<td></td>
</tr>
<tr>
<td>WF_HSLSIZE</td>
<td><code>parm1</code> contains the size of the current slider relative to the size of the scroll bar as a value from 1 to 1000. A value of 1000 indicates that the slider is at its maximum size.</td>
<td></td>
</tr>
<tr>
<td>WF_VSLSIZE</td>
<td><code>parm1</code> contains the size of the current slider relative to the size of the scroll bar as a value from 1 to 1000. A value of 1000 indicates that the slider is at its maximum size.</td>
<td></td>
</tr>
<tr>
<td>WF_SCREEN</td>
<td>This mode returns a pointer to the current AES menu/alert buffer and its size. The pointer's high WORD is returned in <code>parm1</code> and the pointer's low WORD is returned in <code>parm2</code>. The length of the buffer is returned as a LONG with the upper WORD being in <code>parm3</code> and the lower WORD being in <code>parm4</code>. Note that TOS 1.02 returns 0 in <code>w</code> and <code>h</code> by mistake. The menu/alert buffer is used by the AES to save the screen area hidden by menus and alert boxes. It is not recommended that applications use this area as its usage is not guaranteed in future versions of the OS.</td>
<td></td>
</tr>
</tbody>
</table>
### $\text{wind\_get()}$ – 6.153

<table>
<thead>
<tr>
<th>Mode</th>
<th>Value</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>WF_COLOR</strong></td>
<td>18</td>
<td>This <em>mode</em> gets the current color of the window widget specified on entry to the function in the <strong>WORD</strong> pointed to by <em>parm1</em>. Valid window widget indexes are as follows (<strong>W_SMALLER</strong> is only valid as of AES 4.1):</td>
</tr>
<tr>
<td><strong>parm1</strong></td>
<td></td>
<td><strong>ob_type</strong></td>
</tr>
<tr>
<td>W_BOX</td>
<td>0</td>
<td>IBOX</td>
</tr>
<tr>
<td>W_TITLE</td>
<td>1</td>
<td>BOX</td>
</tr>
<tr>
<td>W_CLOSER</td>
<td>2</td>
<td>BOXCHAR</td>
</tr>
<tr>
<td>W_NAME</td>
<td>3</td>
<td>BOXTEXT</td>
</tr>
<tr>
<td>W_FULLER</td>
<td>4</td>
<td>BOXCHAR</td>
</tr>
<tr>
<td>W_INFO</td>
<td>5</td>
<td>BOXTEXT</td>
</tr>
<tr>
<td>W_DATA</td>
<td>6</td>
<td>IBOX</td>
</tr>
<tr>
<td>W_WORK</td>
<td>7</td>
<td>IBOX</td>
</tr>
<tr>
<td>W_SIZER</td>
<td>8</td>
<td>BOXCHAR</td>
</tr>
<tr>
<td>W_VBAR</td>
<td>9</td>
<td>BOX</td>
</tr>
<tr>
<td>W_UPARROW</td>
<td>10</td>
<td>BOXCHAR</td>
</tr>
<tr>
<td>W_DNARROW</td>
<td>11</td>
<td>BOXCHAR</td>
</tr>
<tr>
<td>W_VSLIDE</td>
<td>12</td>
<td>BOX</td>
</tr>
<tr>
<td>W_VELEV</td>
<td>13</td>
<td>BOX</td>
</tr>
<tr>
<td>W_HBAR</td>
<td>14</td>
<td>BOX</td>
</tr>
<tr>
<td>W_LFARROW</td>
<td>15</td>
<td>BOXCHAR</td>
</tr>
<tr>
<td>W_RTARROW</td>
<td>16</td>
<td>BOXCHAR</td>
</tr>
<tr>
<td>W_HSLIDE</td>
<td>17</td>
<td>BOX</td>
</tr>
<tr>
<td>W_HELEV</td>
<td>18</td>
<td>BOX</td>
</tr>
<tr>
<td>W_SMALLER</td>
<td>19</td>
<td>BOXCHAR</td>
</tr>
</tbody>
</table>

The *ob\_spec* field (containing the color information) used for the object when not selected is returned in the **WORD** pointed to by *parm2*. The *ob\_spec* field used for the object when selected is returned in *parm3*.

This *mode* under *wind\_get()* is only valid as of AES version 3.30. From AES versions 4.0 and above, *appl\_getinfo()* should be used to determine if this mode is supported.

| **WF\_DCOLOR** | 19    | This *mode* gets the default color of newly created windows as with **WF\_COLOR** above. As above, this mode under *wind\_get()* only works as of AES version 3.30. As of AES version 4.1, **WF\_DCOLOR** changes the color of open windows unless they have had their colors explicitly set with **WF\_COLOR**. |

<p>| <strong>WF_OWNER</strong> | 20    | <em>parm1</em> is filled in with the <strong>AES</strong> id of the owner of the specified window. <em>parm2</em> is filled in with its open status (0 = closed, 1 = open). <em>parm3</em> is filled in with the handle of the window directly above it (in the window order list) and <em>parm4</em> is filled in with the handle of the window below it (likewise, in the window order list). This mode is only available as of AES version 4.0 (and when indicated by <em>appl_getinfo()</em>). |</p>
<table>
<thead>
<tr>
<th>Window Function</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>WF_BEVENT</strong></td>
<td>24</td>
</tr>
</tbody>
</table>
| *parm1, parm2, parm3, parm4* are each interpreted as bit arrays whose bits indicate supported window features. Currently only one bit is supported. If bit 0 of the value returned in *parm1* is 1, that window has been set to be ‘untappable’ and it will never receive **WM_TOPPED** messages, only button clicks.  
This mode is only available as of **AES** version 4.0 (and when indicated by **appl_getinfo()**). |
| **WF_BOTTOM**   | 25          |
| *parm1* will be filled in with the handle of the window currently on the bottom of the window list (it may actually be on top if there is only one window). Note also that this does not include the desktop window.  
This *mode* is only available as of **AES** version 4.0 (and when indicated by **appl_getinfo()**). |
| **WF_ICONIFY**  | 26          |
| *parm1* will be filled in with 0 if the window is not iconified or non-zero if it is. *parm2* and *parm3* contain the width and height of the icon. *parm4* is unused.  
This *mode* is only available as of **AES** version 4.1 (and when indicated by **appl_getinfo()**). |
| **WF_UNICONIFY**| 27          |
| *parm1, parm2, parm3, and parm4*, are filled in with the x, y, w, and h of the original coordinates of the iconified window.  
This *mode* is only available as of **AES** version 4.1 (and when indicated by **appl_getinfo()**). |
| **WF_TOOLBAR**  | 30          |
| *parm1 and parm2* contain the high and low **WORD** respectively of the pointer to the current toolbar object tree (or **NULL** if none).  
This *mode* is only available as of **AES** version 4.1. |
| **WF_FTOOLBAR** | 31          |
| *parm1, parm2, parm3, and parm4*, are filled in with the x, y, w, and h, respectively of the first uncovered rectangle of the toolbar region of the window. If *parm3* and *parm4* are 0, the toolbar is completely covered.  
This *mode* is only available as of **AES** version 4.1. |
| **WF_NTOOLBAR** | 32          |
| *parm1, parm2, parm3, and parm4*, are filled in with the x, y, w, and h, respectively of subsequent uncovered rectangles of the toolbar region. This mode should be repeated to reveal subsequent rectangles until *parm3* and *parm4* are found to be 0.  
This *mode* is only available as of **AES** version 4.1. |

**Binding**

```c
/* This binding must be different to */
/* accomodate reading WF_COLOR and */
/* WF_DCOLOR */

contrl[0] = 0x68;
contrl[1] = 2;
contrl[2] = 1;
contrl[3] = 0;
contrl[4] = 0;
```


```c
intin[0] = handle;
intin[1] = mode;

if(mode == WF_DCOLOR || mode == WF_COLOR)
{
    intin[2] = *x;
    contrl[1] = 3;
}

aes();

*x = intout[1];
*y = intout[2];
*w = intout[3];
*h = intout[4];

return intout[0];
```

**RETURN VALUE**

`wind_get()` returns a 0 if an error occurred or non-zero otherwise.

**See Also**

`wind_set()`

---

**wind_new()**

WORD wind_new( VOID )

`wind_new()` closes and deletes all of the application’s windows. In addition, the state of `wind_update()`, and the mouse pointer hide count is reset.

**Opcode**

109 (0x6D)

**Availability**

Available as of AES version 0x0140.

**Binding**

`return crys_if(0x6D);`

**Return Value**

The return value is reserved and currently unused.

**Comments**

This function should not be relied upon to clean up after an application. It was designed for parent processes that wish to ensure that a poorly written child process has properly cleaned up after itself.

**See Also**

`wind_delete()`, `graf_mouse()`, `wind_update()`
wind_open()

WORD wind_open( handle, x, y, w, h )
WORD handle;
WORD x, y, w, h;

wind_open() opens the window specified.

O P C O D E 101 (0x65)
A V A I L A B I L I T Y All AES versions.
P A R A M E T E R S handle specifies the handle of the window to open as returned by wind_create(). x, y, w, and h specify the rectangle into which the rectangle should be displayed.
B I N D I N G intin[0] = handle;
return crys_if(0x65);
R E T U R N V A L U E wind_open() returns a 0 if an error occurred or non-zero otherwise.
C O M M E N T S This call will also trigger a WM_REDRAW message which encompasses the work area of the window so applications should not initially render the work area, rather, wait for the message.
S E E A L S O wind_close(), wind_create(), wind_delete()

wind_set()

WORD wind_set( handle, mode, parm1, parm2, parm3, parm4 )
WORD handle, mode, parm1, parm2, parm3, parm4;

wind_set() sets various window attributes.

O P C O D E 105 (0x69)
A V A I L A B I L I T Y All AES versions.
P A R A M E T E R S handle specifies the window handle of the window to modify. mode specifies the attribute to change and the meanings of parm1, parm2, parm3, and parm4 as follows:
<table>
<thead>
<tr>
<th>Name</th>
<th>mode</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>WF_NAME</td>
<td>2</td>
<td>This <em>mode</em> passes a pointer to a character string containing the new title of the window. <em>parm1</em> contains the high <em>WORD</em> of the pointer and <em>parm2</em> contains the low <em>WORD</em>.</td>
</tr>
<tr>
<td>WF_INFO</td>
<td>3</td>
<td>This <em>mode</em> passes a pointer to a character string containing the new information line of the window. <em>parm1</em> contains the high <em>WORD</em> of the pointer, <em>parm2</em> contains the low <em>WORD</em>.</td>
</tr>
<tr>
<td>WF_CURRXYWH</td>
<td>5</td>
<td><em>parm1</em>, <em>parm2</em>, <em>parm3</em>, and <em>parm4</em> specify the x, y, w, and h of the new coordinates of the full extent of the window.</td>
</tr>
<tr>
<td>WF_HSLIDE</td>
<td>8</td>
<td><em>parm1</em> specifies the new position of the horizontal slider between 1 and 1000. A value of 1 indicates that the slider is in its leftmost position.</td>
</tr>
<tr>
<td>WF_VSLIDE</td>
<td>9</td>
<td><em>parm1</em> specifies the new position of the vertical slider between 1 and 1000. A value of 1 indicates that the slider is in its uppermost position.</td>
</tr>
<tr>
<td>WF_TOP</td>
<td>10</td>
<td><em>parm1</em> specifies the window handle of the window to top. Note that if multiple calls of <em>wind_set</em>(<em>WF_TOP</em>, ...) are made without releasing control to the AES (which allows the window to actually be topped), only the most recent window specified will actually change position.</td>
</tr>
<tr>
<td>WF_NEWDESK</td>
<td>14</td>
<td>This <em>mode</em> specifies a pointer to an <em>OBJECT</em> tree which is redrawn automatically by the desktop as the background. <em>parm1</em> contains the high <em>WORD</em> of the pointer and <em>parm2</em> contains the low <em>WORD</em>. To reset the desktop background to the default, specify <em>parm1</em> and <em>parm2</em> as 0.</td>
</tr>
<tr>
<td>WF_HSLSIZE</td>
<td>15</td>
<td><em>parm1</em> defines the size of the current slider relative to the size of the scroll bar as a value from 1 to 1000. A value of 1000 indicates that the slider is at its maximum size.</td>
</tr>
<tr>
<td>WF_VSLSIZE</td>
<td>16</td>
<td><em>parm1</em> defines the size of the current slider relative to the size of the scroll bar as a value from 1 to 1000. A value of 1000 indicates that the slider is at its maximum size.</td>
</tr>
<tr>
<td>Mode</td>
<td>Value</td>
<td>Description</td>
</tr>
<tr>
<td>--------------</td>
<td>-------</td>
<td>-------------</td>
</tr>
<tr>
<td>WF_COLOR</td>
<td>18</td>
<td>This mode sets the current color of the window widget specified on entry in parm1. Valid window widget indexes are as follows (W_SMALLER is only valid as of AES 4.1):</td>
</tr>
<tr>
<td></td>
<td>parm1</td>
<td>Value</td>
</tr>
<tr>
<td></td>
<td>W_BOX</td>
<td>0</td>
</tr>
<tr>
<td></td>
<td>W_TITLE</td>
<td>1</td>
</tr>
<tr>
<td></td>
<td>W_CLOSER</td>
<td>2</td>
</tr>
<tr>
<td></td>
<td>W_NAME</td>
<td>3</td>
</tr>
<tr>
<td></td>
<td>W_FULLER</td>
<td>4</td>
</tr>
<tr>
<td></td>
<td>W_INFO</td>
<td>5</td>
</tr>
<tr>
<td></td>
<td>W_DATA</td>
<td>6</td>
</tr>
<tr>
<td></td>
<td>W_WORK</td>
<td>7</td>
</tr>
<tr>
<td></td>
<td>W_SIZER</td>
<td>8</td>
</tr>
<tr>
<td></td>
<td>W_VBAR</td>
<td>9</td>
</tr>
<tr>
<td></td>
<td>W_UPARROW</td>
<td>10</td>
</tr>
<tr>
<td></td>
<td>W_DNARROW</td>
<td>11</td>
</tr>
<tr>
<td></td>
<td>W_VSLIDE</td>
<td>12</td>
</tr>
<tr>
<td></td>
<td>W_VELEV</td>
<td>13</td>
</tr>
<tr>
<td></td>
<td>W_HBAR</td>
<td>14</td>
</tr>
<tr>
<td></td>
<td>W_LFARROW</td>
<td>15</td>
</tr>
<tr>
<td></td>
<td>W_RTARROW</td>
<td>16</td>
</tr>
<tr>
<td></td>
<td>W_HSLIDE</td>
<td>17</td>
</tr>
<tr>
<td></td>
<td>W_HELEV</td>
<td>18</td>
</tr>
<tr>
<td></td>
<td>W_SMALLER</td>
<td>19</td>
</tr>
</tbody>
</table>

The ob_spec field of the object (containing the color information) while the window is on top is defined in parm2. The ob_spec field for the object while the window is not on top is defined in parm3.

This mode is only valid as of AES version 0x0300.

| WF_DCOLOR   | 19    | This mode sets the default color of newly created windows as with WF_COLOR above. This mode only works as of AES version 0x0300. As of AES version 4.1, this mode causes all currently displayed windows which have not had their color explicitly set with WF_COLOR to be changed. |

| WF_BEVENT   | 24    | parm1, parm2, parm3, and parm4 are each interpreted as bit arrays whose bits indicate supported window features. Currently only one bit is supported. If bit 0 (B_UNTAPPABLE) of parm1 is set, the window will be set to be ‘un-tappable’ and it will never receive WM_TAPPED messages, only button clicks. |

This mode is only available as of AES versions 4.0.

| WF_BOTTOM   | 25    | This mode will place the specified window at the bottom of the window list (if there is more than one window) and top the new window on the top of the list. |

This mode is only available as of AES version 4.0.
WF_ICONIFY 26 This mode iconifies the specified window to the X, Y, width, and height coordinates given in parm1, parm2, parm3, and parm4 respectively. Normally, this happens as the result of receiving a WM_ICONIFY message.

This mode is only available as of AES version 4.1.

WF_UNICONIFY 27 This mode uniconifies the window specified, returning it to its original X, Y, width, and height as specified in parm1, parm2, parm3, and parm4 respectively. Normally, this happens as the result of receiving a WM_UNICONIFY message.

This mode is only available as of AES version 4.1.

WF_UNICONIFYXYWH 28 This mode sets the X, Y, width, and height that will be transmitted to the window with the next WM_UNICONIFY message that targets it. This call is used when a window is opened in an iconified state to give the OS a method of positioning it when it is uniconified.

This mode is only available as of AES version 4.1.

WF_TOOLBAR 30 This mode attaches a toolbar to the specified window. parm1 and parm2 contain the high and low WORD of the address of the toolbar OBJECT tree respectively. parm3 and parm4 are unused.

Set parm1 and parm2 to 0 to remove a toolbar.

Binding

```c
int in[0] = handle;
int in[1] = mode;
int in[2] = x;
int in[3] = y;
int in[4] = w;
int in[5] = h;

return crys_if(0x69);
```

Return Value wind_set() returns 0 if an error occurred or non-zero otherwise.

See Also wind_get()

wind_update()

WORD wind_update( mode )

WORD mode;

wind_update() manages the screen drawing semaphore.

Opcode 107 (0x6B)
**Availability**

All AES versions.

**Parameters**

`mode` specifies an action as follows:

<table>
<thead>
<tr>
<th>Name</th>
<th><code>mode</code></th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>END_UPDATE</td>
<td>0</td>
<td>This mode resets the flag set by BEG_UPDATE and should be called as soon as redrawing is complete. This will allow windows to be moved and menus to be dropped down again.</td>
</tr>
<tr>
<td>BEG_UPDATE</td>
<td>1</td>
<td>Calling this mode will suspend the process until no drop-down menus are showing and no other process is updating the screen. This will then set a flag which guarantees that the screen will not be updated and windows will not be moved until you reset it with END_UPDATE. Generally this call is made whenever a WM_REDRAW message is received to lock the screen semaphore while redrawing.</td>
</tr>
<tr>
<td>END_MCTRL</td>
<td>2</td>
<td>This mode releases control of the mouse to the AES and resumes mouse click message services.</td>
</tr>
<tr>
<td>BEG_MCTRL</td>
<td>3</td>
<td>This mode prevents mouse button messages from being sent to applications other than your own. form_do() makes this call to lock out screen functions. Desk accessories which display a dialog outside of a window must use this function to prevent button clicks from falling through to the desktop.</td>
</tr>
</tbody>
</table>

**Binding**

```c
int in[0] = mode;
return crys_if(0x6B);
```

**Return Value**

`wind_update()` returns 0 if an error occurred or non-zero otherwise.

**Version Notes**

As of AES version 4.0, you may logically OR a mask of NO_BLOCK (0x0100) to either BEG_UPDATE or BEG_MCTRL. This mask will prevent the application from blocking if another application currently has control of the screen semaphore. Instead, if another application has control, the function will immediately return with an error value of 0.

This method should only be used by timing-sensitive applications such as terminal programs in which a long redraw by another application could cause a timeout.

**Comments**

All `wind_update()` modes nest. For instance, to release the screen semaphore, the same number of END_UPDATE calls must be received as were BEG_UPDATE calls. It is recommended that you design your application in a manner that avoids nesting these calls.

Both the BEG_UPDATE and BEG_MCTRL modes should be used prior to displaying a form or popup to prevent them from being overwritten or clicks to them being sent to other applications.
Always wait until \textit{after} the \texttt{BEG\_UPDATE} call to turn off the mouse cursor when updating the screen to be sure you have gained control of the screen.

Applications such as slide-show viewers which require the whole screen area (and may need to change screen modes) may call \texttt{wind\_update()} with parameters of both \texttt{BEG\_UPDATE} and \texttt{BEG\_MCTRL} to completely lock out the screen from other applications. The application would still be responsible for saving the screen area, manipulating video modes as necessary, restoring the screen when done, and returning control of the screen to other applications with \texttt{END\_UPDATE} and \texttt{END\_MCTRL}.

\begin{footnotesize}
\textbf{SEE ALSO} \quad \texttt{wind\_new()}
\end{footnotesize}
Overview

The Virtual Device Interface (VDI) is a collection of drivers designed to provide applications with a device-independent method of accessing graphically based devices such as monitors, printers, and plotters. Applications which are written to use the VDI rather than directly accessing hardware will be compatible with all currently available devices including those which have not yet been developed.

All Atari systems with TOS in ROM include a VDI screen driver adaptable to each display resolution the system can support. Soft-loaded screen drivers and drivers for other devices are loaded through a VDI sub-system called the Graphics Device Operating System (GDOS).

The GDOS system is disk-loaded as a TSR utility at bootup. It loads device drivers based upon the contents of its configuration file(s).

Applications wishing to use the GDOS extensions must verify its presence using the method described later in this chapter. If an application’s output will be limited to the screen and no font other than the system font is needed, then the presence of GDOS is not mandatory.

VDI Workstations

Every system call made to the VDI must include a workstation handle. This handle is a unique integer which identifies the device and current attribute array. Workstation handles are returned by the VDI calls v_opnwk() or v_opnvwk().

Workstations provide a lookup array of attributes such as line width, text color, clipping state, etc. that are unique to it.

Physical Workstations

Each device must be initialized by opening its physical workstation. Opening a physical workstation causes all drawing and clipping attributes to be reset and the current page (display) to be reset to the default background color. Only one physical workstation may be opened to a single device at any given time.

The screen device’s physical workstation is automatically initialized by the AES upon bootup. Its physical workstation handle may be obtained from the AES call graf_handle().

Devices such as printers and plotters must have their physical workstation opened by the application wishing to utilize them. When opening a physical workstation the application must specify a device ID which identifies the device to open. Device identification codes are assigned as follows:
These values correspond to the value listed in the leftmost column of the user’s ‘ASSIGN.SYS’ file. The following code segment demonstrates opening a physical workstation to the printer device with ID #21. It is important to note that the function assumes that the presence of GDOS has been tested for and was verified.

\[
\text{work_in}[0] \text{ is set to the desired device ID and work_in[1-9] are filled in with common defaults for workstation attributes. work_in[10] is set to 2 to indicate raster coordinates as explained later in this chapter. The function returns a non-zero value if an error occurred.}
\]

\[
\begin{align*}
\text{WORD work_in[11],work_out[57];} \\
\text{WORD handle;}
\end{align*}
\]

\[
\begin{align*}
\text{WORD printer_open( VOID )} \\
\text{ { } } \\
\text{WORD i;}
\end{align*}
\]

\[
\begin{align*}
\text{work_in[0] = 21;} \\
\text{for(i = 1;i < 10; work_in[i++] = 1);} \\
\text{work_in[10] = 2;} \\
\text{v_opnwk(work_in,&handle,work_out);} \\
\text{return (handle == 0);} \\
\end{align*}
\]

### Virtual Workstations

Each physical workstation may have multiple virtual workstations opened which allow individual applications to maintain separate workstation attributes. In fact, a single application may open multiple virtual workstations to the same device to manage workstation attributes more efficiently. Opening a virtual workstation does not affect the current contents of the display.

Most GEM applications will open a virtual workstation to the current screen device upon initialization. The following code segment illustrates opening a virtual workstation to the display device.

The device identification code for the display device must be specified as \text{Getrez()} + 2 for all VDI features to work correctly. All other parameters are passed the same as the example for
opening a physical workstation except that *handle* must contain the physical workstation handle of the device for which you wish to obtain a virtual workstation handle.

A more programmer-friendly method of opening workstations involves the use of the **VDI_Workstation** structure which is discussed in the reference entry for **V_Opnvwk()**

```c
WORD work_in[11], work_out[57];
WORD handle;
WORD wcell, hcell, wbox, hbox;

WORD
screen_open( VOID )
{
    WORD i;

    handle = graf_handle( &wcell, &hcell, &wbox, &hbox);
    work_in[0] = Getrez() + 2;
    for(i = 1; i < 10; work_in[i++] = 1);
    work_in[10] = 2;
    v_opnvwk(work_in, &handle, work_out);
    return (handle == 0);
}
```

### Workstation Specifics

**Coordinate Systems**

The **VDI** defaults to the usage of Raster Coordinates (RC) which places the origin at the upper-left of the page or display. As an example, the coordinate range for the 1040ST’s monochrome graphics mode is shown here:

![Coordinate Range](image)

RC coordinate ranges vary with the device. It is up to the application to interpret and scale the size and position of its output appropriately.

With the addition of **GDOS**, the **VDI** gains the ability to utilize Normalized Device Coordinates (NDC). When using NDC, **GDOS** translates and scales all coordinates to the device as...
appropriate. All devices using NDC will have their origin at the lower-left hand corner of the display or page as follows:

( 32767, 32767 )

( 0, 0 )

Using NDC provides an excellent manner of reducing the overhead of having to internally scale every coordinate, however, applications which depend on the proper aspect ratio for their output should consider managing coordinates internally.

Rendering Graphics

Each VDI output function uses attributes set by other related VDI functions to determine characteristics such as line width, text face, and color. The following table lists VDI attribute calls and the functions they affect.

To output a VDI object, set each attribute as desired and then make the appropriate call. For example, to output a line of text in the System font at 9 point colored red, make the following sequence of calls.

```c
vst_font( handle, 1 ); /* Select the System Font */
vst_point( handle, 9 );
vst_color( handle, 2 );
v_ftext( handle, 10, 10, "The Atari Compendium" );
```

Generalized Device Primitives

GDP’s (Generalized Device Primitives) are basic drawing components available through the VDI. All current device drivers support all GDP’s though specialized drivers may not be able to. `intout[14-24]` may be used to determine the presence of GDP’s. Currently there are 10 supported GDP’s as follows:
### VDI Rectangles

Several VDI functions require that a rectangle in VDI format be passed to them. VDI rectangles are different from AES rectangles in the manner in which they are specified.

To correctly define a VDI rectangle you must specify two coordinate pairs one representing the upper-left point of the rectangle and the other specifying the lower-right as follows:

\[(x_1, y_1) \quad \text{to} \quad (x_2, y_2)\]

The following two functions provide simple conversion between AES GRECTs and VDI rectangles in an array.

```c
VOID Grect2xy( GRECT *g, short *pxy)
{
    pxy[0] = g.g_x;
    pxy[1] = g.g_y;
    pxy[2] = g.g_x + g.g_w - 1;
    pxy[3] = g.g_y + g.g_h - 1;
}

VOID Xy2Grect( short *pxy, GRECT *g )
{
    g.g_x = pxy[0];
    g.g_y = pxy[1];
    g.g_w = pxy[2] - pxy[0] + 1;
    g.g_h = pxy[3] - pxy[1] + 1;
}
```
Device Types vs. Required Functions
Not all VDI functions are supported by all drivers. The presence of GDP functions may be checked using the information returned in the \textit{intout} array after a \texttt{v\_opnwk()} call. Other calls may be checked for by entering a test call and comparing returned information with what would be expected.

In addition, each type of driver has a certain number of required functions which must be supported by the device. Each entry in the \textit{VDI Function Reference} specifies the support required for a function.

Write Modes
All VDI graphics primitives are subject to one of four writing modes set by \texttt{vswr\_mode()}, with the exception of \texttt{vro\_cpyfm()} which is passed one of sixteen writing modes.

The following logic tables illustrate the effects of each of the four primary modes. Graphic examples can be found under the reference entry for \texttt{vswr\_mode()}.

<table>
<thead>
<tr>
<th>Mode</th>
<th>Logic</th>
</tr>
</thead>
<tbody>
<tr>
<td>Replace</td>
<td>Destination = Source</td>
</tr>
<tr>
<td>Transparent</td>
<td>Destination = Source OR Destination</td>
</tr>
<tr>
<td>XOR</td>
<td>Destination = Source XOR Destination</td>
</tr>
<tr>
<td>Reverse Transparent</td>
<td>Destination = (NOT Source) AND Destination</td>
</tr>
</tbody>
</table>

Using Color
The color capabilities of VDI devices can be placed into three categories as follows. Determining which category a device falls into is accomplished by examining the return values from \texttt{v\_opnvwk()}, \texttt{v\_opnwk()}, and \texttt{vq\_extnd()}.

<table>
<thead>
<tr>
<th>Categories</th>
<th>\texttt{v_opn/v/wk()}</th>
<th>\texttt{vq_extnd()}</th>
</tr>
</thead>
<tbody>
<tr>
<td>Monochrome Device(^1)</td>
<td>2</td>
<td>0</td>
</tr>
<tr>
<td>Palette-Based Device</td>
<td>(\geq 2)</td>
<td>1</td>
</tr>
<tr>
<td>True Color Device</td>
<td>(&gt; 2)</td>
<td>0</td>
</tr>
</tbody>
</table>

\(^1\)Sometimes monochrome devices appear as palette-based devices with two available colors.
Monochrome Devices

Monochrome devices are only capable of displaying one color. Often, monochrome devices are instead represented by palette-based devices with two fixed colors.

Palette-Based Devices

Palette-based devices have a fixed number of colors that may be rendered on screen simultaneously. Each pixel value is used to index into the palette to decide what color to display. For instance, if you change VDI color #2 to green, draw a box with that color index, and then change VDI color #2 to red, the box will appear first in green and then turn red.

The first 16 VDI color registers are used by the operating system and should be avoided. If your application must change them, they should be restored when no longer needed.

True Color Devices

True-color devices allow each pixel to have a unique color value. Rather than palette entries, colors \(\text{work\_out}[13]\) corresponds to the number of available virtual pens. Drawing is accomplished by using these pens, however, unlike using a palette, changing the color of a pen does not change any pixel’s color drawn with that pen on the screen.

Whatever color is stored in virtual pen #0 is considered the background color for the purpose of computing write modes.

It is possible for external devices, printers, plotters, etc. to behave as if they were a true-color device.

Color Mapping

Color values are defined in the VDI by specifying a red, green, and blue value from 0–1000. The VDI will scale the value to the closest color possible. \(\text{vq\_color()}\) can be used to determine the actual color that was set.

VDI Raster Forms

The VDI handles raster forms using three commands, \(\text{vro\_cpyfm()}, \text{vrt\_cpyfm()}, \) and \(\text{vr\_trnfm()}\). \(\text{vro\_cpyfm()}\) and \(\text{vrt\_cpyfm()}\) are responsible for ‘blitting’ raster images between memory and a workstation. These functions may also be used to copy images from one location on a workstation to another. ‘Blitting’ is the process of copying memory from one location to another. Atari computers use the BLiTTER chip (when one is installed) or a software bit blit algorithm to quickly move memory. While these calls are designed to transfer screen memory, if carefully used, they may also be used to transfer other types of memory as well.

\(\text{vr\_trnfm()}\) is responsible for the transformation of images between device-specific and VDI standard format, the two raster image formats recognized by the VDI. Device-specific format is limited to images in the format of the source device whereas the second is a generic format recommended for transporting images to non-standard displays.
VDI Device-Specific Format

Device-specific format simply mimics the layout of pixels and planes on the source device. When using `vro_cpyfm()` and `vrt_cpyfm()` the source form will be transferred to the destination form in device-specific format\(^2\).

If you intend to save images to disk you should first utilize `vr_trnfm()` to transform the image into a VDI standard format so that the image can be successfully ported to any display.

VDI Standard Format

VDI standard format is designed to provide a portable method of specifying raster images which may be displayed on any device. Images stored in VDI standard format must be transformed with `vr_trnfm()` before copying them to a workstation.

Images in VDI standard format appear in memory in a plane-by-plane fashion. All of the bits for plane #0 appear first followed by the bits for plane #1, and so on for as many planes as exist in the image.

Images may be easily transferred to devices with a higher number of planes by simply inserting empty bytes to account for planes not present in the source image. This method will only work, however, with palette based devices.

Vector Handling

The VDI screen driver is also responsible for managing some hardware vectors responsible for keyboard and mouse input. The functions available for altering these vectors are `vex_motv()`, `vex_timv()`, `vex_curv()`, and `vex_butv()`. For further explanation of these calls please see the VDI Function Reference.

Use of these functions is not recommended with MultiTOS as these vectors are global and affect all applications. In addition, results are undefined if two or more non-resident applications utilized these calls at once.

Existing applications which use these calls must have their program flags set to either supervisor or global memory protection. See the GEMDOS Overview for a discussion of the program flags.

---

\(^2\)The definitions of `vro_cpyfm()` and `vrt_cpyfm()` allow for the specification of the format of the source and destination form, however, this feature is not currently supported by any version of the operating system. Any call which specifies either the source or destination form to be in device-independent format will fail.
The Graphics Device Operating System (GDOS) is a disk-based component of the operating system which allows disk-loadable device drivers and additional fonts to be accessible through standard VDI calls.

Several versions of Atari GDOS have been released in addition to several third-party GDOS ‘clones’. All of these forms have stayed backward-compatible with GDOS 1.0, however it is recommended that programs be written to support newer GDOS calls when it can be determined that a more recent release of GDOS is present.

Each VDI call documented in the VDI Function Reference specifies if GDOS is required, and if so, what type.

Determining the Version of GDOS Present

A non-standard VDI call is available to check for the presence of GDOS. The following machine-code subroutine will return a longword result in d0 which can be used to determine the variety of GDOS present. Beware of older bindings which looked only for the original GDOS and returned a 1 or 0 as a result.

```
.text
_vq_gdos:
    move.l #-2,d0
    trap  #2
    rts
.end
```

The longword return value in d0 can be interpreted as follows:

<table>
<thead>
<tr>
<th>Name</th>
<th>Value</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>GDOS_NONE</td>
<td>-2</td>
<td>No GDOS is installed.</td>
</tr>
<tr>
<td>—</td>
<td>Any other value.</td>
<td>Original GDOS 1.x is installed.</td>
</tr>
<tr>
<td>GDOS_FNT</td>
<td>0x5F464E54 '_FNT'</td>
<td>FONTGDOS is installed.</td>
</tr>
<tr>
<td>GDOS_FSM</td>
<td>0x5F46534D '_FSM'</td>
<td>FSM GDOS or SpeedoGDOS is installed. For information on determining the specific variety of outline GDOS available, see the description of the ‘FSMC’ cookie in Chapter 3: BIOS</td>
</tr>
</tbody>
</table>
7.12 – VDI

FSM GDOS vs. SpeedoGDOS

Since FSMGDOS (a QMS/Imagen outline font-based GDOS) was never officially released from Atari (though shipped in limited quantity with third-party products), some changes have been made to calls in SpeedoGDOS that were never exploited by developers. For that reason, these calls will only be documented in the Speedo-compatible way in the VDI Function Reference. This does mean, however, that use of these calls will cause your application to fail under the original FSMGDOS.

The calls which were affected are v_getoutline(), v_getbitmap_info(), v_killoutline(), and vqt_get_table(). In addition, use of the new SpeedoGDOS calls vst_charmap(), vqt_trackkern(), vqt_pairkern(), vqt_fontheader(), vst_kern(), or any of the older calls when used with the fix31 data type will fail with the older FSM.

To determine the type of outline-font GDOS installed, look for the ‘FSMC’ cookie. The cookie value is a pointer to a longword which contains the character string ‘_FSM’ for Imagen-based FSMGDOS or ‘_SPD’ for Speedo-based FSMGDOS.

GDOS 1.x

GDOS 1.0 and the other 1.x versions which followed it was the original GDOS developed by Digital Research for Atari. It handled only bitmap fonts and was slow compared to the newer FONTGDOS which now replaces it.

When a v_opnwk() call is made with GDOS installed, a check is done to see if a driver was assigned to the device ID specified in the ‘ASSIGN.SYS’ file, and if so, loaded.

All VDI calls which specify the returned handle will subsequently be redirected to the driver.

Not all VDI functions are available with every driver. Check the ‘Availability’ heading for each specific function in the VDI Function Reference for specific availability.

Bitmap Fonts

Bitmap fonts have the ability to be quickly rendered and highly accurate. They do generally require more disk space and a font file must be available for each point size and aspect ratio required. Bitmap fonts follow a special naming convention as follows:

```
ATSS12LS.FNT
```

The vendor code is a unique two-letter identifier which specifies the creator of the font. The font code is a two-letter code which abbreviates the font’s name. The point size field specifies the point size of the font. The device type is a two-letter abbreviation which should match the aspect ratio of the device as follows:
Device Type | Destination Ratio
---|---
None or HI | 91x91 (Screen Devices)
CG | 91x45 (Screen Devices)
LS | 300x300 (Laser Printers, Inkjets)
EP | 120x144 (Lo-Res Dot-Matrix Printers)
LB | 160x72 (Lo-Res Dot-Matrix Printers)
SP | 180x180 (Med-Res Dot-Matrix Printers)
QD | 240x216 (Med-Res Dot-Matrix Printers)
NP | 360x360 (High-Res Dot-Matrix Printers)

For a driver to recognize a bitmap font it must be listed in the user’s ‘ASSIGN.SYS’ file and be of the correct aspect ratio. No extra fonts are made available to applications until a vst_load_fonts() call is made.

**FONTGDOS**

**FONTGDOS** is the successor to GDOS 1.x. As with the original GDOS, FONTGDOS supports only bitmap fonts. Its differences are improved driver support, support for bezier curves, improved error handling, and a much quicker response time.

**Bezier Curves**

FONTGDOS conforms to the PC-GEM/3 file standard with the inclusion of bezier curve rendering capability with the v_bez() and v_bez_fill() calls. v_bez_on() must be used to allow FONTGDOS to allocate the memory necessary for bezier rendering. Likewise v_bez_off() should be used before an application exits to free any memory used for bezier calls.

**Error Support**

When GDOS 1.x encountered an error condition, it simply wrote an error message at the top of the display overwriting a portion of the menu bar and display screen. FONTGDOS allows an application to disengage this behavior and instead return error codes in a global variable. It is then the applications responsibility to check this variable after calls which may cause an error condition. See the *VDI Function Reference* call vst_error() for more information.

**FSMGDOS**

FSMGDOS was developed by Atari in conjunction with QMS/Imagen Corp. to provide Imagen outline fonts which could be displayed at any point size, aspect ratio, or device. It provided all of the improved features of FONTGDOS with outline fonts and caching capability. This version of GDOS was, however, never officially released. Third-party manufacturers did ship many copies of this GDOS to the public. In addition, many developers did update their products to utilize the special features of FSMGDOS.

Most VDI function calls added with this version of GDOS have remained compatible with SpeedoGDOS, however, some calls which were never used by developers were changed.
means that applications written to support SpeedoGDOS may not be backwardly compatible. For specific compatibility information, consult the *VDI Function Reference*.

### SpeedoGDOS

**SpeedoGDOS** is a new variety of **FSM** which employs outline font technology from Bitstream using Speedo-format outline fonts. In addition, several new calls were added to gain access to internal font information and provide true WYSIWYG (What-You-See-Is-What-You-Get) output.

#### The fix31 Data Type

**SpeedoGDOS** optionally allows the use of the **fix31** data type in some calls for parameters and return values. Old bindings designed for the Imagen-based **FSM** will still function properly. Newer bindings may be written to take advantage of this data type.

The **fix31** data type allows for the internal representation and manipulation of floating-point values without the use of a floating-point library. It is a 32-bit value with a 1-bit sign and a 31-bit magnitude. Each value specifies a number in 1/65536 pixels. Examples of this data type follow:

<table>
<thead>
<tr>
<th>fix31</th>
<th>Floating Point</th>
</tr>
</thead>
<tbody>
<tr>
<td>0x00010000</td>
<td>1.0</td>
</tr>
<tr>
<td>0xFFFF0000</td>
<td>-1.0</td>
</tr>
<tr>
<td>0x00018000</td>
<td>1.5</td>
</tr>
</tbody>
</table>

Character advances can be simply be added or subtracted to each other using integer arithmetic. To convert a **fix31** unit to an integer (rounding to 0) use the following code:

```c
x_integer = (WORD)(x_fix31 >> 16);
```

To convert a **fix31** to an integer and round it to the closest integer use the following code:

```c
x_integer = (WORD)((x_fix31 + 32768) >> 16);
```

Use of **fix31** values provides higher character placement accuracy and access to non-integer point sizes. For specific implementation notes, see the *VDI Function Reference* entries for **vqt_advance32()**, **v_getbitmap_info()**, **vst_arbpt32()**, and **vst_setsize32()**.

#### Kerning

**SpeedoGDOS** outline fonts have the ability to be kerned using two methods. Track kerning is global for an entire font and has three settings, normal, tight, and extra tight. Pair kerning works for individual pair groups of characters. In addition, new pairs may be defined as necessary to produce the desired output.

Kerning is taken into account with **v_ftext()** and **vqt_advance()** only when enabled. Use the calls **vst_kern()**, **vqt_pairkern()**, and **vqt_trackkern()** to access kerning features.
Caching

All SpeedoGDOS extent and outline rendering calls are cached for improved performance. Cache files may be loaded or saved to disk as desired to preserve the current state of the cache. In addition, an application might want to flush the cache before doing an output job to a device such as a printer to improve performance with new fonts.

The call vqt_cachesize() can be used to estimate the ability of the cache to store data for an unusually large character and prevent memory overflow errors.

Special Effects

The call vst_scratch() determines the method used when calculating the size of the special effects buffer. In general an application should not allow the user to use algorithmically generated effects on Speedo fonts. In most cases, special effects are available by simply choosing another font.

The problem is that Speedo fonts may be scaled to any size and SpeedoGDOS has no way of predicting the upper-limit on the size of a character to allocate special effects memory. Currently, SpeedoGDOS allocates a buffer large enough to hold the largest character possible from the point sizes in the ‘ASSIGN.SYS’ file and those listed in the ‘EXTEND.SYS’ file. If your application limits special effects to these sizes then no problems will occur.

If you intend to restrict users to using special effects only with bitmap fonts you may call vst_scratch() with a mode parameter of 1, memory allocation will be relaxed to only take bitmap fonts into account. You may also specify a mode parameter of 2 if you plan to allow no special effects at all. The vst_scratch() call must be made prior to calling vst_load_fonts().

Speedo Character Indexes

Speedo fonts contain more characters than the Atari ASCII set can define. Fonts may be re-mapped with a CPX using the vqt_get_table() call (this method is not recommended on an application basis as this call affects all applications in the system).

Another method involves the use of a new call, vst_charmap(). Calling this function with a mode parameter of 0 causes all functions which take character indexes (like v_ftext(), vqt_width(), etc.) to interpret characters as WORDs rather than BYTEs and utilize Speedo International Character Encoding rather than ASCII.

The Function Reference provides two alternate bindings for v_ftext() and v_ftext_offset() called v_ftext16() and v_ftext_offset16() which correctly output 16-bit Speedo character text rather than 8-bit ASCII text.

A complete listing of the Bitstream International Character Set is listed in Appendix G: Speedo Fonts.

Speedo Font IDs
The function `vqt_name()` is used with all versions of GDOS to return a unique integer identifier for each font. Because some bitmap font ID’s conflicted with Bitstream outline font ID’s, SpeedoGDOS versions 4.20 and higher add 5000 to each of the outline font ID’s to differentiate them from bitmap fonts.

## Device Drivers

### Printer and Plotter Drivers

Printer drivers are the most common form of GDOS driver available, though some plotter drivers do exist. The *VDI Function Reference* can be used to determine if a particular function call is required to be available on a particular device. This does not, however, prohibit the addition of supplementary functions.

Some special printer driver features are available with drivers designed to support them as follows:

#### Dot-Matrix Printers

Dot-matrix printers with wide carriages can have their print region expanded by passing a custom X and Y resolution for the driver in `ptsin[0]` and `ptsin[1]` respectively prior to the `v_opnwk()` call. In addition, `ctrl[1]` should be set to 1 to indicate the presence of the parameters.

**SLM804**

After a `v_opnwk()` call to an SLM804 driver `ctrl[0]` will contain the MSB and `ctrl[1]` will contain the LSB of the allocated printer buffer address.

After a `v_updwk()` call, `intout[0]` will contain a printer status code as follows:

<table>
<thead>
<tr>
<th>Name</th>
<th>Error Code</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>SLM_OK</td>
<td>0x00</td>
<td>No Error</td>
</tr>
<tr>
<td>SLM_ERROR</td>
<td>0x02</td>
<td>General Printer Error</td>
</tr>
<tr>
<td>SLM_NOTONER</td>
<td>0x03</td>
<td>Toner Empty</td>
</tr>
<tr>
<td>SLM_NOPAPER</td>
<td>0x05</td>
<td>Paper Empty</td>
</tr>
</tbody>
</table>
All Printer Drivers

A user-defined printer buffer may be passed to the `v_updwk()` call by specifying the address of the buffer in `intin[0]` and `intin[1]`. In addition, `contrl[3]` must be set to 2 to indicate the new parameters and `contrl[1]` must be set to 1 to instruct the VDI to not clear the buffer first.

Camera and Tablet Drivers

As of this writing, no camera or tablet drivers existed for Atari GEM. Several functions are reserved to support them which were developed under PC-GEM, however, many remain undocumented. Where documentation was available, those calls are included for completeness in the VDI Function Reference.

The Metafile Driver

‘META.SYS’ drivers are specially designed drivers which create ‘.GEM’ disk files rather than produce output on a device. When a metafile device is opened, the file ‘GEMFILE.GEM’ is created in the current GEMDOS path. The function `vm_filename()` may be used to change the filename to which the metafile is written to, however, the file ‘GEMFILE.GEM’ must be deleted by the application.

When a metafile is opened, several defaults relating to the coordinate space and pixel size are set. Each pixel is assigned a default width and height of 85 microns (1 micron = 1/25400 inch). This equates to a default resolution of 300dpi.

The device size is specified where Normalized Device Coordinates (NDC) = Raster Coordinates (RC). The coordinate space of the metafile has (0, 0) in the lower-left corner and (32767, 32767) in the upper-right. This coordinate system may be modified with `vm_coords()`. The size of the actual object space being written to the metafile should also be specified with `vm_pagesize()` so that an application may correctly clip the objects when reading.

After changing coordinate space, values returned by `vq_extnd()` related to pixel width, height and page size will not change. Also, font metrics returned by functions such as `vqt_fontinfo()` and `vqt_advance()` will remain based on the default metafile size information. In most cases, text metric information should be embedded based on the workstation metrics of the destination device (such as a screen or printer) anyway.

The metafile is closed when a `v_clswk()` call is issued. Other applications which read metafiles will play back the file by issuing commands in the same order as recorded by the driver. For more information on the metafile format see Appendix C: Native File Formats.
The Memory Driver

‘MEMORY.SYS’ includes all of the standard VDI calls yet works only in memory and is not designed to be output to a device. Normally, the memory driver should be assigned in the user’s ‘ASSIGN.SYS’ file as device number 61. Upon calling v_opnwk() to the memory driver, contrl[1] should be set to 1 and ptsin[0] and ptsin[1] should contain the X and Y extent of the memory area. Upon return from the call, contrl[0] and contrl[1] will contain the high and low WORD respectively of the address of the memory device raster. v_updwk() clears the raster.

VDI Function Calling Procedure

The GEM VDI is accessed through a 68x00 TRAP #2 statement. Prior to the TRAP, register d0 should contain the magic number 0x73 and register d1 should contain a pointer to VDI parameter block. An example binding is as follows:

```
.text
_vdi:
  move.l #_VDIpb,d1
  move.l #$73,d0
  trap #2
  rts
```

The VDI parameter block is an array of 5 pointers which each point to a specialized array of WORD values which contain input parameters and function return values. Different versions of the VDI support different size arrays. The following code contains the ‘worst case’ sizes for these arrays. Many newer versions of the VDI support larger array sizes. You can inquire what the maximum array size that VDI supports by examining the work_out array after a v_opnvwk() or v_opnwk(). Larger array sizes allow more points to be passed at a time for drawing functions and longer strings to be passed for text functions. The definition of the VDI parameter block follows:

```
.data
  _contrl: ds.w 12
  _intin: ds.w 128
  _ptsin: ds.w 256
  _intout: ds.w 128
  _ptsout: ds.w 256
  _VDIpb: dc.l _contrl, _intin, _ptsin
           dc.l _intout, _ptsout
.end
```

The contrl array contains the opcode and number of parameters being passed the function as follows:

<table>
<thead>
<tr>
<th>contrl[x]</th>
<th>Contents</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>Function Opcode</td>
</tr>
<tr>
<td>1</td>
<td>Number of Input Vertices in ptsin</td>
</tr>
<tr>
<td>2</td>
<td>Number of Output Vertices in ptsout</td>
</tr>
<tr>
<td></td>
<td>Description</td>
</tr>
<tr>
<td>---</td>
<td>------------------------------------------------</td>
</tr>
<tr>
<td>3</td>
<td>Number of Parameters in <code>intin</code></td>
</tr>
<tr>
<td>4</td>
<td>Number of Output Values in <code>intout</code></td>
</tr>
<tr>
<td>5</td>
<td>Function Sub-Opcode</td>
</tr>
<tr>
<td>6</td>
<td>Workstation Handle</td>
</tr>
<tr>
<td>7-11</td>
<td>Function Specific</td>
</tr>
</tbody>
</table>


For specific information on bindings, see the *VDI Function Reference*. 
VDI/GDOS Function Reference
v_alpha_text()

VOID v_alpha_text( handle, str )
WORD handle;
char *str;

v_alpha_text() outputs a line of alpha text.

OPCODE 5

SUB-OPCODE 25

AVAILABILITY Supported by all printer and metafile drivers.

PARAMETERS handle is a valid workstation handle. str is a pointer to a null-terminated text string which will be printed. Two special BYTE codes may be embedded in the text. ASCII 12 will cause a printer form-feed. ASCII 18 ‘DC2’ will initiate an escape sequence followed by a command descriptor BYTE (in ASCII) indicating which action to take as follows.

<table>
<thead>
<tr>
<th>Command BYTE</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>‘0’</td>
<td>Enable bold print.</td>
</tr>
<tr>
<td>‘1’</td>
<td>Disable bold print.</td>
</tr>
<tr>
<td>‘2’</td>
<td>Enable italic print.</td>
</tr>
<tr>
<td>‘3’</td>
<td>Disable italic print.</td>
</tr>
<tr>
<td>‘4’</td>
<td>Enable underlining.</td>
</tr>
<tr>
<td>‘5’</td>
<td>Disable underlining.</td>
</tr>
<tr>
<td>‘6’</td>
<td>Enable superscript.</td>
</tr>
<tr>
<td>‘7’</td>
<td>Disable superscript.</td>
</tr>
<tr>
<td>‘8’</td>
<td>Enable subscript.</td>
</tr>
<tr>
<td>‘9’</td>
<td>Disable subscript.</td>
</tr>
<tr>
<td>‘A’</td>
<td>Enable NLQ mode.</td>
</tr>
<tr>
<td>‘B’</td>
<td>Disable NLQ mode.</td>
</tr>
<tr>
<td>‘C’</td>
<td>Enable wide printing.</td>
</tr>
<tr>
<td>‘D’</td>
<td>Disable wide printing.</td>
</tr>
<tr>
<td>‘E’</td>
<td>Enable light printing.</td>
</tr>
<tr>
<td>‘F’</td>
<td>Disable light printing.</td>
</tr>
<tr>
<td>‘W’</td>
<td>Switch to 10-cpi printing.</td>
</tr>
<tr>
<td>‘X’</td>
<td>Switch to 12-cpi printing.</td>
</tr>
<tr>
<td>‘Y’</td>
<td>Toggle compressed printing.</td>
</tr>
<tr>
<td>‘Z’</td>
<td>Toggle proportional printing.</td>
</tr>
</tbody>
</table>
WORD i = 0;

while(intin[i++] = (WORD)*str++);

ctrl[0] = 5;
ctrl[1] = 0;
ctrl[3] = --i;
ctrl[5] = 25;
ctrl[6] = handle;

vdi();

C A V E A T S

The line of text must not exceed the maximum allowable length of the \textit{intin} array as returned by \texttt{vq\_extnd()} or the maximum length of your compilers’ array.

C O M M E N T S

Only commands ‘0’, ‘1’, ‘2’, ‘3’, ‘4’, and ‘5’ are available with most printer drivers.

S E E A L S O

\texttt{v\_gtext()}, \texttt{v\_ftext()}

\textbf{v\_arc()}

\texttt{VOID v\_arc( handle, x, y, radius, startangle, endangle )}

\texttt{WORD handle, x, y, radius, startangle, endangle;}

\texttt{v\_arc()} outputs an arc to the specified workstation.

O P C O D E

11

S U B -O P C O D E

2

A V A I L A B I L I T Y

Supported by all drivers. This function composes one of the 10 \textbf{VDI GDP’s} (Generalized Drawing Primitives). Although all current drivers support all GDP’s, their availability is not guaranteed and may vary. To check for a particular GDP refer to the table returned by \texttt{v\_opnvwk()} or \texttt{v\_opnwk()}. 
PARAMETERS

handle is a valid workstation handle. $x$ and $y$ specify the center of an arc with a radius of radius and starting and ending angles of startangle and endangle specified in tenths of degrees as follows:

```
   900
    |
    |
    |
    |
    |
   2700
```

BINDING

```c
const[0] = 11;
const[1] = 4;
const[6] = handle;

intin[0] = startangle;
intin[1] = endangle;

ptsin[0] = x;
ptsin[1] = y;
ptsin[6] = radius;
ptsin[7] = 0;
```

vdi();

SEE ALSO

vsl_color()

---

**v_bar()**

VOID v_bar(handle, pxy)

WORD handle;

WORD *pxy;

v_bar() outputs a filled rectangle to the specified workstation.

OPCODE

11

SUB-OPCODE

1

AVAILABILITY

Supported by all drivers. This function composes one of the 10 VDI GDP’s (Generalized Drawing Primitives). Although all current drivers support all GDP’s, their availability is not guaranteed and may vary. To check for a particular GDP refer to the table returned by v_opnvwk() or v_opnwk().
**PARAMETERS**

`handle` is a valid workstation handle. `pxy` points to an array of four `WORD`s specifying a VDI format rectangle to output.

**BINDING**

```c
contr1[0] = 11;
contr1[1] = 2;
contr1[3] = 0;
contr1[5] = 1;
contr1[6] = handle;
```

```c
ptsin[0] = pxy[0];
ptsin[1] = pxy[1];
ptsin[2] = pxy[2];
ptsin[3] = pxy[3];
```

```c
vdi();
```

**COMMENTS**

This function, as opposed to `vr_recfl()`, *does* take the setting of `vsf_perimeter()` into consideration.

**SEE ALSO**

`vsf_interior()`, `vsf_style()`, `vsf_color()`, `vsf_perimeter()`, `vsf_udpat()`

---

### v_bez()

**VOID**  
`v_bez( handle, count, pxy, bezarr, extent, totpts, totmoves )`

**WORD** `handle, count;`

**WORD** `*pxy, *extent;`

**char** `*bezarr;`

**WORD** `*totpts, *totmoves;`

`v_bez()` outputs a bezier curve path.

**OPCODE**

6

**SUB-OPCODE**

13

**AVAILABILITY**

Available only with `FONTGDOS`, `FSMGDOS` or `SpeedoGDOS`.

**PARAMETERS**

`handle` is a valid workstation handle. `count` specifies the number of vertices in the path. `pxy` is a pointer to a `WORD` array (`count * 2`) `WORD`s long containing the vertices where `pxy[0]` is the X coordinate of the first point, `pxy[1]` is the Y coordinate of the first point and so on. `bezarr` is a pointer to a character array `count` `BYTE`s long where each byte is a bit mask with two flags that dictate the interpretation of each vertex as follows:
Upon exit, a 4 WORD array pointed to by `extent` is filled in with a VDI format rectangle defining a bounding box of the path drawn. The WORD pointed to by `totpts` is filled in with the number of points in the resulting path whereas the total number of moves is stored in the WORD pointed to by `totmoves`.

```c
WORD i;
contrl[0] = 6;
contrl[1] = count;
contrl[3] = (count + 1)/2;
contrl[5] = 13;
contrl[6] = handle;

for(i = 0; i < count; i++)
{
    intin[i] = (WORD)bezarr[i];
    ptsin[ i*2 ] = pxy[ i*2 ];
    ptsin[ (i+2) + 1 ] = pxy[ (i+2) + 1 ];
}

vdi();

*totpts = intin[0];
*totmoves = intin[1];

for(i = 0; i < 4; i++)
    extent[i] = ptsout[i];
```

**See Also**

v_bez_fill(), v_bez_on(), v_bez_off(), v_bez_qual(), v_set_app_buff()

---

**v_bez_fill()**

VOID v_bez_fill( handle, count, pxy, bezarr, extent, totpts, totmoves )

WORD handle, count;
WORD *pxy, *extent;
char *bezarr;
WORD *totpts, *totmoves;

v_bez_fill() outputs a filled bezier path.

**Opcode**

9
### v_bez_off()

**VOID v_bez_off( handle )**

**WORD handle;**

v_bez_off() disables bezier capabilities and frees associated memory.

---

**STATUS** 11

**SUB-OPCODE** 13

**AVAILABILITY** Available only with FONTGDOS, FSM, or SpeedoGDOS.

**PARAMETERS** handle is a valid workstation handle.

**BINDING**

```c
contrl[0] = 11;
contrl[1] = 0;
contrl[3] = 0;
contrl[5] = 13;
```
v_bez_on() – 7.29

```c
contrl[6] = handle;

vdi();
```

**COMMENTS**
This function should be called to free any memory reserved by the bezier functions.

**SEE ALSO**
`v_bez_on()`

---

**v_bez_on()**

WORD `v_bez_on( handle )`

WORD `handle`;

`v_bez_on()` enables bezier capabilities.

**OPCODE**
11

**SUB-OPCODE**
13

**AVAILABILITY**
Available only with **FONTGDOS, FSM, or SpeedoGDOS**.

**PARAMETERS**
`handle` is a valid workstation handle.

**BINDING**
```c
contrl[0] = 11;
contrl[1] = 1;
contrl[3] = 0;
contrl[5] = 13;
contrl[6] = handle;

vdi();

return intout[0];
```

**RETURN VALUE**
`v_bez_on()` returns a **WORD** value indicating the number of line segments each curve is composed of (smoothness). The value returned (0-7) is a power of 2 meaning that a return value of 7 indicates 128 line segments per curve.

**SEE ALSO**
`v_bez_off()`
v_bez_qual()

VOID v_bez_qual( handle, percent, actual )
WORD handle, percent;
WORD *actual;

v_bez_qual() sets the speed/quality ratio of the bezier curve rendering engine.

OPCODE

5

SUB-OPCODE

99

AVAILABILITY

Available only with FONTGDOS, FSM, or SpeedoGDOS.

PARAMETERS

handle specifies a valid workstation handle. percent is a value (0–100) specifying the tradeoff between bezier quality and speed. A value of 0 renders a bezier fastest with the lowest quality while a value of 100 renders a bezier slowest with the highest possible quality. On return, the WORD pointed to by actual will contain the actual value used.

BINDING

contrl[0] = 5;
contrl[1] = 0;
contrl[3] = 3;
contrl[5] = 99;
contrl[6] = handle;

intin[0] = 32;
intin[1] = 1;
intin[2] = percent;

vdi();

*actual = intout[0];

COMMENTS

actual may not be an exact percentage as the rendering engine may not actually support every value possible between 1–99.

SEE ALSO

v_bez(), v_bez_fill(), v_bez_on()
v_bit_image()

VOID v_bit_image( handle, fname, ratio, xscale, yscale, halign, valign, pxy )
WORD handle;
char *fname;
WORD aspect, xscale, yscale, halign, valign;
WORD *pxy;

v_bit_image() outputs a disk-based GEM `.IMG' file.

OPCODE 5
SUB-OPCODE 23
AVAILABILITY Supported by all printer, metafile, and memory drivers.

PARAMETERS handle is a valid workstation handle. fname specifies the GEMDOS file
specification for the GEM bit-image file to print. ratio should be 0 to ignore the
aspect ratio of the image and 1 to adhere to it.

xscale and yscale specify the method of scaling to apply to the image. Fractional
scaling is specified by a value of 0 whereas a value of 1 represents integer
scaling.

If fractional scaling is used, the image will be displayed at the coordinates given
by the VDI format rectangle pointed to by pxy. If integer scaling is applied, the
image will be displayed as large as possible within the given coordinates using
halign and valign to specify the image justification as follows:

<table>
<thead>
<tr>
<th>Value</th>
<th>halign</th>
<th>valign</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>Left</td>
<td>Top</td>
</tr>
<tr>
<td></td>
<td>IMAGE_LEFT</td>
<td>IMAGE_TOP</td>
</tr>
<tr>
<td>1</td>
<td>Center</td>
<td>Center</td>
</tr>
<tr>
<td></td>
<td>IMAGE_CENTER</td>
<td>IMAGE_CENTER</td>
</tr>
<tr>
<td>2</td>
<td>Right</td>
<td>Bottom</td>
</tr>
<tr>
<td></td>
<td>IMAGE_RIGHT</td>
<td>IMAGE_BOTTOM</td>
</tr>
</tbody>
</table>

BINDING

WORD tmp = 5;
intin[0] = ratio;
intin[1] = xscale;
intin[2] = yscale;
intin[3] = halign;
intin[4] = valign;
while(intin[tmp++] = (WORD)*fname++);

contrl[0] = 5;
contrl[1] = 2;
contrl[3] = --tmp;
contrl[5] = 23;
contrl[6] = handle;

ptsin[0] = pxy[0];
ptsin[1] = pxy[1];
ptsin[2] = pxy[2];
ptsin[3] = pxy[3];

vd़();

COMMENTS
A flag indicating whether the device supports scaling can be found in vq_extnd(). This call used with the memory driver can provide image scaling for transfer to the screen with vrt_cpyfm().

SEE ALSO vq_scan()

v_cellarray()

VOID v_cellarray( handle, pxy, rowlen, elements, num_rows, wrmode, colarray )
WORD handle;
WORD *pxy;
WORD rowlen, elements, num_rows, wrmode;
WORD *colarray;

v_cellarray() outputs an array of colored cells.

OPCODE 10

AVAILABILITY Not supported by any current drivers.

PARAMETERS handle specifies a valid workstation handle. pxy points to a WORD array with 4 entries specifying a VDI format rectangle giving the extent of the array to output.

rowlen specifies the length of each color array row. elements specifies the total number of color array elements. num_rows specifies the number of rows in the color array. wrmode specifies a valid writing mode (1–4) and colarray points to an array of WORDs (num_rows * elements) long.

BINDING

WORD i;
contrl[0] = 10;
contrl[1] = 2;
contrl[3] = num_rows * elements;
contrl[6] = handle;
contrl[7] = rowlen;
contrl[8] = elements;
contrl[9] = num_rows;
contrl[10] = wrt_mode;

for(i = 0;i < (num_rows * elements);i++)
    intin[i] = colarray;

ptsin[0] = pxy[0];
ptsin[1] = pxy[1];
ptsin[2] = pxy[2];
ptsin[3] = pxy[3];

vdi();

C A V E A T S

This function is not guaranteed available in any driver and should therefore be avoided unless you are sure the driver you are utilizing understands it.

S E E  A L S O  vq_cellarray()

v_circle()

VOID v_circle( handle, x, y, radius )
WORD handle, x, y, radius;

v_circle() outputs a filled circle.

O P C O D E  11

S U B - O P C O D E  4

A V A I L A B I L I T Y  Supported by all drivers. This function composes one of the 10 VDI GDP’s (Generalized Drawing Primitives). Although all current drivers support all GDP’s, their availability is not guaranteed and may vary. To check for a particular GDP refer to the table returned by v_opnvwk() or v_opnwk().

P A R A M E T E R S  handle specifies a valid workstation. x and y specify the center of a circle with a radius of radius.

B I N D I N G  contrl[0] = 11;
contrl[1] = 3;
contrl[3] = 0;
contrl[5] = 4;
contrl[6] = handle;

ptsin[0] = x;
ptsin[1] = y;

vdi();

S E E  A L S O  vsf_color(), vsf_interior(), vsf_style(), vsf_udpat()
v_clear_disp_list()

VOID v_clear_disp_list( handle )
WORD handle;

v_clear_disp_list() clears the display list of a workstation.

_OPCODE 5

_SUB-_OPCODE 22

_AVAILABILITY Supported by printer, plotter, metafile, and camera drivers.

_PARAMETERS handle specifies a valid workstation handle.

_BINDING contrl[0] = 5;
contrl[5] = 22;
contrl[6] = handle;

vdli();

 COMMENTS v_clear_disp_list() is essentially the same as v_clrwk() except that no form feed is issued.

SEE ALSO v_clrwk()

v_clrwk()

VOID v_clrwk( handle )
WORD handle;

v_clrwk() clears a physical workstation.

_OPCODE 3

_AVAILABILITY Supported by all drivers.

_PARAMETERS handle specifies a valid workstation.

_BINDING contrl[0] = 3;
contrl[6] = handle;
v_clsvwk()

VOID v_clsvwk( handle )
WORD handle;

v_clsvwk() closes a virtual workstation.

OPCODE 101

AVAILABILITY Supported by all drivers.

PARAMETERS handle specifies a valid virtual workstation to close.

BINDING
contl[0] = 101;
contl[6] = handle;

vdi();

SEE ALSO v_opnvwk()

v_clswk()

VOID v_clswk( handle )
WORD handle;

v_clswk() closes a physical workstation.

OPCODE 2

AVAILABILITY Available only with some form of GDOS.

THE ATARI COMPENDIUM
**v_contourfill()**

VOID v_contourfill( handle, x, y, color )

WORD handle, x, y, color;

_v_countourfill() outputs a ‘seed’ fill.

**OPCODE**

103

**AVAILABILITY**

Supported by all *current* screen, printer and metafile drivers. The availability of this call can be checked for using _vq_extnd()_.

**PARAMETERS**

(handle) specifies a valid workstation handle. x and y specify the starting point for the fill. If *color* is OTHER_COLOR (-1) then the fill continues in all directions until a color other than that found in x and y is found. If *color* is positive then the fill continues in all directions until color *color* is found.

**BINDING**

contrl[0] = 103;
contrl[6] = handle;

intin[0] = color;
ptsin[0] = x;
ptsin[1] = y;

_vdi();

**COMMENTS**

In true-color mode if a positive value for *color* is used, the fill spreads until a pixel is found with the same color as ‘virtual pen’ *color*.

**SEE ALSO**

_vsf_color(), vsf_interior(), vsf_style(), vsf_udpat()
v_curdown()

VOID v_curdown( handle )
WORD handle;

v_curdown() moves the text cursor down one line.

OPCODE 5

SUB-OPCODE 5

AVAILABILITY Supported by all screen drivers.

PARAMETERS handle specifies a valid workstation handle.

BINDING contrl[0] = 5;
contrl[5] = 5;
contrl[6] = handle;

vdi();

COMMENTS This call is equivalent to the ESC-B VT-52 code.

SEE ALSO v_curup()

v_curhome()

VOID v_curdown( handle )
WORD handle;

v_curhome() moves the text cursor to the upper-left of the screen.

OPCODE 5

SUB-OPCODE 8

AVAILABILITY Supported by all screen drivers.

PARAMETERS handle specifies a valid workstation handle.

BINDING contrl[0] = 5;
contrl[5] = 8;
contrl[6] = handle;
vdii();

**COMMENTS**
This call is equivalent to the ESC-H VT-52 code.

---

**v_curleft()**

VOID v_curleft( handle )
WORD handle;

v_curleft() moves the text cursor left one character position.

**OPCODE**
5

**SUB-OPCODE**
7

**AVAILABILITY**
Supported by all screen drivers.

**PARAMETERS**
handle is a valid workstation handle.

**BINDING**
contrl[0] = 5;
contrl[5] = 7;
contrl[6] = handle;
vdii();

**COMMENTS**
This call is equivalent to the ESC-D VT-52 code.

**SEE ALSO**
v_curright()
**v_curtext()**

VOID v_curtext( handle, str )
WORD handle;
char *str;

-v_curtext() outputs a line of text to the screen in text mode.

**OPCODE**
5

**SUB-OPCODE**
12

**AVAILABILITY**
Supported by all screen drivers.

**PARAMETERS**
handle is a valid workstation handle. str is a character pointer to a string no more than 127 characters long.

**BINDING**
WORD i = 0;
while(intin[i++] = (WORD)*str++);

intin[1] = 0;
contrl[0] = 5;
contrl[1] = 0;
contrl[3] = --i;
contrl[5] = 12;
contrl[6] = handle;

vdri();

**COMMENTS**
The line of text must not exceed the maximum length of the intin array as returned by vq_extnd() or the maximum length of your compilers’ array.

**SEE ALSO**
vs_curaddress(), v_rvon(), v_rvoff()
v_curup()

VOID v_curup( handle )
WORD handle;

v_curup() moves the text cursor up one line.

_OPCODE 5
_SUB_OPCODE 4

_AVAILABILITY Supported by all screen drivers.

_PARAMETERS handle specifies a valid workstation handle.

_BINDING contl[0] = 5;
contl[5] = 4;
contl[6] = handle;

vdii();

_COMMENTS This call is equivalent to the ESC-A VT-52 code.

_SEE_ALSO v_curdown()

v_dspcur()

VOID v_dspcur( handle, x, y )
WORD handle, x, y;

v_dspcur() displays the mouse pointer on screen at the specified position.

_OPCODE 5
_SUB_OPCODE 18

_AVAILABILITY Supported by all screen drivers.

_PARAMETERS handle specifies a valid workstation handle. x and y specify the screen coordinates of where to display the mouse pointer.
**v_eeol()**

VOID v_eeol( handle )

WORD handle;

v_eeol() erases the text line from the current cursor position rightwards.

**OPCODE**
5

**SUB-OPCODE**
10

**AVAILABILITY**
Supported by all screen drivers.

**PARAMETERS**
handle specifies a valid workstation handle.

**BINDING**

```c
contrl[0] = 5;
contrl[1] = 1
contrl[3] = 0;
contrl[5] = 18;
contrl[6] = handle;
ptsin[0] = x;
ptsin[1] = y;

vdi();
```

**COMMENTS**
This call is equivalent to the ESC-K VT-52 code.

**SEE ALSO**
v_eeos()
**v_eeos()**

WORD v_eeos( handle )
WORD handle;

v_eeos() erases the current screen of text from the cursor position.

**OPCODE** 5

**SUB-OPCODE** 9

**AVAILABILITY** Supported by all screen drivers.

**PARAMETERS** handle specifies a valid workstation handle.

**BINDING**

```c
contrl[0] = 5;
contrl[5] = 9;
contrl[6] = handle;

vdi();
```

**COMMENTS** This call is equivalent to the ESC-J VT-52 code.

**SEE ALSO** v_eeol()

---

**v_ellarc()**

VOID v_ellarc( handle, x, y, xradius, yradius, startangle, endangle )
WORD handle, x, y, xradius, yradius, startangle, endangle;

v_ellarc() outputs an elliptical arc segment.

**OPCODE** 11

**SUB-OPCODE** 6

**AVAILABILITY** Supported by all drivers. This function composes one of the 10 VDI GDP’s (Generalized Drawing Primitives). Although all current drivers support all GDP’s, their availability is not guaranteed and may vary. To check for a particular GDP refer to the table returned by v_opnvwk() or v_opnwk().

**PARAMETERS** handle specifies a valid workstation handle. x and y specify the coordinates of the
center of an arc with an X radius of \textit{xradius} and a Y radius of \textit{yradius}. Only the portion of the arc which falls between the angles specified in \textit{startangle} and \textit{endangle} will be drawn. Angles are specified in tenths of degrees as follows:

\begin{figure}[h]
\centering
\includegraphics[width=0.5\textwidth]{ellipse_diagram.png}
\end{figure}

**BINDING**

\begin{verbatim}
contrl[0] = 11;
contrl[5] = 6;
contrl[6] = handle;

intin[0] = startangle;
intin[1] = endangle;

ptsin[0] = x;
ptsin[1] = y;
ptsin[2] = xradius;
ptsin[3] = yradius;

vdi();
\end{verbatim}

**SEE ALSO**

\textit{v_ellipse()}, \textit{v_ellpie()}, \textit{vsl_color()}, \textit{vsl_type()}, \textit{vsl_width()}, \textit{vsl_udsty()}

---

**\textbf{v_ellipse()}**

\texttt{VOID v_ellipse( handle, x, y, xradius, yradius)}

\texttt{WORD handle, x, y, xradius, yradius;}

\textit{v_ellipse()} outputs a filled ellipse.

**OPCODE**

11

**SUB-OPCODE**

5

**AVAILABILITY**

Supported by all drivers. This function composes one of the 10 \textit{VDI} GDP’s (Generalized Drawing Primitives). Although all current drivers support all GDP’s, their availability is not guaranteed and may vary. To check for a particular GDP refer to the table returned by \textit{v_opnvwk()} or \textit{v_opnwk()}.

**PARAMETERS**

\textit{handle} specifies a valid workstation handle. \textit{x} and \textit{y} specify the center point of an arc with an X radius of \textit{xradius} and a Y radius of \textit{yradius}.
**BINDING**

\[
\text{ctrl}[0] = 11; \\
\text{ctrl}[1] = 2; \\
\text{ctrl}[3] = 0; \\
\text{ctrl}[5] = 5; \\
\text{ctrl}[6] = \text{handle}; \\
\text{ptsin}[0] = x; \\
\text{ptsin}[1] = y; \\
\text{ptsin}[2] = xradius; \\
\text{ptsin}[3] = yradius; \\
\text{vdi}();
\]

**SEE ALSO**

v_ellpie(), v_ellarc(), vsf_color(), vsf_interior(), vsf_style(), vsf_udpat(), vs_perimeter()

---

**v_ellpie()**

VOID v_ellpie( handle, x, y, xradius, yradius, startangle, endangle)

WORD handle, x, y, xradius, yradius, startangle, endangle;

\[\text{v_ellpie()}\] outputs a filled elliptical pie segment.

**OPCODE**

11

**SUB-OPCODE**

7

**AVAILABILITY**

Supported by all drivers. This function composes one of the 10 **VDI GDP**’s (Generalized Drawing Primitives). Although all current drivers support all GDP’s, their availability is not guaranteed and may vary. To check for a particular GDP refer to the table returned by v_opnvwk() or v_opnwk().

**PARAMETERS**

*handle* specifies a valid workstation handle. *x* and *y* specify the center coordinates of an elliptical pie segment to draw with an X radius of *xradius* and a Y radius of *yradius*. Only the portion of the arc will be drawn falling between the angles specified in *startangle* and *endangle* (as shown below). The ends of this arc is connected to the center point with lines forming the pie segment.
v_enter_cur()

VOID v_enter_cur( handle )
WORD handle;

v_enter_cur() clears the screen to color 0, removes the mouse cursor and enters
text mode.

OPCODE 5

SUB-OPCODE 3

AVAILABILITY Supported by all screen drivers.

PARAMETERS handle specifies a valid workstation handle.

BINDING

contrl[0] = 5;
contrl[5] = 3;
contrl[6] = handle;

vdi();

CAVEATS You should check that the left mouse button has been released with vq_mouse()
prior to calling this function. If the button is depressed when you call this function
the VDI will lock waiting for it to be released after v_exit_cur().

COMMENTS This call is used by a GEM application to prepare for executing a TOS
application when not running under MultiTOS.
v_exit_cur()

VOID v_exit_cur( handle )
WORD handle;

v_exit_cur() exits text mode and restores the mouse pointer.

OPCODE 5
SUB-OPCODE 2
AVAILABILITY Supported by all screen drivers.
PARAMETERS handle specifies a valid workstation handle.
BINDING contrl[0] = 5;
contrl[5] = 2;
contrl[6] = handle;
vd1();
CAVEATS See v_enter_cur().
COMMENTS To completely restore the screen you should call form_dial(FMD_FINISH, sx, sy, sw, sh) where sx, sy, sw, and sh are the coordinates of the screen.
SEE ALSO v_enter_cur()

v_fillarea()

VOID v_fillarea( handle, count, pxy )
WORD handle, count;
WORD *pxy;

v_fillarea() outputs a filled polygon.

OPCODE 9
AVAILABILITY Supported by all drivers.
PARAMETERS

handle specifies a valid workstation handle. count specifies the number of vertices in the polygon to output. pxy should point to an array of coordinate pairs with the first WORD being the first X point, the second WORD being the first Y point and so on.

BINDING

WORD i;

contrl[0] = 9;
contrl[1] = count;
contrl[3] = 0;
contrl[6] = handle;

for(i = 0; i < count*2; i++)
    ptsin[i] = pxy[i];

vdi();

COMMENTS

This function will automatically connect the first point with the last point.

SEE ALSO

v_pline(), v_contourfill()
v_fontinit()

VOID v_fontinit( fptr_high, fptr_low )
WORD fptr_high, fptr_low;

v_fontinit() allows replacement of the built-in system font.

OPCODE 5
SUB-OPCODE 102
AVAILABILITY All TOS versions.
PARAMETERS fptr_high and fptr_low are the high and low WORDs of a pointer to a Line-A compatible font header structure in Motorola (Big-Endian) format which contains information about the font to be used as a replacement for the system font.
BINDING
contr1[0] = 5;
contr1[1] = 0;
contr1[3] = 2;
contr1[5] = 102;
contr1[6] = handle;
intin[0] = fptr_high;
intin[1] = fptr_low;
vd1();
COMMENTS This function has never been officially documented though it exists in all current versions of TOS.

v_form_adv()

VOID v_form_adv( handle )
WORD handle;

v_form_adv() outputs the current page without clearing the display list.

OPCODE 5
SUB-OPCODE 20
AVAILABILITY Supported by all drivers.
PARAMETERS handle specifies a valid workstation handle.
**v_ftext()**

VOID v_ftext(handle, x, y, str)

WORD handle, x, y;
char *str;

**v_ftext()** outputs outline text taking spacing remainders into consideration.

**OPCODE**

241

**AVAILABILITY**

Available only with FSMGDOS or SpeedoGDOS.

**PARAMETERS**

*handle* specifies a valid workstation handle. *x* and *y* specify the starting coordinate of the **NULL**-terminated text string (see **vst_alignment()**) pointed to by *str* to print.

**BINDING**

```c
WORD i = 0;
while(intin[i++] = (WORD)*str++);

contrl[0] = 241;
contrl[1] = 1;
contrl[3] = --i;
contrl[6] = handle;

ptsin[0] = x;
ptsin[1] = y;

vdi();
```

**COMMENTS**

The text contained in *str* (including its **NULL** byte) should not exceed the maximum allowable size of the *intin* array (as indicated in the *work_out* array) or the size of the *intin* array allocated by your compiler.

To output 16-bit Speedo character indexes, use **v_ftext16()**.
This function produces output more properly spaced than with `v_gtext()` because it takes the remainder amounts from `vqt_f_extent()` into consideration.

**See Also**

`v_ftext()`, `v_ftext_offset()`, `v_ftext_offset16()`, `v_gtext()`, `vst_alignment()`, `vst_color()`, `vst_effects()`, `vst_arbpt()`, `vst_height()`, `vst_font()`, `vqt_f_extent()`, `vst_point()`

---

### v_ftext16()

**VOID v_ftext16( handle, x, y, wstr, wstrlen)**

**WORD handle, x, y;**

**WORD *wstr;**

**WORD wstrlen;**

`v_ftext16()` is a variant binding of `v_ftext()` that outputs 16-bit Speedo character text rather than 8-bit ASCII text.

**_OPCODE**

241

**AVAILABILITY**

Available only with **SpeedoGDOS**.

**PARAMETERS**

`handle` specifies a valid workstation handle. `x` and `y` specify the starting coordinate of the location to output text. `wstr` points to a NULL-terminated text string composed of **WORD**-sized Speedo characters. `wstrlen` specifies the length of the text string.

**BINDING**

```c
WORD i;
for( i = 0; i < wstrlen; i++)
    intin[i] = wstr[i];

ctrl[0] = 241;
ctrl[1] = 1;
ctrl[3] = wstrlen;
ctrl[6] = handle;

ptsin[0] = x;
ptsin[1] = y;

vdi();
```

**COMMENTS**

This function should only be used when `vst_charmap()` has been used to indicate that **WORD**-sized Speedo character indexes should be recognized rather than 8-bit ASCII.

The text contained in `wstr` (including its NULL byte) should not exceed the maximum allowable size of the `intin` array (as indicated in the `work_out` array) or
the size of the \texttt{intin} array allocated by your compiler.

\textbf{Caveats} \hfill \textbf{Ingredients}

Current versions of \texttt{SpeedoGDOS} become confused when the space character (index 0) is encountered in the string. It is suggested that one of the three space characters (of varying widths) at indexes 560-562 be used instead.

\textbf{See Also} \hfill \textbf{Ingredients}

\texttt{v\_ftext()}, \texttt{v\_ftext\_offset()}, \texttt{v\_ftext\_offset16()}, \texttt{v\_gtext()}, \texttt{vst\_alignment()}, \texttt{vst\_color()}, \texttt{vst\_effects()}, \texttt{vst\_arbp()}, \texttt{vst\_height()}, \texttt{vst\_font()}, \texttt{vqt\_f\_extent()}, \texttt{vst\_point()}

\textbf{v\_ftext\_offset()}

\begin{verbatim}
VOID v_ftext_offset(handle, x, y, str, offset)
WORD handle, x, y;
char *str;
WORD *offset;
\end{verbatim}

\texttt{v\_ftext\_offset()} is a variant binding of \texttt{v\_ftext()} available under \texttt{SpeedoGDOS} which allows an offset vector for each character to be specified.

\textbf{Opcode} \hfill \textbf{Ingredients}

241

\textbf{Availability} \hfill \textbf{Ingredients}

Available only with \texttt{SpeedoGDOS}.

\textbf{Parameters} \hfill \textbf{Ingredients}

\texttt{handle} specifies a valid workstation handle. \texttt{x} and \texttt{y} give the point where the string will be rendered. \texttt{offset} points to an array of \texttt{WORDS} which contains one \texttt{x} and \texttt{y} offset value for each character in \texttt{str}.

\textbf{Binding} \hfill \textbf{Ingredients}

\begin{verbatim}
WORD i = 0;
while(intin[i++] = (WORD)*str++); --i;

ptsin[0] = x;
ptsin[1] = y;

for(j = 0; j < i * 2; j++)
ptsin[j + 2] = offset[j];

contrl[0] = 241;
contrl[1] = i + 1;
contrl[3] = i;
contrl[6] = handle;

vdi();
\end{verbatim}

\textbf{Comments} \hfill \textbf{Ingredients}

The text contained in \texttt{str} (including its \texttt{NULL} byte) should not exceed the maximum allowable size of the \texttt{intin} array (as indicated in the \texttt{work\_out} array) or
the size of the intin array allocated by your compiler.

To output 16-bit Speedo character indexes, use \texttt{v_ftext_offset16()}.

\textbf{SEE ALSO} \texttt{v_ftext_offset16()}, \texttt{v_ftext()}, \texttt{v_gtext()}

\section*{\texttt{v_ftext_offset16()}}

\begin{verbatim}
VOID v_ftext_offset( handle, x, y, wstr, wstrlen, offset )
WORD handle, x, y;
WORD *wstr;
WORD wstrlen;
WORD *offset;
\end{verbatim}

\texttt{v_ftext_offset16()} is a variant binding of \texttt{v_ftext_offset()} which allows 16-bit Speedo character strings to be output rather than 8-bit ASCII codes.

\section*{OPCODE}

241

\section*{AVAILABILITY}

Available only with SpeedoGDOS.

\section*{PARAMETERS}

\texttt{handle} specifies a valid workstation handle. \texttt{x} and \texttt{y} give the point where the string will be rendered. \texttt{offset} points to an array of \texttt{WORD}s which contains one \texttt{x} and \texttt{y} offset value for each character in \texttt{wstr}.

\section*{BINDING}

\begin{verbatim}
WORD i;
for( i = 0; i < wstrlen; i++)
    intin[i] = wstr[i];
ptsin[0] = x;
ptsin[1] = y;
for( j = 0; j < i * 2; j++)
    ptsin[j + 2] = offset[j];
contrl[0] = 241;
contrl[1] = wstrlen + 1;
contrl[3] = wstrlen;
contrl[6] = handle;
vdii();
\end{verbatim}

\section*{COMMENTS}

This function should only be used when \texttt{vst_charmap()} has been used to indicate that \texttt{WORD} sized Speedo character indexes should be recognized rather than 8-bit ASCII.

The text contained in \texttt{wstr} (including its NULL byte) should not exceed the
maximum allowable size of the intin array (as indicated in the work_out array) or the size of the intin array allocated by your compiler.

**CAVEATS**

Current versions of SpeedoGDOS become confused when the space character (index 0) is encountered in the string. It is suggested that one of the three space characters (of varying widths) at indexes 560-562 be used instead.

**SEE ALSO**

v_ftext16(), v_ftext_offset()

---

**v_getbitmap_info()**

VOID v_getbitmap_info( handle, ch, advx, advy, xoff, yoff, width, height, bitmap)

WORD handle, ch;
fix31 *advx, *advy, *xoff, *yoff;
WORD *width, *height;
VOID *bitmap;

v_getbitmap_info() returns placement information for the bitmap of a character based on the current character font, size, and alignment.

**OPCODE**

239

**AVAILABILITY**

Available only with SpeedoGDOS.

**PARAMETERS**

handle specifies a valid workstation handle. ch is the character to return information about.

The fix31 variables pointed to by advx, advy, xoff, and yoff will be filled in with the x and y advance and offset vectors respectively. The WORDs pointed to by width and height will be filled in with the width and height of the bitmap pointed to by the value returned in bitmap.

**BINDING**

```c
contr1[0] = 239;
contr1[1] = 0;
contr1[3] = 1;
contr1[6] = handle;
intin[0] = ch;

vd1();

*width = intout[0];
*height = intout[1];
*advx = *(fix31 *)&intout[2];
```

1This call did exist in FSMGDOS, however the call had a completely different calling format. Atari changed the existing call as no FSMGDOS program had yet been written to utilize it.
The advance vector represents the amount to add to the current point to properly place the character. The offset vector, when added to the current point, gives the location of the upper-left corner of the bitmap.

**v_getoutline()**

```c
VOID v_getoutline(
    handle, ch, xyarray, bezarray, maxverts, numverts
)
```

**OPCODE**

243

**AVAILABILITY**

Available only with SpeedoGDOS2.

**PARAMETERS**

- `handle` specifies a valid workstation handle.
- `ch` specifies the character to return information about. The arrays pointed to by `xyarray` and `bezarray` are filled in with the bezier information for the character. The definition of `xyarray` and `bezarray` is given in the binding for `v_bez()`.

- `maxverts` should indicate the maximum number of vertices the buffer can hold. The `WORD` pointed to by `numverts` will be filled in with the actual number of vertices for the character.

**BINDING**

```c
contrl[0] = 243;
contrl[1] = 0;
contrl[3] = 6;
contrl[6] = handle;

intin[0] = ch;
intin[1] = maxverts;
*(WORD *)&intin[2] = xyarray;
*(WORD *)&intin[4] = bezarray;
```

#2 This call was present under FSMGDOS, however it’s binding has dramatically changed. Applications using this binding will not operate under the older FSMGDOS.
*numverts = intout[0];

v_get_pixel()

VOID v_get_pixel( handle, x, y, pindex, vindex )
WORD handle, x, y;
WORD *pindex, *vindex;

v_get_pixel() returns the color value for a specified coordinate on the screen.

OPCODE 105

AVAILABILITY Supported by all screen drivers.

PARAMETERS handle specifies a valid workstation handle. x any y specify the coordinate to return color information for.

In a palette-based mode the WORD pointed to by pindex will contain the hardware register index of the color and the WORD pointer to by vindex will contain the VDI index of the color.

In 16-bit true-color modes, pindex will be 0 and vindex will return the 16-bit RGB pixel value in the format {RRRR RGGG GGGB BBBB}.

In 32-bit color modes, the lower byte of vindex will contain the 8 bits of red data, the upper byte of pindex will contain the 8 bits of green data, and the lower byte of pindex will contain the 8 bits of blue data. The upper byte of vindex is reserved for non-color data.

BINDING

contrl[0] = 105;
contrl[1] = 1;
contrl[3] = 0;
contrl[6] = handle;

ptsin[0] = x;
ptsin[1] = y;

vdic();

*pindex = intout[0];
*vindex = intout[1];
v_gtext()

VOID v_gtext( handle, x, y, str)

WORD handle, x, y;
char *str;

v_gtext() outputs graphic text.

**OPCODE**
8

**AVAILABILITY**
Supported by all drivers.

**PARAMETERS**
- *handle* specifies a valid workstation handle. *x* and *y* specify the starting coordinates of the text (see `vst_alignment()`).
- *str* is a pointer to a NULL-terminated character string to print.

**BINDING**
WORD i = 0;

while(intin[i++] = (WORD)*str++);
contrl[0] = 8;
contrl[1] = 1;
contrl[3] = --i;
contrl[6] = handle;
ptsin[0] = x;
ptsin[1] = y;

vdi();

**COMMENTS**
The text contained in *str* (including its NULL byte) should not exceed the maximum allowable size of the intin array (as indicated in the work_out array) or the size of the intin array allocated by your compiler.

Using this function to output outline text with FSMGDOS is possible to remain backward-compatible but not recommended as it will introduce small errors as spacing remainders are lost.

**SEE ALSO**
v_ftext(), v_ftext_offset(), vst_color(), vst_effects(), vst_alignment(), vst_height(), vst_point()
v_hardcopy()

VOID v_hardcopy( handle )
WORD handle;

_v_hardcopy() invokes the ALT-HELP screen dump.

**Opcode**

5

**Sub-Opcode**

17

**Availability**

Supported by screen drivers running under ST compatible resolutions.

**Parameters**

*handle* specifies a valid workstation handle.

**Binding**

```c
contr1[0] = 5;
contr1[1] = contr1[3] = 0;
contr1[5] = 17;
contr1[6] = handle;

vdi();
```

**Caveats**

This function works in only ST compatible screen modes and should thus be avoided.

**See Also**

Scrdmp()

---

v_hide_c()

VOID v_hide_c( handle )
WORD handle;

_v_hide_c() hides the mouse cursor.

**Opcode**

123

**Availability**

Supported by all screen drivers.

**Parameters**

*handle* specifies a valid workstation handle.

**Binding**

```c
contr1[0] = 123;
contr1[1] = contr1[3] = 0;
contr1[6] = handle;

vdi();
```
**v_justified()**

VOID v_justified(handle, x, y, str, length, wflag, cflag)

WORD handle, x, y;
char *str;
WORD length, wflag, cflag;

v_justified() outputs justified graphics text.

**OPCODE**

11

**SUB-OPCODE**

10

**AVAILABILITY**

Supported by all drivers. This function composes one of the 10 VDI GDP’s (Generalized Drawing Primitives). Although all current drivers support all GDP’s, their availability is not guaranteed and may vary. To check for a particular GDP refer to the table returned by v_opnvwk() or v_opnwk().

**PARAMETERS**

handle specifies a valid workstation handle. x and y specify the starting coordinates at which to draw the NULL-terminated text string (see vst_alignment() ) pointed to by str. length specifies the pixel length of the area to justify on.

wflag and cflag specify the type of justification to perform between words and characters respectively. A value of NOJUSTIFY (0) indicates no justification whereas a value of JUSTIFY (1) indicates to perform justification.

**BINDING**

WORD i = 0;

while(intin[i++] = (WORD)*str++);

contrl[0] = 11;
contrl[1] = 2;
contrl[3] = --i;
contrl[5] = 10;
contrl[6] = handle;

intin[0] = wflag;
intin[1] = cflag;

ptsin[0] = x;
ptsin[1] = y;
ptsin[2] = length;
ptsin[3] = 0;

vdi();

**COMMENTS**
This call does not take into account remainder information from outline fonts.

**SEE ALSO**
v_gtext(), v_ftext(), vst_color(), vst_font(), vst_effects(), vst_alignment(), vst_point(), vst_height()

---

**v_killoutline()**

VOID v_killoutline( handle, outline)
WORD handle;
FSMOUTLINE outline;

v_killoutline() releases an outline from memory.

**OPCODE**
242

**AVAILABILITY**
Available only with FSMGDOS or SpeedoGDOS.

**COMMENTS**
Under FSMGDOS this call was required to release memory allocated for an outline returned from v_getoutline(). With SpeedoGDOS, this call is no longer required and is thus not documented further.

**SEE ALSO**
v_getoutline()

---

**v_loadcache()**

WORD v_loadcache( handle, fname, mode)
WORD handle;
char *fname;
WORD mode;

v_loadcache() loads a previously saved cache file from disk.

**OPCODE**
250

**AVAILABILITY**
Supported only by FSMGDOS and SpeedoGDOS.

**PARAMETERS**
handle specifies a valid workstation handle. fname specifies the GEMDOS file
specification of the cache file to load. \textit{mode} specifies whether current data will be flushed first. A value of 0 will append the loaded cache to the current cache whereas a value of 1 will flush the cache prior to loading.

\textbf{BINDING}

\begin{verbatim}
WORD i = 1;
intin[0] = mode;
while(intin[i++] = (WORD)*fname++);

contrl[0] = 250;
contrl[1] = 0;
contrl[3] = --i;
contrl[6] = handle;

vdi();
return intout[0];
\end{verbatim}

\textbf{RETURN VALUE}

\texttt{v_loadcache()} returns 0 if successful or -1 if an error occurred.

\textbf{COMMENTS}

This command only affects the cache responsible for storing bitmaps created from outline characters.

\textbf{SEE ALSO}

\texttt{v_savecache()}, \texttt{v_flushcache()}

---

\textbf{v_meta_extents()}

\texttt{VOID v_meta_extents( handle, xmin, ymin, xmax, ymax)\hfill}

\texttt{WORD handle, xmin, ymin, xmax, ymax;}

\texttt{v_meta_extents()} embeds placement information for a metafile.

\textbf{OPCODE}

5

\textbf{SUB-OPCODE}

98

\textbf{AVAILABILITY}

Supported by all metafile drivers.

\textbf{PARAMETERS}

\textit{handle} specifies a valid workstation handle. \textit{xmin} and \textit{ymin} specify the upper left corner of the bounding box of the metafile. \textit{xmax} and \textit{ymax} specify the lower left corner.

\textbf{BINDING}

\begin{verbatim}
contrl[0] = 5;
contrl[1] = 2;
contrl[3] = 0;
contrl[5] = 98;
contrl[6] = handle;
ptsin[0] = xmin;
\end{verbatim}
**v_opnvwk()**

VOID v_opnvwk(work_in, handle, work_out)

WORD *work_in, *handle, *work_out;

**v_opnvwk()** opens a virtual VDI workstation.

**OPCODE**

100

**AVAILABILITY**

Supported by all drivers.

**PARAMETERS**

*work_in* is a pointer to an array of 11 **WORDs** which define the initial defaults for the workstation as follows:

<table>
<thead>
<tr>
<th><em>work_in[x]</em></th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>Device identification number. This indicates the physical device ID of the device (the line number of the driver in ASSIGN.SYS when using GDOS). For screen devices you should normally use the value Getrez() + 2, however, a value of 1 is acceptable if not using any loaded fonts.</td>
</tr>
<tr>
<td>1</td>
<td>Default line type (same as vsl_type() ).</td>
</tr>
<tr>
<td>2</td>
<td>Default line color (same as vsl_color() ).</td>
</tr>
<tr>
<td>3</td>
<td>Default marker type (same as vsm_type() ).</td>
</tr>
<tr>
<td>4</td>
<td>Default marker color (same as vsm_color() ).</td>
</tr>
<tr>
<td>5</td>
<td>Default font (same as vst_font() ).</td>
</tr>
<tr>
<td>6</td>
<td>Default text color (same as vst_color() ).</td>
</tr>
<tr>
<td>7</td>
<td>Default fill interior.</td>
</tr>
<tr>
<td>8</td>
<td>Default fill style.</td>
</tr>
<tr>
<td>9</td>
<td>Default fill color.</td>
</tr>
</tbody>
</table>
7.62 – VDI/GDOS Function Reference

<table>
<thead>
<tr>
<th></th>
<th>Coordinate type flag. A value of 0 specifies NDC ‘Normalized Device Coordinates’ coordinates whereas a value of 2 specifies RC ‘Raster Coordinates’. All other values are reserved. NDC coordinates are only available when using external drivers with GDOS.</th>
</tr>
</thead>
<tbody>
<tr>
<td>10</td>
<td></td>
</tr>
</tbody>
</table>

handle should be set to the current handle (not the device ID) of the physical workstation for this device. For screen devices this is the value returned by graf_handle(). On exit handle will be filled in the VDI workstation handle allocated, if successful, or 0 if the workstation could not be opened.

work_out points to an array of 57 WORDs which on exit will be filled in by the VDI with information regarding the allocated workstation as follows (a structure name is listed beside its array member for those using the ‘C’ style VDI_Workstation structure instead of the array):

<table>
<thead>
<tr>
<th>work_out[x]</th>
<th>VDI Structure Member</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>xres</td>
<td>Width of device in pixels - 1.</td>
</tr>
<tr>
<td>1</td>
<td>yres</td>
<td>Height of device in pixels - 1.</td>
</tr>
</tbody>
</table>
| 2            | noscale              | Device coordinate units flag:  
|               |                      | 0 = Device capable of producing a precisely scaled image (screen, printer, etc...)  
<p>|               |                      | 1 = Device not capable of producing a precisely scaled image (film recorder, etc...) |
| 3            | wpixel               | Width of pixel in microns (1/25400 inch). |
| 4            | hpixel               | Height of pixel in microns (1/25400 inch). |
| 5            | cheights             | Number of character heights (0 = continuous scaling). |
| 6            | linetypes            | Number of line types. |
| 7            | linewidths           | Number of line widths (0 = continuous scaling). |
| 8            | markertypes          | Number of marker types. |
| 9            | markersizes          | Number of marker sizes (0 = continuous scaling). |
| 10           | faces                | Number of faces supported by the device. |
| 11           | patterns             | Number of available patterns. |
| 12           | hatches              | Number of available hatches. |
| 13           | colors               | Number of predefined colors/pens (ST High = 2, ST Medium = 4, TT Low = 256, True Color = 256). |
| 14           | ngdps                | Number of supported GDP’s |</p>
<table>
<thead>
<tr>
<th>15-24</th>
<th>cangbps[10]</th>
<th>cangbps[ 0 – (ngdps - 1)] contains a list of the GDP's the device supports as follows:</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>1 = Bar</td>
<td>2 = Arc</td>
</tr>
<tr>
<td></td>
<td>3 = Pie Slice</td>
<td>4 = Circle</td>
</tr>
<tr>
<td></td>
<td>5 = Ellipse</td>
<td>6 = Elliptical Arc</td>
</tr>
<tr>
<td></td>
<td>7 = Elliptical Pie</td>
<td>8 = Rounded Rectangle</td>
</tr>
<tr>
<td></td>
<td>9 = Filled Rounded Rectangle</td>
<td>10 = Justified Graphics Text</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>25-34</th>
<th>gdpattr[10]</th>
<th>For each GDP as listed above, gdpattr[ 0 – (ngdps - 1)] indicates the attributes which are applied to that GDP as follows:</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>1 = Polyline (vsl_...)</td>
<td>2 = Polymarker (vsm_...)</td>
</tr>
<tr>
<td></td>
<td>3 = Text (vst_...)</td>
<td>4 = Fill Area (vsf_...)</td>
</tr>
<tr>
<td></td>
<td>5 = None</td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>35</th>
<th>cancolor</th>
<th>Color capability flag.</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>0 = No</td>
<td>1 = Yes</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>36</th>
<th>cantextrot</th>
<th>Text rotation flag.</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>0 = No</td>
<td>1 = Yes</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>37</th>
<th>canfillarea</th>
<th>Fill area capability flag.</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>0 = No</td>
<td>1 = Yes</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>38</th>
<th>cancellarray</th>
<th>Cell array capability flag.</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>0 = No</td>
<td>1 = Yes</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>39</th>
<th>palette</th>
<th>Number of available colors in palette.</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>0 = &gt; 32767 colors</td>
<td>2 = Monochrome</td>
</tr>
<tr>
<td></td>
<td>&gt; 2 = Color</td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>40</th>
<th>locators</th>
<th>Number of locator devices.</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>1 = Keyboard only.</td>
<td>2 = Keyboard and other.</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>41</th>
<th>valuators</th>
<th>Number of valuator devices.</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>1 = Keyboard only.</td>
<td>2 = Keyboard and other.</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>42</th>
<th>choicedevs</th>
<th>Number of choice devices.</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>1 = Function keys.</td>
<td>2 = Function keys + keypad.</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>43</th>
<th>stringdevs</th>
<th>Number of string devices.</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>1 = Keyboard.</td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>44</th>
<th>wstype</th>
<th>Workstation type.</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>0 = Output only</td>
<td>1 = Input only</td>
</tr>
<tr>
<td></td>
<td>2 = Input/Output</td>
<td></td>
</tr>
<tr>
<td></td>
<td>3 = Metafile</td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>45</th>
<th>minwchar</th>
<th>Minimum character width in pixels.</th>
</tr>
</thead>
</table>

<table>
<thead>
<tr>
<th>46</th>
<th>minhchar</th>
<th>Minimum character height in pixels.</th>
</tr>
</thead>
</table>

<table>
<thead>
<tr>
<th>47</th>
<th>maxwchar</th>
<th>Maximum character width in pixels.</th>
</tr>
</thead>
<tbody>
<tr>
<td>48</td>
<td>maxhchar</td>
<td>Maximum character height in pixels.</td>
</tr>
<tr>
<td>49</td>
<td>minwline</td>
<td>Minimum line width.</td>
</tr>
<tr>
<td>50</td>
<td>zero5</td>
<td>Reserved (0).</td>
</tr>
<tr>
<td>51</td>
<td>maxwline</td>
<td>Maximum line width.</td>
</tr>
<tr>
<td>52</td>
<td>zero7</td>
<td>Reserved (0).</td>
</tr>
<tr>
<td>53</td>
<td>minwmark</td>
<td>Minimum marker width.</td>
</tr>
<tr>
<td>54</td>
<td>minhmark</td>
<td>Minimum marker height.</td>
</tr>
<tr>
<td>55</td>
<td>maxwmark</td>
<td>Maximum marker width.</td>
</tr>
<tr>
<td>56</td>
<td>maxhmark</td>
<td>Maximum marker height.</td>
</tr>
</tbody>
</table>

**BINDING**

```c
WORD i;

contrl[0] = 100;
contrl[1] = 0;
contrl[3] = 11;
contrl[6] = *handle;

for(i = 0; i < 11; i++)
    intin[i] = work_in[i];

vdi();

*handle = contrl[6];

for(i = 0; i < 45; i++)
    work_out[i] = intout[i];

for(i = 0; i < 13; i++)
    work_out[45+i] = intout[i];
```

**CAVEATS**

The VDI included with TOS versions less than 2.06 sometimes returned the same handle for consecutive calls using the same physical handle.

**COMMENTS**

Using multiple virtual workstations provides the benefit of being able to define multiple sets of default line types, text faces, etc... without having to constantly set them.

The VDI_Workstation structure method is the recommended method of using this function. See the VDI entry for V_Opnwk() and V_Opnvwk().

Desk accessories running under TOS versions below 1.4 should not leave a workstation open across any call which might surrender control to GEM (event_button(), event_multi(), etc...). This could give GEM time to change screen resolutions and TOS versions below 1.4 did not release memory allocated by a desk accessory (including workstations) when a resolution change occurred.

**SEE ALSO**

v_opnwk(), vq_extend(), v_clsvwk(), V_Opnvwk()
**V_Opnvwk()**

```c
WORD V_Opnvwk( dev )
VDI_Workstation dev;
```

**V_Opnvwk()** is not a component of the VDI, rather an interface binding designed to simplify working with virtual screen workstations. It will open a virtual screen workstation with a `VDI_Workstation` structure as a parameter rather than `work_in` and `work_out` arrays.

**Opcode**

N/A

**Availability**

User-defined.

**Parameters**

`ws` is a pointer to a `VDI_Workstation` structure defined as follows (for the meaning of each structure member, refer to `v_opnvwk()`):

```c
typedef struct
{
    WORD handle, dev_id;
    WORD wchar, hchar, wbox, hbox;
    WORD xres, yres;
    WORD noscale;
    WORD wpixel, hpixel;
    WORD cheights;
    WORD linetypes, linewidths;
    WORD markertypes, markersizes;
    WORD faces, patterns, hatches, colors;
    WORD ngdps;
    WORD cangdps[10];
    WORD gdattrs[10];
    WORD cancolor, cantextrot;
    WORD canfillarea, cancellarray;
    WORD palette;
    WORD locators, valuators;
    WORD choicedevs, stringdevs;
    WORD wstype;
    WORD minwchar, minhchar;
    WORD maxwchar, maxwchar;
    WORD minwline;
    WORD zero5;
    WORD maxwline;
    WORD zero7;
    WORD minwmark, minhmark;
    WORD maxwmark, maxhmark;
    WORD screentype;
    WORD bgcolors, textfx;
    WORD canscale;
    WORD planes, lut;
    WORD rops;
    WORD cancontourfill, textrot;
    WORD writemodes;
    WORD inputmodes;
} VDI_Workstation;
```
WORD textalign, inking, rubberbanding;
WORD maxvertices, maxintin;
WORD mousebuttons;
WORD widestyles, widemodes;
WORD reserved[38];
} VDI_Workstation;

BINDING

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

    in[0] = Getrez() + 2;
    dev->dev_id = in[0];
    for(i = 1; i < 10; 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)
        vq_extnd( i, 1, &dev->screentype );

    return (i);
}

RETURN VALUE
V_Opnvwk() returns 0 if non-successful or the workstation handle otherwise.

COMMENTS
This function definition is adapted from an article which appeared in the ‘Atari .RSC’ developers newsletter (Nov ‘90 - Jan ‘91).

SEE ALSO
v_opnvwk(), V_Opnwk(), vq_extnd()

v_opnwk()

VOID v_opnwk( work_in, handle, work_out )
WORD *work_in, *handle, *work_out;

v_opnwk() opens a physical workstation.

OPCODE
1

AVAILABILITY
Available only with some form of GDOS.

PARAMETERS
All parameters for this function are consistent with v_opnvwk() except as follows:

On entry, handle does not need to contain any specific value. On return, however,
it will contain a workstation handle if successful or 0 if the call failed.

**BINDING**

```c
WORD i;

contrl[0] = 1;
contrl[1] = 0;
contrl[3] = 11;

for(i = 0; i < 11; i++)
    intin[i] = work_in[i];

vdi();

*handle = contrl[6];

for(i = 0; i < 45; i++)
    work_out[i] = intout[i];

for(i = 0; i < 13; i++)
    work_out[45+i] = ptsout[i];
```

**COMMENTS**

Physical workstations should be opened when needed and closed immediately afterwards. For example, a word processor should not open the printer workstation when the application starts and close it when it ends. If this is done, the user will be unable to change printers with the Printer Setup CPX(s).

**SEE ALSO**

V_Opnwk(), v_opnvwk(), vq_extnd()

---

**V_Opnwk()**

```c
WORD V_Opnwk( devno, dev )
WORD devno;
VDI_Workstation dev;
```

**OPCODE**

N/A

**AVAILABILITY**

User-defined.

**PARAMETERS**

`devno` specifies the device ID of the device to open. Valid values for `devno` follow:

- 1-10 = Screen (loaded device drivers only)
- 11-20 = Plotters
- 21-30 = Printers
- 31-40 = Metafile Drivers
**ws** is a **VDI_Workstation** structure as defined in **V_Opnvwk()**.

**BINDING**

```c
WORD
V_Opnvwk( devno, dev )
WORD devno;
VDI_Workstation dev;
{
   WORD i, in[11];
   in[0] = dev->dev_id = devno;
   for(i = 1; i < 10; in[i++] = 1);
   in[10] = 2;
   i = devno;
   v_opnvwk( in, &i, &dev->xres );
   dev->handle = i;
   if(i)
      vq_extnd( i, 1, &dev->screentype );
   return (i);
}
```

**RETURN VALUE**

**V_Opnwk()** returns a workstation handle if successful or 0 if the call failed.

**COMMENTS**

This function definition is adapted from an article which appeared in the ‘Atari .RSC’ developers newsletter (Nov ‘90 - Jan ‘91).

**SEE ALSO**

**v_opnwk(), vq_extnd(), v_opnvwk(), V_Opnvwk()**

---

**v_output_window()**

```c
VOID v_output_window( handle, pxy )
WORD handle;
WORD *pxy;
```

**v_output_window()** outputs a specified portion of the current page.

**OPCODE**

5

**SUB-OPCODE**

22

**AVAILABILITY**

Supported by all printer and metafile drivers under any type of **GDOS**.

**PARAMETERS**

*handle* specifies a valid workstation handle, *pxy* is a pointer to an array of four
**v_pgcount()**

VOID v_pgcount( handle, numcopies)

**WORD** handle, numcopies;

*v_pgcount()* is used to cause the laser printer to output multiple copies of the current page.

**OPCODE** 5

**SUB-OPCODE** 2000

**AVAILABILITY** Supported only with some laser printer drivers (for instance the Atari laser printer driver) under some form of **GDOS**.

**PARAMETERS**

*handle* specifies a valid workstation handle. *numcopies* specifies the number of copies to print minus one. A value of 0 means print one copy, a value of 1, two copies, and so on.

**BINDING**

```
contrl[0] = 5;
contrl[1] = 0;
contrl[3] = 1;
contrl[6] = handle;
```

```
intin[0] = numcopies;
```
vdi();

**COMMENTS**  This call is preferred over repeatedly calling v_updwk() and v_form_adv() as this method forces the printer data to be resent for each page.

---

**v_pieslice()**

VOID v_pieslice( handle, x, y, radius, startangle, endangle )

WORD handle, x, y, radius, startangle, endangle;

v_pieslice() outputs a filled pie segment.

**OPCODE**  11

**SUB-OPCODE**  3

**AVAILABILITY**  Supported by all drivers. This function composes one of the 10 VDI GDP’s (Generalized Drawing Primitives). Although all current drivers support all GDP’s, their availability is not guaranteed and may vary. To check for a particular GDP refer to the table returned by v_opnvwk() or v_opnwk().

**PARAMETERS**  handle specifies a valid workstation handle. x and y specify the center of a circular segment of radius radius which is drawn between the angles of startangle and endangle (specified in tenths of degrees - legal values illustrated below) and connected to the center point.

```
<table>
<thead>
<tr>
<th>Start Angle</th>
<th>End Angle</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>90</td>
</tr>
<tr>
<td>90</td>
<td>180</td>
</tr>
<tr>
<td>180</td>
<td>270</td>
</tr>
<tr>
<td>270</td>
<td>360</td>
</tr>
</tbody>
</table>
```

**BINDING**

```c
contrl[0] = 11;
contrl[1] = 4;
contrl[3] = 2;
contrl[5] = 3;
contrl[6] = handle;
ptsin[0] = x;
ptsin[1] = y;
ptsin[6] = radius;
intin[0] = startangle;
```
int in[1] = endangle;

vdi();

SEE ALSO v_ellpie(), vsf_color(), vsf_style(), vsf_interior(), vsf_udpat(), vsf_perimeter()

---

v_pline()

VOID v_pline( handle, count, pxy )
WORD handle, count;
WORD *pxy;

v_pline() outputs a polyline (group of one or more lines).

OPCODE 6

AVAILABILITY Supported by all drivers.

PARAMETERS handle specifies a valid workstation handle. count specifies the number of
vertices in the line path (2 to plot a single line). pxy points to a WORD array with
count * 2 elements containing the vertices to plot as in (X1, Y1), (X2, Y2), etc...

BINDING WORD i;

contrl[0] = 6;
contrl[1] = count;
contrl[3] = 0;
contrl[6] = handle;

for(i = 0; i < (count*2); i++)
    ptsin[i] = count[i];

vdi();

COMMENTS To draw a single point with this function, pxy[2] should equal pxy[0], pxy[3]
should equal pxy[1], and count should be 2.

SEE ALSO v_fillarea(), vsl_color(), vsl_type(), vsl_udsty(), vsl_ends()
v_pmarker()

VOID v_pmarker( handle, count, pxy )
WORD handle, count;
WORD *pxy;

v_pmarker() outputs one or several markers.

OPCODE 7

AVAILABILITY Supported by all drivers.

PARAMETERS handle specifies a valid workstation. count specifies the number of markers to plot. pxy points to a WORD array with (count * 2) elements containing the vertices of the markers to plot as in (X1, Y1), (X2, Y2), etc...

BINDING WORD i;

contr1[0] = 7;
contr1[1] = count;
contr1[3] = 0;
contr1[6] = handle;

for(i = 0;i < (count * 2); i++)
    ptsin[i] = pxy[i];

vdi();

COMMENTS Single points may be plotted quickly with this function when the proper marker type is selected with vsm_type().

SEE ALSO vsm_type(), vsm_height(), vsm_color()

v_rbox()

VOID v_rbox( handle, pxy )
WORD handle;
WORD *pxy;

v_rbox() outputs a rounded box (not filled).

OPCODE 11

SUB-OPCODE 8
**v_rfbox()**

**VOID v_rfbox( handle, pxy )**

**WORD handle;**

**WORD *pxy;**

v_rfbox() outputs a filled rounded-rectangle.

**OPCODE**

11

**SUB-OPCODE**

9

**AVAILABILITY**

Supported by all drivers. This function composes one of the 10 VDI GDP’s (Generalized Drawing Primitives). Although all current drivers support all GDP’s, their availability is not guaranteed and may vary. To check for a particular GDP refer to the table returned by v_opnvwk() or v_opnwk().

**PARAMETERS**

*handle* specifies a valid workstation handle. *pxy* points to an array of 4 WORDs containing the VDI format rectangle of the rounded box to output.

**BINDING**

```c
contrl[0] = 11;
contrl[1] = 2;
contrl[3] = 0;
contrl[5] = 8;
contrl[6] = handle;

pxy[0] = ptsin[0];
pxy[1] = ptsin[1];
pxy[2] = ptsin[2];
pxy[3] = ptsin[3];

vdi();
```

**CAVEATS**

There is no way to define to size of the ‘roundness’ of the corners.

**SEE ALSO**

v_rfbox(), v_bar(), vsl_type(), vsl_color(), vsl_udsty(), vsl_ends()
contrl[1] = 2;
contrl[3] = 0;
contrl[5] = 9;
contrl[6] = handle;

ptsin[0] = pxy[0];
ptsin[1] = pxy[1];
ptsin[2] = pxy[2];
ptsin[3] = pxy[3];

vdi();

CAVEATS
There is no way to specify the 'roundness' of the rectangle.

SEE ALSO v_rbox(), v_bar(), vsf_color(), vsf_style(), vsf_interior(), vsf_udpat()

---

v_rmcur()

VOID v_rmcur( handle )
WORD handle;

v_rmcur() removes the last mouse cursor displayed.

OPCODE 5
SUB-OPCODE 19
AVAILABILITY Supported by all screen drivers.
PARAMETERS handle specifies a valid workstation handle.
BINDING contrl[0] = 5;
contrl[5] = 19;
contrl[6] = handle;

vdi();

COMMENTS v_rmcur() should only be used in conjunction with v_dspcur() when the mouse is moved manually. graf_mouse() or v_hide_c() should be used unless this is your intention.

SEE ALSO v_hide_c(), graf_mouse()
v_rvoff()

VOID v_rvoff( handle )
WORD handle;

v_rvoff() causes alpha screen text to be displayed in normal video (as opposed to inverse).

OPCODE  5
SUB-OPCODE  14
AVAILABILITY Supported by all screen drivers.
PARAMETERS handle specifies a valid workstation handle.
BINDING contrl[0] = 5;
        contrl[5] = 14;
        contrl[6] = handle;
        vdi();

COMMENTS This call is equivalent to the ESC-Q VT-52 code.
SEE ALSO v_rvon(), v_curtext()

v_rvon()

VOID v_rvon( handle )
WORD handle;

v_rvon() causes alpha screen text to be displayed in inverse mode.

OPCODE  5
SUB-OPCODE  13
AVAILABILITY Supported by all screen devices.
PARAMETERS handle specifies a valid workstation handle.
BINDING contrl[0] = 5;
contrl[5] = 13;
contrl[6] = handle;

vdi();

**COMMENTS**
This call is equivalent to the ESC-P VT-52 code.

**SEE ALSO**
v_rvoff(), v_curtext()

---

**v_savecache()**

```c
WORD v_savecache( handle, fname )
WORD handle;
char *fname;
```

**v_savecache()** saves the current outline cache.

**OPCODE**
249

**AVAILABILITY**
Available only with FSMGDOS or SpeedoGDOS.

**PARAMETERS**
*handle* specifies a valid workstation handle. *fname* specifies the GEMDOS file specification of the cache file to save.

**BINDING**
```c
WORD i = 0;
while(intin[i++] = (WORD)*fname++);
```

```c
contrl[0] = 249;
contrl[1] = 0;
contrl[3] = --i;
contrl[6] = handle;
```

```c
vdi();
```

```c
return intout[0];
```

**RETURN VALUE**
v_savecache() returns 0 if successful or -1 if an error occurred.

**COMMENTS**
This call only saves the portion of the cache responsible for storing bitmaps created from outlines.

**SEE ALSO**
v_loadcache(), v_flushcache()
**v_set_app_buff()**

VOID v_set_app_buff( but, nparagraphs )
VOID *buf;
WORD nparagraphs;

\[ \text{v_set_app_buff()} \] designates memory for use by the bezier generation routines.

**_OPCODE**

-1

**SUB_OPCODE**

6

**AVAILABILITY**

Available only with **FONTGDOS**, **FSMGDOS** or **SpeedoGDOS**.

**PARAMETERS**

*buf* specifies the address of a buffer which the bezier generator routines may safely use. *nparagraphs* specifies the size of the buffer in ‘paragraphs’ (16 bytes).

**BINDING**

\[
\begin{align*}
\text{ctrl}[0] &= -1; \\
\text{ctrl}[1] &= 0; \\
\text{ctrl}[3] &= 2; \\
\text{ctrl}[5] &= 6; \\
*(\text{VOID} *)\&in[0] &= \text{buf}; \\
in[2] &= \text{nparagraphs}; \\
\text{vdi}();
\end{align*}
\]

**COMMENTS**

Before the application exits, it should call \text{v_set_app_buff( NULL, 0 )} to ‘unmark’ memory. The application is then responsible for deallocating the memory.

In the absence of this call the first \text{v_bez()} or \text{v_bezfill()} call will allocate its own buffer of 8K. Atari documentation recommends a size of about 9K depending on the extents of the bezier you wish to generate.

**SEE ALSO**

\text{v_bez()}

---

**v_show_c()**

VOID v_show_c( handle, reset )
WORD handle, reset;

\[ \text{v_show_c()} \] ‘unhides’ the mouse cursor.

**_OPCODE**

122

---

\text{T he A t a r i C o m p e n d i u m}
Availabilty

Supported by all screen drivers.

Parameters

*handle* specifies a valid workstation handle. If *reset* is 0 the mouse will be displayed regardless of the number of times it was ‘hidden’. Otherwise, the call will only display the cursor if the function has been called an equal number of times compared to `v_hide_c()`.

Binding

```c
contr1[0] = 122;
contr1[1] = 0;
contr1[3] = 1;
contr1[6] = handle;
intin[0] = reset;
vdi();
```

Caveats

While it may be tempting to always use a *reset* value of 0, it is not recommended. Doing so may confuse the system so that when the critical error handler is called, the mouse is not displayed.

See Also

`v_hide_c()`, `graf_mouse()`

---

### `v_updwk()`

**VOID v_updwk( handle )**

**WORD handle;**

`v_updwk()` outputs the current page to the specified device.

**Opcode**

4

**Availability**

Supported by all printer, metafile, plotter, and camera devices when using any form of GDOS.

**Parameters**

*handle* specifies a valid workstation handle.

**Binding**

```c
contr1[0] = 4;
contr1[1] = contr1[3] = 0;
contr1[6] = handle;

vdi();
```

**Comments**

This call does not cause the ‘page’ to be ejected. You must use either `v_clrwk()` or `v_form_adv()` to accomplish that.

**See Also**

`v_clrwk()`, `v_form_adv()`
v_write_meta()

VOID v_write_meta( handle, intin_len, intin, ptsin_len, ptsin )
WORD handle, intin_len;
WORD *intin;
WORD ptsin_len;
WORD *ptsin;

v_write_meta() writes a customized metafile sub-opcode.

OPCODE
5

S U B - O P C O D E
99

A VAILABILITY
Supported by all metafile drivers.

P A RAMETERS
handle specifies a valid workstation handle. intin points to an array of WORDs
with intin_len (0-127) elements. ptsin points to an array of WORDs with
ptsin_len (0-127) elements. ptsin is not required to be of any length, however,
intin should be at least one word long to specify the sub-opcode in intin[0]. Sub-
opcodes 0-100 are reserved for use by Atari. Several pre-defined sub-opcodes in
this range already exist as follows:

<table>
<thead>
<tr>
<th>Sub-Opcode:</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>intin[0]</td>
<td></td>
</tr>
<tr>
<td>10</td>
<td>Start group.</td>
</tr>
<tr>
<td>11</td>
<td>End group.</td>
</tr>
<tr>
<td>49</td>
<td>Set no line style.</td>
</tr>
<tr>
<td>50</td>
<td>Set attribute shadow on.</td>
</tr>
<tr>
<td>51</td>
<td>Set attribute shadow off.</td>
</tr>
<tr>
<td>80</td>
<td>Start draw area type primitive.</td>
</tr>
<tr>
<td>81</td>
<td>End draw area type primitive.</td>
</tr>
</tbody>
</table>

B I NDING

WORD i;
contrl[0] = 5;
contrl[1] = ptsin_len;
contrl[3] = intin_len;
contrl[5] = 99;
contrl[6] = handle;

for(i = 0; i < intin_len; i++)
    intin[i] = m_intin[i];
for(i = 0; i < ptsin_len; i++)
    ptsin[i] = m_ptsin[i];
vdi();

**COMMENTS**
Metafile readers should ignore and safely skip any opcodes not understood.

---

## vex_butv()

VOID vex_butv( handle, butv, old_butv )
WORD handle;
WORD (*butv)( (WORD) bstate );
WORD (**old_butv)( (WORD) bstate );

---

**vex_butv()** installs a routine which is called by the **VDI** every time a mouse button is pressed.

### OPCODE
125

### AVAILABILITY
Supported by all screen drivers.

### PARAMETERS
*handle* specifies a valid physical workstation handle. *butv* points to a user-defined button-click handler routine. The address pointed to by *old_butv* will be filled in with the address of the old button-click handler.

### BINDING
contrl[0] = 125;
contrl[6] = handle;
contrl[7] = (WORD)((LONG)butv >> 16);
contrl[8] = (WORD)((LONG)butv);

```c
vdi();
*(LONG *)old_butv = (LONG)(((LONG)contrl[9] << 16) |
                          (LONG)contrl[10]);
```

### COMMENTS
Upon entry to *butv*, the mouse status is contained in 68x00 register D0 (in the same format as the button return value in **vq_mouse()**). A ‘C’ handler should, therefore, be sure to specify register calling parameters for this function. Any registers which will be modified should be saved and restored upon function exit. The routine may call the **BIOS** and/or **XBIOS** sparingly but should not call the **AES**, **VDI**, or **GEMDOS**.

### SEE ALSO
vex_curv(), vex_motv()
vex_curv() installs a routine which is called every time the mouse cursor is drawn allowing a customized mouse rendering routine to replace that of the system.

**OPCODE**

126

**AVAILABILITY**

Supported by all screen devices.

**PARAMETERS**

*handle* specifies a valid physical workstation handle. *curv* points to a user defined function which will be called every time the mouse is to be refreshed. *old_curv* is the address of a pointer to the old rendering routine which will be filled in by the function on exit.

**BINDING**

```c
contrl[0] = 126;
contrl[6] = handle;
contrl[7] = (WORD)((LONG)curv >> 16);
contrl[8] = (WORD)((LONG)curv);

vdi();

*(LONG *)old_curv = (LONG)(((LONG)contrl[9] << 16) | (LONG)contrl[10]);
```

**COMMENTS**

Upon entry to *curv*, the mouse’s X and Y location on screen is contained in 68x00 registers D0 and D1 respectively. A ‘C’ handler should, therefore, be sure to specify register calling parameters for this function. Any registers which will be modified should be saved and restored upon function exit. The routine may call the BIOS and/or XBIOS sparingly but should not call the AES, VDI, or GEMDOS.

**SEE ALSO**

vex_butv(), vex_motv()
**vex_motv()**

VOID vex_motv( handle, motv, old_motv )

WORD handle;

WORD (*motv)( (WORD) mx, (WORD) my );

WORD (**old_motv)( (WORD) mx, (WORD) my );

vex_motv() installs a user routine which is called every time the mouse pointer is moved.

**OPCODE**

126

**AVAILABILITY**

Supported by all screen drivers.

**PARAMETERS**

*handle* specifies a valid physical workstation handle. *motv* points to a user-defined routine which is called every time the mouse is moved. *old_motv* is an address to a pointer which will be filled in containing the address of the old function.

**BINDING**

\[
\begin{align*}
\text{ctrl}[0] &= 126; \\
\text{ctrl}[1] &= \text{ctrl}[3] = 0; \\
\text{ctrl}[6] &= \text{handle}; \\
\text{ctrl}[7] &= (\text{WORD})((\text{LONG})\text{motv} >> 16); \\
\text{ctrl}[8] &= (\text{WORD})((\text{LONG})\text{motv}); \\
\text{vdi}(); \\
*(\text{LONG} *)\text{old_motv} &= (\text{LONG})((\text{LONG})\text{ctrl}[9] << 16) | \\
& (\text{LONG})\text{ctrl}[10]);
\end{align*}
\]

**COMMENTS**

Upon entry to *motv*, the mouse’s new X and Y location is contained in 68x00 registers D0 and D1 respectively. A ‘C’ handler should, therefore, be sure to specify register calling parameters for this function. Any registers which will be modified should be saved and restored upon function exit. The routine may call the BIOS and/or XBIOS sparingly but should not call the AES, VDI, or GEMDOS. The routine may modify the contents of D0 and D1 as necessary to affect the movement of the mouse (one way of implementing a mouse accelerator).

**SEE ALSO**

vex_curv(), vex_butv()
vex_timv()

VOID vex_timv( handle, timv, old_timv, mpt )
WORD handle;
VOID (*timv)( VOID );
VOID (**old_timv)( VOID );
WORD *mpt;

vex_timv() installs a user-defined routine that will be called at each timer tick (currently once every 50 milliseconds).

OPCODE 118

AVAILABILITY Supported by all screen drivers.

PARAMETERS handle specifies a valid physical workstation handle. timv should point to a user-defined timer tick routine. old_timv is an address to a pointer which will be filled in with the old timer tick routine. mpt is a pointer to a WORD which will be filled in with the value representing the current number of milliseconds per timer tick.

BINDING

contrl[0] = 118;
contrl[6] = handle;
contrl[7] = (WORD)((LONG)timv >> 16);
contrl[8] = (WORD)((LONG)timv);

vdi();

*(LONG *)old_timv = (LONG)(((LONG)contrl[9] << 16) | (LONG)contrl[10]);

COMMENTS Any registers which will be modified should be saved and restored upon function exit. The routine may call the BIOS and/or XBIOS sparingly but should not call the AES, VDI, or GEMDOS. The routine should fall through to the old routine. As this vector is jumped through quite often, the routine should be very simple to avoid system performance slowdowns.

vm_coords()

VOID vm_coords( handle, xmin, ymin, xmax, ymax )
WORD handle, xmin, ymin, xmax, ymax;

vm_coords() allows the use of variable coordinate systems with metafiles.

OPCODE 5
SUB-OPCODES 99, 1

AVAILABILITY Supported by all metafile drivers.

PARAMETERS handle specifies a valid workstation handle. xmin and ymin specify the coordinate pair which provides an anchor for the upper-left point of the coordinate system. xmax and ymax specify the coordinate pair which provides an anchor for the lower-right point of the coordinate system.

BINDING

```c
ctrl[0] = 5;
ctrl[1] = 0;
ctrl[3] = 5;
ctrl[5] = 99;
ctrl[6] = handle;

intin[0] = 1;
intin[1] = xmin;
intin[2] = ymin;
intin[3] = xmax;
intin[4] = ymax;
```

vdi();

COMMENTS Use of this function allows the use of practically any coordinate system with a limit of (-32768, -32768), (32767, 32767).

Metafiles default to a coordinate space of (0, 32767), (32767, 0).

SEE ALSO vm_pagesize(), v_meta_extents()

---

### vm_filename()

VOID vm_filename( handle, fname )

WORD handle;
char *fname;

vm_filename() allows specifying a user-defined filename for metafile output.

OPCODE 5

SUB-OPCODE 100

AVAILABILITY Supported by all metafile drivers.

PARAMETERS handle specifies a valid workstation handle. fname points to a NULL-terminated GEMDOS filename which all metafile output should be redirected to.
**vm_pagesize()**

**VOID vm_pagesize( handle, pwidth, pheight )**

**WORD handle, pwidth, pheight;**

`vm_pagesize()` specifies a metafile’s source page size.

**OPCODE**

5

**SUB-OPCODES**

99, 0

**AVAILABILITY**

Supported by all metafile drivers.

**PARAMETERS**

`handle` specifies a valid workstation handle. `pwidth` specifies the width of the page which the metafile was originally placed on in tenths of a millimeter. `pheight` specifies the height of the page which the metafile was originally placed on in tenths of a millimeter.

**BINDING**

```c
WORD i = 0;

while(intin[i++] = (WORD)*fname++);

contrl[0] = 5;
contrl[1] = 0;
contrl[3] = --i;
contrl[5] = 100;
contrl[6] = handle;

vdi();
```

**CAVEATS**

When a metafile is opened, the default file ‘GEMFILE.GEM’ is created in the current GEMDOS path on the current drive and is not deleted as a result of this call. You will need to manually delete it yourself.

**COMMENTS**

This call should be made immediately after a `v_opnw()` to a metafile handle if you wish to use an alternate filename to prevent data from being lost.
A metafile originally designed on an 8.5” x 11” page would have a \textit{pwidth} value of 2159 and a \textit{pheight} value of 2794.

\textbf{SEE ALSO} \texttt{v\_meta\_extents()}

\section*{vq\_cellarray()}

\texttt{VOID vq\_cellarray( handle, pxy, rowlen, num\_rows, elements, rows\_used, status, colarray )}

\texttt{WORD handle;}
\texttt{WORD *pxy;}
\texttt{WORD rowlen, num\_rows;}
\texttt{WORD *elements, *rows\_used, *status, *colarray;}

\texttt{vq\_cellarray()} returns the cell array definitions of specified pixels.

\textbf{OPCODE} 27

\textbf{AVAILABILITY} Not supported by any known drivers.

\textbf{PARAMETERS} \texttt{handle} specifies a valid workstation handle. \texttt{pxy} points to an array of 4 \texttt{WORD}s which specify a VDI format rectangle. \texttt{row\_length} specifies the length of each row in the color array. \texttt{num\_rows} specifies the number of total rows in the color array.

Upon return, the \texttt{WORD} pointed to by \texttt{elements} will indicate the number of array elements used per row. In addition, \texttt{rows\_used} will be filled in with actual number of rows used by the color array and the \texttt{WORD} pointed to by \texttt{status} will be filled in with 0 if the operation was successful or 1 if at least one element could not be determined. Finally, the \texttt{WORD} array (with \texttt{(num\_rows \times row\_length)} elements) pointed to by \texttt{colarray} will be filled in with the color index array stored one row at a time. On return \texttt{colarray} will actually contain \texttt{(elements \times rows\_used)} valid elements.

\textbf{BINDING}

\begin{verbatim}
WORD i;

cntrl[0] = 27;
cntrl[1] = 2;
cntrl[3] = 0;
cntrl[6] = handle;
cntrl[7] = row_length;
cntrl[8] = num_rows;

ptsin[0] = pxy[0];
ptsin[1] = pxy[1];
ptsin[2] = pxy[2];
ptsin[3] = pxy[3];
\end{verbatim}
vq_chcells()

VOID vq_chcells( handle, rows, columns )
WORD handle;
WORD *rows, *columns;

vq_chcells() returns the current number of columns and rows on the alpha text mode of the device.

Opcode 5

Sub-Opcode 1

Availability Supported by all screen and printer drivers.

Parameters

handle specifies a valid workstation handle. rows and columns each point to a WORD which will be filled in with the current number of rows and columns of the device (in text mode).

Binding

contrl[0] = 5;
contrl[5] = 1;
contrl[6] = handle;
vd1();

*rows = intout[0];
*columns = intout[1];

See Also v_curtext()
vq_color()

WORD vq_color( handle, index, flag, rgb)
WORD handle, index, flag;
WORD *rgb;

vq_color() returns RGB information for a particular VDI color index.

OPCODE

26

AVAILABILITY

Supported by all drivers.

PARAMETERS

handle specifies a valid workstation handle. index specifies the VDI color index of which you wish to inquire. rgb points to an array of 3 WORDs which will be filled in with the red, green, and blue values (0-1000) of the color index. The values returned in the RGB array are affected by the value of flag as follows:

<table>
<thead>
<tr>
<th>Name</th>
<th>flag</th>
<th>Values returned in rgb</th>
</tr>
</thead>
<tbody>
<tr>
<td>COLOR_REQUESTED</td>
<td>0</td>
<td>Return the values as last requested by the user (ie: not mapped to the actual color value displayed).</td>
</tr>
<tr>
<td>COLOR_ACTUAL</td>
<td>1</td>
<td>Return the values as the actual color being displayed.</td>
</tr>
</tbody>
</table>

BINDING

contrl[0] = 26;
contrl[1] = 0;
contrl[3] = 2;
contrl[6] = handle;

intin[0] = index;
intin[1] = flag;

vdi();

rgb[0] = intout[1];
rgb[1] = intout[2];
rgb[2] = intout[3];

return intout[0];

RETURN VALUE

vq_color() returns -1 if the specified index is out of range for the device.

COMMENTS

Some drivers for color printers do not allow you to modify the color of each register. A simple test will allow you to determine if the driver will allow you to change index colors as follows:

• Call vq_color() with a flag value of 0 and save the return.
• Call vs_color() to modify that color index by a significant value.
• Call vq_color() with a flag value of 0 and compare with what you set.
• Restore the old value.
• If equivalent values are returned, you may modify each color index.

SEE ALSO vs_color()

---

**vq_curaddress()**

VOID vq_curaddress( handle, row, column )
WORD handle;
WORD *row, *column;

```
vq_curaddress() returns the current position of the alpha text cursor.
```

**OPCODE** 5

**SUB-OPCODE** 15

**AVAILABILITY** Supported by all screen drivers.

**PARAMETERS** handle specifies a valid workstation handle. The WORDs pointed to by row and column will be filled in with the current row and column respectively of the text cursor in alpha mode.

**BINDING**

```
contrl[0] = 5;
contrl[5] = 15;
contrl[6] = handle;

vdi();

*row = intout[0];
*column = intout[1];
```

SEE ALSO v_curtext(), vq_chcells()

---

**vq_extnd()**

VOID vq_extnd( handle, mode, work_out )
WORD handle, mode;
WORD *work_out;

```
vq_extnd() returns extra information about a particular workstation.
```

**OPCODE** 102
The Availability

Supported by all drivers.

The Parameters

handle specifies a valid workstation handle. If mode is set to 0 then this call fills in the array pointed to by work_out with the same 57 WORDs which are returned by either v_opnwk() or v_opnvwk(). If mode is 1 then the 57 WORDs of work_out are filled in with other information as follows:

<table>
<thead>
<tr>
<th>work_out[x]</th>
<th>VDI Structure Member</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>screentype</td>
<td>Type of display screen:</td>
</tr>
<tr>
<td></td>
<td></td>
<td>0 = Not screen.</td>
</tr>
<tr>
<td></td>
<td></td>
<td>1 = Separate alpha/graphic controllers and displays.</td>
</tr>
<tr>
<td></td>
<td></td>
<td>2 = Separate alpha/graphic controllers with common screen.</td>
</tr>
<tr>
<td></td>
<td></td>
<td>3 = Common alpha/graphic controllers with separate image memory.</td>
</tr>
<tr>
<td></td>
<td></td>
<td>4 = Common alpha/graphic controllers and image memory.</td>
</tr>
<tr>
<td></td>
<td></td>
<td>(All known devices either return 0 or 4.)</td>
</tr>
<tr>
<td>1</td>
<td>bgcolors</td>
<td>Number of background colors available.</td>
</tr>
<tr>
<td>2</td>
<td>textfx</td>
<td>Text effects supported. (Same bitmask as with vst_effects().)</td>
</tr>
<tr>
<td>3</td>
<td>canscale</td>
<td>Scaling of rasters:</td>
</tr>
<tr>
<td></td>
<td></td>
<td>0 = Can't scale.</td>
</tr>
<tr>
<td></td>
<td></td>
<td>1 = Can scale.</td>
</tr>
<tr>
<td>4</td>
<td>planes</td>
<td>Number of planes.</td>
</tr>
<tr>
<td>5</td>
<td>lut</td>
<td>Lookup table supported:</td>
</tr>
<tr>
<td></td>
<td></td>
<td>0 = Table not supported.</td>
</tr>
<tr>
<td></td>
<td></td>
<td>1 = Table supported.</td>
</tr>
<tr>
<td></td>
<td></td>
<td>(True color modes return a value of 0 for lut and &gt;2 for colors in v_opnvwk()).</td>
</tr>
<tr>
<td></td>
<td></td>
<td>See the caveat listed below.</td>
</tr>
<tr>
<td>6</td>
<td>rops</td>
<td>Performance factor. Number of 16x16 raster operations per second.</td>
</tr>
<tr>
<td>7</td>
<td>cancontourfill</td>
<td>v_contourfill() availability:</td>
</tr>
<tr>
<td></td>
<td></td>
<td>0 = Not available.</td>
</tr>
<tr>
<td></td>
<td></td>
<td>1 = Available.</td>
</tr>
<tr>
<td>8</td>
<td>textrot</td>
<td>Character rotation capability:</td>
</tr>
<tr>
<td></td>
<td></td>
<td>0 = None.</td>
</tr>
<tr>
<td></td>
<td></td>
<td>1 = 90 degree increments.</td>
</tr>
<tr>
<td></td>
<td></td>
<td>2 = Any angle of rotation.</td>
</tr>
<tr>
<td>9</td>
<td>writemodes</td>
<td>Number of writing modes available.</td>
</tr>
<tr>
<td>10</td>
<td>inputmodes</td>
<td>Highest level of input modes available:</td>
</tr>
<tr>
<td></td>
<td></td>
<td>0 = None.</td>
</tr>
<tr>
<td></td>
<td></td>
<td>1 = Request.</td>
</tr>
<tr>
<td></td>
<td></td>
<td>2 = Sample.</td>
</tr>
<tr>
<td>11</td>
<td>textalign</td>
<td>Text alignment capability flag:</td>
</tr>
<tr>
<td></td>
<td></td>
<td>0 = Not available.</td>
</tr>
<tr>
<td></td>
<td></td>
<td>1 = Available.</td>
</tr>
<tr>
<td>12</td>
<td>inking</td>
<td>Inking capability flag:</td>
</tr>
<tr>
<td></td>
<td></td>
<td>0 = Device can't ink.</td>
</tr>
<tr>
<td></td>
<td></td>
<td>1 = Device can ink.</td>
</tr>
<tr>
<td>13</td>
<td>rubberbanding</td>
<td>Rubberbanding capability flag:</td>
</tr>
<tr>
<td>-----</td>
<td>------------------------</td>
<td>----------------------------------</td>
</tr>
<tr>
<td></td>
<td>0 = No rubberbanding.</td>
<td></td>
</tr>
<tr>
<td></td>
<td>1 = Rubberbanded lines.</td>
<td></td>
</tr>
<tr>
<td></td>
<td>2 = Rubberbanded lines and rectangles.</td>
<td></td>
</tr>
<tr>
<td>14</td>
<td>maxvertices</td>
<td>Maximum vertices for polyline, polymarker, or filled area (-1 = no maximum).</td>
</tr>
<tr>
<td>15</td>
<td>maxintin</td>
<td>Maximum length of intin array (-1 = no maximum).</td>
</tr>
<tr>
<td>16</td>
<td>mousebuttons</td>
<td>Number of mouse buttons.</td>
</tr>
<tr>
<td>17</td>
<td>widestyles</td>
<td>Styles available for wide lines?</td>
</tr>
<tr>
<td></td>
<td>0 = No</td>
<td></td>
</tr>
<tr>
<td></td>
<td>1 = Yes</td>
<td></td>
</tr>
<tr>
<td>18</td>
<td>widemodes</td>
<td>Writing modes available for wide lines?</td>
</tr>
<tr>
<td></td>
<td>0 = No</td>
<td></td>
</tr>
<tr>
<td></td>
<td>1 = Yes</td>
<td></td>
</tr>
<tr>
<td>19-56</td>
<td>reserved1</td>
<td>Reserved for future use.</td>
</tr>
</tbody>
</table>

**BINDING**

```c
WORD i;

contrl[0] = 102;
contrl[1] = 0;
contrl[3] = 1;
contrl[6] = handle;

intin[0] = mode;

vdi();

for(i = 0; i < 45; i++)
    work_out[i] = intout[i];

for(i = 0; i < 13; i++)
    work_out[45+i] = ptsout[i];
```

**COMMENTS**

See the entry for `V_Opnwk()` and `V_Opnvwk()` to see how the `vq_extnd()` information and `v_opn/v/wk()` calls are integrated into a ‘C’ style structure.

**CAVEATS**

The `lut` member of the `VDIWORK` structure was originally misdocumented by Atari with the values reversed. The Falcon030 as well as some third-party true-color boards return the correct values. Some older boards may not, however.

One alternative method of determining if the current screen is not using a software color lookup table (i.e. true color) is to compare the value for `2 ^ planes` with the number of colors in the palette found in `colors`. If this number is different, the `VDI` is not using a software color lookup table.

**SEE ALSO**

`v_opnwk()`, `v_opnvwk()`, `V_Opnwk()`, `V_Opnvwk()`
vq_gdos()

ULONG vq_gdos( VOID )

vq_gdos() determines the availability and type of GDOS present.

OPCODE

N/A

AVAILABILITY

Supported in ROM by all Atari computers.

BINDING

; Correct binding for vq_gdos. Some compilers
; use the name vq_vgdos for the new version
; and vq_gdos for the old version which
; looked like:
;
;  move.w  #-2,d0
;  trap    #2
;  cmp.w   #-2,d0
;  sne     d0
;  ext.w   d0

_vq_gdos:

move.w  #-2,d0
trap    #2
rts

RETURN VALUE

Currently one of the following values are returned:

<table>
<thead>
<tr>
<th>Name</th>
<th>Value</th>
<th>GDOS Type</th>
</tr>
</thead>
<tbody>
<tr>
<td>GDOS_NONE</td>
<td>-2</td>
<td>GDOS not installed.</td>
</tr>
<tr>
<td>—</td>
<td>Any other value.</td>
<td>GDOS 1.0, 1.1, or 1.2 installed.</td>
</tr>
<tr>
<td>GDOS_FNT</td>
<td>0x5F464E54 ('_FNT')</td>
<td>FONTGDOS installed.</td>
</tr>
<tr>
<td>GDOS_FSM</td>
<td>0x5F46534D ('_FSM')</td>
<td>FSMGDOS installed.</td>
</tr>
</tbody>
</table>

COMMENTS

Calling a GDOS function without GDOS loaded is fatal and will cause a system crash.

To determine whether FSMGDOS or SpeedoGDOS is loaded look for the ‘FSMC’ cookie in the cookie jar. The cookie value points to a longword which will contain either ‘_FSM’ or ‘_SPD’.
vq_key_s()

VOID vq_key_s( handle, status )
WORD handle;
WORD *status;

vq_key_s() returns the current shift-key status.

OPCODE 128

AVAILABILITY Supported by all Atari computers.

PARAMETERS handle specifies a valid workstation handle. status points to a WORD which is filled in on function exit with a bit mask containing the current shift key status as follows:

<table>
<thead>
<tr>
<th>Name</th>
<th>Bit</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>K_RSHIFT</td>
<td>0</td>
<td>Right shift key depressed</td>
</tr>
<tr>
<td>K_LSHIFT</td>
<td>1</td>
<td>Left shift key depressed</td>
</tr>
<tr>
<td>K_CTRL</td>
<td>2</td>
<td>Control key depressed</td>
</tr>
<tr>
<td>K_ALT</td>
<td>3</td>
<td>Alternate key depressed</td>
</tr>
</tbody>
</table>

BINDING

contrl[0] = 128;
contrl[6] = handle;

vd1();

*status = intout[0];

SEE ALSO graf_mkstate()

vq_mouse()

VOID vq_mouse( handle, mb, mx, my )
WORD handle;
WORD *mb, *mx, *my;

vq_mouse() returns information regarding the current state of the mouse.

OPCODE 124

AVAILABILITY Supported by all screen drivers.
**PARAMETERS**

*handle* specifies a valid workstation handle. *mb* points to a **WORD** which will be filled in upon function exit with a bit mask indicating the current status of the mouse buttons as follows:

<table>
<thead>
<tr>
<th>Name</th>
<th>Mask</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>LEFT_BUTTON</td>
<td>0x01</td>
<td>Left mouse button</td>
</tr>
<tr>
<td>RIGHT_BUTTON</td>
<td>0x02</td>
<td>Right mouse button</td>
</tr>
<tr>
<td>MIDDLE_BUTTON</td>
<td>0x04</td>
<td>Middle button (this button would be the first button to the left of the rightmost button on the device).</td>
</tr>
<tr>
<td></td>
<td>0x08</td>
<td>Other buttons (0x08 is the mask for the button to the immediate left of the middle button. Masks continue leftwards).</td>
</tr>
</tbody>
</table>

*mx* and *my* both point to **WORDS** which will be filled in upon function exit with the current position of the mouse pointer.

**BINDING**

```c
  ctrl[0] = 124;
  ctrl[6] = handle;

  vdi();

  *mb = intout[0];
  *mx = ptsout[0];
  *my = ptsout[1];
```

**SEE ALSO**

*graf_mkstate(), v_key_s()*

---

**vq_scan()**

**VOID vq_scan( handle, grh, passes, alh, apage, div )**

**WORD handle;**


*vq_scan()* returns information regarding printer banding.

**OPCODE**

5

**SUB-OPCODE**

24

**AVAILABILITY**

Supported by all printer drivers.

**PARAMETERS**

*handle* specifies a valid workstation handle. *passes* specifies the number of graphic passes per printer page.
The value obtained through the formula $grh/div$ specifies the number of graphics scan lines per pass. The value obtained by the formula $alh/div$ specifies the number of graphic scan lines per alpha text line. $apage$ specifies the number of alpha lines per page.

**BINDING**

```c
contrl[0] = 5;
contrl[5] = 24;
contrl[6] = handle;

vdi();

*grh = intout[0];
*passes = intout[1];
*alh = intout[2];
*apage = intout[3];
*div = intout[4];
```

**COMMENTS**

This call has been previously mis-documented.

---

**vq_tabstatus()**

**OPCODE**

5

**SUB-OPCODE**

16

**AVAILABILITY**

Supported by all screen drivers.

**PARAMETERS**

$handle$ specifies a valid workstation handle.

**BINDING**

```c
contrl[0] = 5;
contrl[5] = 16;
contrl[6] = handle;

vdi();

return intout[0];
```

**RETURN VALUE**

$vq\_tabstatus()$ returns 0 if no tablet is available or 1 if a tablet device is present.

**SEE ALSO**

$vq\_tdimensions(), vt\_origin(), vt\_axis(), vt\_resolution(), vt\_alignment()$
vq_tdimensions()

VOID vq_tdimensions( handle, xdim, ydim )
WORD handle;
WORD *xdim, *ydim;

vq_tdimensions() returns the scanning dimensions of the attached graphics tablet.

OPCODE 5
SUB-OPCODE 84
AVAILABILITY Supported by all tablet drivers.
PARAMETERS handle specifies a valid workstation handle. xdim and ydim point to WORDs which upon function exit will contain the X and Y dimensions of the tablet scanning area specified in tenths of an inch.
BINDING
contrl[0] = 5;
contrl[5] = 84;
contrl[6] = handle;
vidi();
*xdim = intout[0];
*ydim = intout[1];

SEE ALSO vq_tabstatus()

vqf_attributes()

VOID vqf_attributes( handle, attr )
WORD handle;
WORD *attr;

vqf_attributes() returns information regarding the current fill attributes.

OPCODE 37
AVAILABILITY Supported by all devices.
PARAMETERS handle specifies a valid workstation handle. attr points to an array of five WORDs which upon exit will be filled in as follows:
### vqin_mode()

VOID vqin_mode( handle, dev, mode )

WORD handle, dev;

WORD mode;

vqin_mode() returns the input status of the specified VDI device.

### Opcode

115

### Availability

Supported by all Atari computers.

### Parameters

*handle* specifies a valid workstation handle. *mode* points to a WORD which upon exit will be filled in with 1 if the specified device is in request mode or 2 if in sample mode. *dev* specifies the device to inquire as follows:

<table>
<thead>
<tr>
<th>Name</th>
<th>dev</th>
<th>Device</th>
</tr>
</thead>
<tbody>
<tr>
<td>LOCATOR</td>
<td>1</td>
<td>Locator (Mouse, Mouse Buttons, and Keyboard)</td>
</tr>
<tr>
<td>VALUATOR</td>
<td>2</td>
<td>Valuator (not currently defined)</td>
</tr>
<tr>
<td>CHOICE</td>
<td>3</td>
<td>Choice (not currently defined)</td>
</tr>
<tr>
<td>STRING</td>
<td>4</td>
<td>String (Keyboard)</td>
</tr>
</tbody>
</table>

### Binding

```c
ctrl[0] = 37;
ctrl[6] = handle;

vdi();

attr[0] = intout[0];
attr[1] = intout[1];
attr[2] = intout[2];
attr[3] = intout[3];
attr[4] = intout[4];
```

### See Also

vqt_attributes(), vql_attributes(), vqm_attributes()
contrl[1] = 0
contrl[3] = 1;
contrl[6] = handle;

intin[0] = dev;
vdi();

*mode = intout[0];

SEE ALSO  vsin_mode()

---

**vql_attributes()**

VOID vql_attributes( handle, attr )
WORD handle;
WORD *attr;

vql_attributes() returns information regarding current settings which affects line drawing functions.

**OPCODE**
36

**AVAILABILITY**
Supported by all drivers.

**PARAMETERS**
handle specifies a valid workstation handle. attr is an array of 6 WORDs which describe the current parameters for line drawing as follows:

<table>
<thead>
<tr>
<th>attr[x]</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>Line type (see vsl_type() ).</td>
</tr>
<tr>
<td>1</td>
<td>Line color (see vsl_color() ).</td>
</tr>
<tr>
<td>2</td>
<td>Writing mode (see vswr_mode() ).</td>
</tr>
<tr>
<td>3</td>
<td>End style for start of lines (see vsl_ends() ).</td>
</tr>
<tr>
<td>4</td>
<td>End style for end of lines (see vsl_ends() ).</td>
</tr>
<tr>
<td>5</td>
<td>Current line width (see vsl_width() ).</td>
</tr>
</tbody>
</table>

**BINDING**

contrl[0] = 36;
contrl[6] = handle;

vdi();

attr[0] = intout[0];
attr[1] = intout[1];
attr[2] = intout[2];
attr[3] = intout[3];
attr[4] = intout[4];
att[5] = intout[5];

SEE ALSO  vqm_attributes(), vqt_attributes(), vqf_attributes()

vqm_attributes()

VOID vqm_attributes( handle, attr )
WORD handle;
WORD *attr;

vqm_attributes() returns information regarding current settings which apply to polymarker output.

OPCODE  36

AVAILABILITY  Supported by all drivers.

PARAMETERS  handle specifies a valid workstation handle. attr points to an array of 5 WORDs which specify the current polymarker attributes as follows:

<table>
<thead>
<tr>
<th>att[x]</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>Marker type (see vsm_type() ).</td>
</tr>
<tr>
<td>1</td>
<td>Marker color (see vsm_color() ).</td>
</tr>
<tr>
<td>2</td>
<td>Writing mode (see vswr_mode() ).</td>
</tr>
<tr>
<td>3</td>
<td>Polymarker width (see vsm_height() ).</td>
</tr>
<tr>
<td>4</td>
<td>Polymarker height (see vsm_height() ).</td>
</tr>
</tbody>
</table>

BINDING

contrl[0] = 36;
contrl[6] = handle;

vdi();

att[0] = intout[0];
att[1] = intout[1];
att[2] = intout[2];
att[3] = intout[3];
att[4] = intout[4];

SEE ALSO  vql_attributes(), vqt_attributes(), vqf_attributes()
vqp_error()

WORD vqp_error( handle )
WORD handle;

vqp_error() returns error information for the camera driver.

_OPCODE 5
_SUB_OPCODE 96

_AVAILABILITY Supported by all camera drivers.

_PARAMETERS handle specifies a valid workstation handle.

_BINDING
contrl[0] = 5;
contrl[5] = 96;
contrl[6] = handle;

vdi();
return intout[0];

RETURN VALUE vqp_error() returns the current error state as follows:

<table>
<thead>
<tr>
<th>Return Value</th>
<th>Error State</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>No error.</td>
</tr>
<tr>
<td>1</td>
<td>Open dark slide for print film.</td>
</tr>
<tr>
<td>2</td>
<td>No port at location specified by driver.</td>
</tr>
<tr>
<td>3</td>
<td>Palette not found at specified port.</td>
</tr>
<tr>
<td>4</td>
<td>Video cable disconnected.</td>
</tr>
<tr>
<td>5</td>
<td>Memory allocation error.</td>
</tr>
<tr>
<td>6</td>
<td>Inadequate memory for buffer.</td>
</tr>
<tr>
<td>7</td>
<td>Memory not freed.</td>
</tr>
<tr>
<td>8</td>
<td>Driver file not found.</td>
</tr>
<tr>
<td>9</td>
<td>Driver file is not correct type.</td>
</tr>
<tr>
<td>10</td>
<td>Prompt user to process print film.</td>
</tr>
</tbody>
</table>

_COMMENTS Use of this function does not stop the generation of on-screen messages. You must use vsp_message() to accomplish that.

SEE ALSO vsp_message()
vqp_films()

VOID vqp_films(handle, films )
WORD handle;
char *films;

vqp_films() returns strings which represent up to five possible film types for the camera driver to utilize.

Opcode 5
Sub-Opcode 91
Availability Supported by all camera drivers.

Parameters handle specifies a valid workstation handle. films is a character pointer to a buffer at least 125 characters in length. Upon return films will be filled in with 5 character strings. Bytes 0-24 will contain a string for the first type of film, bytes 25-49 will contain a string for the second type, and so on. These strings are not NULL-terminated but are padded with spaces.

Binding
WORD i;
contrl[0] = 5;
contrl[5] = 91;
contrl[6] = handle;

vdi();

for(i = 0; i < 125; i++)
    films[i] = (char)intout[i];

See Also vqp_state()

vqp_state()

VOID vqp_state(handle, port, film, lightness, interlace, planes, indices )
WORD handle;

vqp_state() returns information regarding the current state of the palette driver.

Opcode 5
**SUB-OPCODE** 92

**AVAILABILITY** Supported by all camera drivers.

**PARAMETERS**

*handle* specifies a valid workstation handle. The rest of the parameters are all **WORDs** which are filled in as follows:

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>port</strong></td>
<td>Communication port number.</td>
</tr>
<tr>
<td><strong>film</strong></td>
<td>Film type (0 – 4).</td>
</tr>
<tr>
<td><strong>lightness</strong></td>
<td>Lightness (-3 – 3). A value of 0 specifies the current f-stop setting. A value of three results in an exposure half as long as normal while a value of 3 results in an exposure twice as long as normal.</td>
</tr>
<tr>
<td><strong>interlace</strong></td>
<td>Interlace mode. A value of 0 is non-interlaced, 1 is interlaced.</td>
</tr>
<tr>
<td><strong>planes</strong></td>
<td>Number of planes (1 – 4)</td>
</tr>
<tr>
<td><strong>indices</strong></td>
<td>This is actually a <strong>WORD</strong> array with at least 16 members. ((2 ^ \text{planes})) members will be filled in with color codes for the driver. <strong>indices[0]</strong> and <strong>indices[1]</strong> will specify the first color, <strong>indices[2]</strong> and <strong>indices[2]</strong> the second, and so on.</td>
</tr>
</tbody>
</table>

**BINDING**

```c
WORD i;

contrl[0] = 5;
contrl[5] = 92;
contrl[6] = handle;

vdi();

*port = intout[0];
*film = intout[1];
*lightness = intout[2];
*interlace = intout[3];
*planes = intout[4];

for(i = 0;i < 21;i++)
    indices[i] = intout[5 + i];
```

**SEE ALSO**

vsp_state()

---

**vqt_advance()**

**VOID vqt_advance( handle, wch, advx, advy, xrem, yrem )**

**WORD handle, wch;**

**WORD *advx, *ady, *xrem, *yrem;**

**vqt_advance()** returns the advance vector and remainder for a character.
**OPCODE** 247

**AVAILABILITY** Available only with **FSMGDOS** or **SpeedoGDOS**.

**PARAMETERS**

*handle* specifies a valid workstation handle. *wch* contains the character which you desire information for. Upon return the **WORDS** pointed to by *advx*, *advy*, *xrem*, and *yrem* will be filled in with the correct advance vector and remainders.

**BINDING**

```c
contrl[0] = 247;
contrl[1] = 0;
contrl[3] = 1;
contrl[6] = handle;

intin[0] = wch;

vdi();

*advx = ptsout[0];
*advy = ptsout[1];
*xrem = ptsout[2];
*yrem = ptsout[3];
```

**COMMENTS**

*advx* and *advy*, when added to the position where the character was rendered will indicate the position to draw the next character. This advance vector works in all directions with all character rotations. *xrem* and *yrem* give the remainder value as a modulus of 16384. These remainders should be summed by an application an managed to nudge the advance vector by a pixel when necessary.

**SEE ALSO**

*vqt_width(), vqt_extent(), vqt_f_extent()*

---

**vqt_advance32()**

```c
VOID vqt_advance32( handle, wch, advx, advy )
WORD handle, wch;
fix31 *advx, *advy;
```

*vqt_advance32()* is a variation of the binding for *vqt_advance()* which returns the advance vector and remainder for a character as two **fix31** values.

**OPCODE** 247

**AVAILABILITY** Available only with **SpeedoGDOS**.

**PARAMETERS**

*handle* specifies a valid workstation handle. *wch* contains the character which you desire information for. Upon return the **fix31**s pointed to by *advx* and *advy* will be filled in with the correct advance vector.
 seven bindings

```c
contrl[0] = 247;
contrl[1] = 0;
contrl[3] = 1;
contrl[6] = handle;

intin[0] = wch;

vdi();

*advx = (fix31)((ptsout[4] << 16) | ptsout[5]);
*advy = (fix31)((ptsout[6] << 16) | ptsout[7]);
```

**Comments**

`advx` and `advy`, when added to the position where the character was rendered will indicate the position to draw the next character. This advance vector works in all directions with all character rotations.

**See Also**

`vqt_width()`, `vqt_extent()`, `vqt_f_extent()`

---

### vqt_attributes()

**VOID vqt_attributes( handle, attr )**

**WORD handle;**

**WORD *attr;**

vqt_attributes() returns information regarding the current attributes which affect text output.

**Opcode**

38

**Availability**

Supported by all drivers.

**Parameters**

`handle` specifies a valid workstation handle. `attr` points to an array containing 10 WORDs which are filled in upon function exit as follows:

<table>
<thead>
<tr>
<th>attr[x]</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>Text face (see vst_font() ).</td>
</tr>
<tr>
<td>1</td>
<td>Text color (see vst_color() ).</td>
</tr>
<tr>
<td>2</td>
<td>Text rotation (see vst_rotation() ).</td>
</tr>
<tr>
<td>3</td>
<td>Horizontal alignment (see vst_alignment() ).</td>
</tr>
<tr>
<td>4</td>
<td>Vertical alignment (see vst_alignment() ).</td>
</tr>
<tr>
<td>5</td>
<td>Writing mode (see vswr_mode() ).</td>
</tr>
<tr>
<td>6</td>
<td>Character width (see vst_height() ).</td>
</tr>
<tr>
<td>7</td>
<td>Character height (see vst_height() ).</td>
</tr>
<tr>
<td>8</td>
<td>Character cell width (see vst_height() ).</td>
</tr>
<tr>
<td>9</td>
<td>Character cell height (see vst_height() ).</td>
</tr>
</tbody>
</table>
vqt_cachesize() - 7.105

BINDING
contrl[0] = 38;
contrl[6] = handle;

vdi();
attr[0] = intout[0];
attr[1] = intout[1];
attr[2] = intout[2];
attr[3] = intout[3];
attr[4] = intout[4];
attr[5] = intout[5];
attr[6] = intout[6];
attr[7] = intout[7];
attr[8] = intout[8];
attr[9] = intout[9];

COMMENTS
The values pertaining to character and cell width and have limited usefulness as they are only constant with non-proportional fonts.

SEE ALSO
vql_attributes(), vqm_attributes(), vqf_attributes()

vqt_cachesize()

WORD vqt_cachesize( handle, which, size )
WORD handle, which;
LONG *size;

vqt_cachesize() returns the size of the largest allocatable block of memory in one of two caches.

OPCODE
255

AVAILABILITY
Available only with FSMGDOS or SpeedoGDOS.

PARAMETERS
handle specifies a valid workstation handle. which specifies which cache. A value of CACHE_CHAR (0) selects the character bitmap cache. A value of CACHE_MISC (1) selects the miscellaneous cache. The LONG pointed to by size will be filled in upon function exit with the size of the largest allocatable block of memory in the selected cache.

BINDING
contrl[0] = 255;
contrl[1] = 0;
contrl[3] = 1;
contrl[6] = handle;

intin[0] = which;

vdi();
*size = (LONG)(((LONG)intin[0] << 16) | (LONG)intin[1]);

**COMMENTS**

An application can estimate the amount of memory required to generate a character and print a warning message if the user attempts to exceed it. **FSMGDOS** will simply print a message on screen (you can intercept this with `vst_error()` ) and ask the user to reboot. You can estimate the amount of memory required for a particular character in the character bitmap cache with the formula:

\[(\text{width in pixels} + 7)/8 \times \text{height in pixels}\]

Likewise, you can estimate the amount of memory needed for the miscellaneous cache as:

\[84 \times (\text{width} + \text{height})\]

**SEE ALSO**

`vst_error()` , `v_flushcache()`

---

**vqt_devinfo()**

**VOID vqt_devinfo( handle, devid, exists, devstr )**

**WORD handle, devid;**

**WORD *exists;**

**char *devstr;**

`vqt_devinfo()` determines if a particular device ID is available, and if so, the name of the device driver.

**OPCODE**

248

**AVAILABILITY**

Available only with **FONTGDOS, FSM, or SpeedoGDOS**.

**PARAMETERS**

`handle` specifies a valid workstation handle. `devid` specifies the device ID as listed in the ‘ASSIGN.SYS’ file. `exists` is a pointer to a **WORD** which will be filled in with **DEV_INSTALLED** (1) if a device is installed with the specified ID number or **DEV_MISSING** (0) if not. If the device does exist, the character buffer pointer to by `devstr` will be filled in with the filename of the device padded with spaces to the standard **GEMDOS 8 + 3** format.

**BINDING**

**WORD i;**

```c
contrl[0] = 248;
contrl[1] = 0;
contrl[3] = 1;
contrl[6] = handle;
intin[0] = devid;
```
vqt_extent()

VOID vqt_extent( handle, str, pts )
WORD handle;
char *str;
WORD *pts;

vqt_extent() returns the pixel extent of a string of text.

OPCODE 116

AVAILABILITY Supported by all drivers.

PARAMETERS handle specifies a valid workstation handle. str points to a text string to return extent information for. pts points to an array of 8 WORDs which will be filled in as follows:

```
  4  3
  1  2
```

<table>
<thead>
<tr>
<th>pts[x]</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>X coordinate of point 1.</td>
</tr>
<tr>
<td>1</td>
<td>Y coordinate of point 1.</td>
</tr>
<tr>
<td>2</td>
<td>X coordinate of point 2.</td>
</tr>
<tr>
<td>3</td>
<td>Y coordinate of point 2.</td>
</tr>
<tr>
<td>4</td>
<td>X coordinate of point 3.</td>
</tr>
<tr>
<td>5</td>
<td>Y coordinate of point 3.</td>
</tr>
<tr>
<td>6</td>
<td>X coordinate of point 4.</td>
</tr>
<tr>
<td>7</td>
<td>Y coordinate of point 4.</td>
</tr>
</tbody>
</table>

BINDING

WORD i = 0;

while(intin[i++] = (WORD)*str++);

contrl[0] = 116;
contrl[1] = 0;
contrl[3] = --i;
contrl[6] = handle;

vdi();

pts[0] = ptsout[0];
pts[1] = ptsout[1];
pts[2] = ptsout[2];
pts[3] = ptsout[3];
pts[4] = ptsout[4];
pts[5] = ptsout[5];
pts[6] = ptsout[6];
pts[7] = ptsout[7];

**COMMENTS**

This function will also output correct bounding information for rotated text. It is recommended that `vqt_f_extent()` be used for outline fonts as it takes special factors into consideration which makes its output more accurate.

**SEE ALSO**

`vqt_f_extent()`, `vqt_advance()`, `vqt_width()`

---

### vqt_f_extent()

**VOID vqt_f_extent( handle, str, pts )**

WORD *handle;

char *str;

WORD *pts;

`vqt_f_extent()` returns the bounding box required to enclose the specified string of text.

**OPCODE**

240

**AVAILABILITY**

Available only with **FSMGDOS** or **SpeedoGDOS**.

**PARAMETERS**

Same as `vqt_extent()`.

**BINDING**

```c
WORD i = 0;

while(intin[i++] = (WORD)*str++);

contrl[0] = 240;
contrl[1] = 0;
contrl[3] = --i;
contrl[6] = handle;

vdi();

pts[0] = ptsout[0];
pts[1] = ptsout[1];
pts[2] = ptsout[2];
pts[3] = ptsout[3];
pts[4] = ptsout[4];
```
vqt_f_extent16()

VOID vqt_f_extent( handle, wstr, wstrlen, pts )

WORD handle;
WORD *wstr;
WORD wstrlen;
WORD *pts;

vqt_f_extent16() is a variant binding of vqt_f_extent() that returns the bounding box required to enclose the specified string of 16-bit Speedo character indexed text.

OPCODE

240

AVAILABILITY

Available only with FSMGDOS or SpeedoGDOS.

PARAMETERS

handle specifies a valid workstation handle. wstr points to a 16-bit text string composed of Speedo character indexes. wstrlen indicates the length of wstr. The array pointed to by pts is filled in with the same values as vqt_extent().

BINDING

WORD i;

for( i = 0; i < wstrlen; i++)
    intin[i] = wstr[i];

contrl[0] = 240;
contrl[1] = 0;
contrl[3] = wstrlen;
contrl[6] = handle;

vdi();

pts[0] = ptsout[0];
pts[1] = ptsout[1];
pts[2] = ptsout[2];
pts[3] = ptsout[3];
pts[4] = ptsout[4];
pts[5] = ptsout[5];
pts[6] = ptsout[6];
pts[7] = ptsout[7];
**vqt_fontheader()**

VOID vqt_fontheader( handle, buffer, pathname )
WORD *handle;
char *buffer, *pathname;

vqt_fontheader() returns font-specific information for the currently selected Speedo font.

**OPCODE**
234

**AVAILABILITY**
Available only with SpeedoGDOS.

**PARAMETERS**
handle specifies a valid workstation handle. buffer should point to a buffer of at least 421 bytes into which the font header will be copied. pathname should point to a buffer of at least 128 bytes into which the full pathname of the font’s corresponding `.TDF` file will be copied.

**BINDING**

WORD i;
contrl[0] = 234;
contrl[1] = 0;
contrl[3] = 2;
contrl[6] = handle;
vdii();

for(i = 0; i < contrl[4]; i++)
    pathname[i] = (char)intout[i];

**COMMENTS**
The font header format and `.TDF` file contents are contained in Appendix G: Speedo Fonts.

**SEE ALSO**
vqt_fontinfo()
vqt_fontinfo()

VOID vqt_fontinfo( handle, first, last, dist, width, effects )
WORD handle;
WORD *first, *last, *dist, *width, *effects;

vqt_fontinfo() returns information regarding the current text font.

OPCODE 131

AVAILABILITY Supported by all drivers.

PARAMETERS handle specifies a valid workstation handle. first and last each point to a WORD which will be filled in with the first and last character in the font respectively. dist points to an array of 5 WORDs which indicate the distances between the baseline and the point indicated as follows:

```
+------------------+
|                  |
|   dist[4]       |
|      dist[3]    |
|         dist[2] |
|            G   |
|     g         |
|                |
|                |
|                |
|   dist[1]   |
|dist[0]       |
+------------------+
```

Baseline

width specifies the width of the largest cell in the font in pixels not including effects. effects points to an array of 3 WORDs which contain information relating to the offsets of the font when printed with the current effects.

```
effects[0]
+--------+
|        |
|        |
|        |
|        |
|        |
|        |
+--------+
effects[1]
effects[2] = effects[0] + effects[1]
```

effects[0] specifies the number of X pixels of the left slant. effects[1] specifies the number of X pixels of the right slant. effects[2] specifies the extra number of X
pixels to add to compensate for the special effects.

**BINDING**

```c
contrl[0] = 131;
contrl[6] = handle;

vdi();

*first = intout[0];
*last = intout[1];
*width = ptsout[0];
dist[0] = ptsout[1];
dist[1] = ptsout[3];
dist[2] = ptsout[5];
dist[3] = ptsout[7];
effects[0] = ptsout[2];
effects[1] = ptsout[4];
effects[2] = ptsout[6];
```

**CAVEATS**

SpeedoGDOS is not capable of generating values for \( \text{dist}[1] \) or \( \text{dist}[2] \) so \( \text{dist}[1] \) is set to equal \( \text{dist}[0] \) and \( \text{dist}[2] \) is set to equal \( \text{dist}[3] \).

**SEE ALSO**

vqt_width()

---

### vqt_get_table()

**VOID vqt_get_table( handle, map )**  
**WORD handle;**  
**VOID **map;**

vqt_get_table() returns pointers to seven tables which map the Atari character set to the Bitstream character indexes.

**OPCODE**

254

**AVAILABILITY**

Available only with SpeedoGDOS.

**PARAMETERS**

*handle* specifies a valid workstation handle. The location pointed to by map will be filled in with a pointer to seven internal tables, each 224 WORD size entries long mapping ASCII characters 32–255 to Bitstream character indexes.

The tables are defined as follows:

<table>
<thead>
<tr>
<th>Position</th>
<th>Table</th>
</tr>
</thead>
<tbody>
<tr>
<td>1st</td>
<td>Master mapping.</td>
</tr>
<tr>
<td>2nd</td>
<td>Bitstream International Character Set</td>
</tr>
<tr>
<td>3rd</td>
<td>Bitstream International Symbol Set</td>
</tr>
</tbody>
</table>
vqt_name()

WORD vqt_name( handle, index, fontname )
WORD handle;
WORD index;
char *fontname;

vqt_name() returns the name of the specified font.

OPCODE

130

AVAILABILITY

Supported by all drivers.

PARAMETERS

handle specifies a valid workstation handle. fontname points to a character buffer of at least 33 characters which will be filled in with the name of font index and a flag which distinguishes bitmap and outline fonts. fontname[0–31] will contain the name of the font (not necessarily NULL-terminated).

If FSMGDOS or SpeedoGDOS is installed, fontname[32] will contain a flag equalling OUTLINE_FONT (1) if the specified font is an outline font or BITMAP_FONT (0) if it is a bitmap font.

BINDING

WORD i;
contrl[0] = 130;
contrl[1] = 0;
contrl[3] = 1;
contrl[6] = handle;
intin[0] = index;

vdi();
for(i = 0; i < 33; i++)
    fontname[i] = intout[i + 1];

return intout[0];

**RETURN VALUE**  
vqt_name() returns the unique code value which identifies this font (and is passed to vst_font()).

**SEE ALSO**  
vst_load_fonts(), vst_font()

---

### vqt_pairkern()

**VOID vqt_pairkern( handle, char1, char2, x, y )**

**WORD char1, char2;**

**fix31 *x, *y;**

vqt_pairkern() returns adjustment vector information for the kerning of a character pair.

**OPCODE**  
235

**AVAILABILITY**  
Available only with SpeedoGDOS.

**PARAMETERS**  
*handle* specifies a valid workstation handle. *char1* and *char2* specify the left and right members of the character pair to inquire. *x* and *y* will be filled with the adjustment vector for the specified character pair.

**BINDING**  
contr1[0] = 235;
contr1[1] = 0;
contr1[3] = 2;
contr1[6] = handle;

intin[0] = char1;
intin[1] = char2;

vdi();

*x = ((LONG)ptsout[0] << 16 ) | ptsout[1];
*y = ((LONG)ptsout[2] << 16 ) | ptsout[3];

**SEE ALSO**  
vqt_trackkern(), vst_kern()
vqt_trackkern()

VOID vqt_trackkern( handle, x, y )
fix31 *x, *y;

vqt_trackkern() returns the horizontal and vertical adjustment vector for track kerning.

OPCODE 234

AVAILABILITY Available only with SpeedoGDOS.

PARAMETERS handle specifies a valid workstation handle. x and y are the horizontal and vertical adjustment vectors currently used to modify character spacing in track kerning.

BINDING
contrl[0] = 234;
contrl[1] = 0;
contrl[3] = 0;
contrl[6] = handle;

vdi();
*x = ((LONG)ptsout[0] << 16 ) | ptsout[1];
*y = ((LONG)ptsout[2] << 16 ) | ptsout[2];

SEE ALSO vqt_pairkern(), vst_kern()

vqt_width()

WORD vqt_width( handle, wch, cellw, left, right )
WORD handle, wch;
WORD *cellw, *left, *right;

vqt_width() returns information regarding the width of a character cell.

OPCODE 117

AVAILABILITY Supported by all drivers.

PARAMETERS handle specifies a valid workstation handle. The lower eight bits of wch specify the ASCII character to return width information about. The following three values are each WORDs which are filled in by the function upon return with information about the width of the specified character in pixels as illustrated here.
**BINDING**

```c
contrl[0] = 117;
contrl[1] = 0;
contrl[3] = 1;
contrl[6] = handle;

intin[0] = wch;

vdi();

*cellw = ptsout[0];
*left = ptsout[2];
*right = ptsout[4];

return intout[0];
```

**RETURN VALUE**

`vqt_width()` returns `wch` or `-1` if an error occurred.

**CAVEATS**

`vqt_width()` does not take into account remainders when dealing with outline fonts. It is therefore recommended that `vqt_advance()` be used instead when inquiring about outline fonts.

**SEE ALSO**

`vqt_advance()`
vr_recfl()

VOID vr_recfl( handle, pxy )
WORD handle;
WORD *pxy;

vr_recfl() outputs a filled rectangle.

OPCODE 114

AVAILABILITY Supported by all drivers.

PARAMETERS handle specifies a valid workstation handle. pxy points to an array of 4 WORDs which give a VDI format rectangle of the object to draw.

BINDING

```c
contrl[0] = 114;
contrl[1] = 2;
contrl[3] = 0;
contrl[6] = handle;

ptsin[0] = pxy[0];
ptsin[1] = pxy[1];
ptsin[2] = pxy[2];
ptsin[3] = pxy[3];

vdi();
```

COMMENTS vr_recfl(), as opposed to v_bar(), never draws an outline regardless of the settings of vsf_perimeter().

SEE ALSO v_bar()

vr_trnfm()

VOID vr_trnfm( handle, src, dest )
WORD handle;
MFDB *src, *dest;

vr_trnfm() transforms a memory block from device-independent to device-dependent and vice-versa.

OPCODE 110

AVAILABILITY Supported by all drivers.
PARAMETERS  
handle specifies a valid workstation handle. src specifies the MFDB (as defined in vro_cpyfm() ) whereas dest specifies the MFDB of the destination.

BINDING  
control[0] = 110;  
control[6] = handle;  
control[7] = (WORD)((LONG)src >> 16);  
control[8] = (WORD)src;  
control[9] = (WORD)((LONG)dest >> 16);  
control[10] = (WORD)dest;

vdi();

CAVEATS  
While vr_trnfm() will work for in-place transformations, this process can be time-consuming for large forms.

This call will not translate between forms with multiple planes. For instance, you can not translate a 2 plane device-independent image to an 8-plane device-specific image.

COMMENTS  
To stay compatible with future hardware developments it is recommended that all images be initially either stored or manually translated to device-independent format and subsequently converted with this function to match the planar configuration of the device.

When this call is used to transform forms with either 2 or 4 bit planes, color translation is performed on each pixel as follows:

<table>
<thead>
<tr>
<th>Four-Plane Transformations</th>
<th>Two Plane</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Device</strong></td>
<td><strong>VDI</strong></td>
</tr>
<tr>
<td>0000</td>
<td>0</td>
</tr>
<tr>
<td>0001</td>
<td>2</td>
</tr>
<tr>
<td>0010</td>
<td>3</td>
</tr>
<tr>
<td>0011</td>
<td>6</td>
</tr>
<tr>
<td>0100</td>
<td>4</td>
</tr>
<tr>
<td>0101</td>
<td>7</td>
</tr>
<tr>
<td>0110</td>
<td>5</td>
</tr>
<tr>
<td>0111</td>
<td>8</td>
</tr>
</tbody>
</table>

SEE ALSO  
vro_cpyfm()
**vro_cpyfm()**

VOID vro_cpyfm(handle, mode, pxy, src, dest)

WORD handle, mode;
WORD *pxy;
MFDB *src, *dest;

vro_cpyfm() ‘blits’ a screen or memory block from one location to another.

**Opcode**

109

**Availability**

Supported by all screen drivers.

**Parameters**

handle specifies valid workstation handle. mode specifies the writing mode as follows:

<table>
<thead>
<tr>
<th>Name</th>
<th>Mode</th>
<th>Result</th>
</tr>
</thead>
<tbody>
<tr>
<td>ALL_WHITE</td>
<td>0</td>
<td>All zeros.</td>
</tr>
<tr>
<td>S_AND_D</td>
<td>1</td>
<td>source AND destination</td>
</tr>
<tr>
<td>S_AND_NOTD</td>
<td>2</td>
<td>source AND (NOT destination)</td>
</tr>
<tr>
<td>S_ONLY</td>
<td>3</td>
<td>source</td>
</tr>
<tr>
<td>NOTS_AND_D</td>
<td>4</td>
<td>(NOT source) AND destination</td>
</tr>
<tr>
<td>D_ONLY</td>
<td>5</td>
<td>destination</td>
</tr>
<tr>
<td>S_XOR_D</td>
<td>6</td>
<td>source XOR destination</td>
</tr>
<tr>
<td>S_OR_D</td>
<td>7</td>
<td>source OR destination</td>
</tr>
<tr>
<td>NOT_SORD</td>
<td>8</td>
<td>NOT (source OR destination)</td>
</tr>
<tr>
<td>NOT_SXORD</td>
<td>9</td>
<td>NOT (source XOR destination)</td>
</tr>
<tr>
<td>NOT_D</td>
<td>10</td>
<td>NOT destination</td>
</tr>
<tr>
<td>S_OR_NOTD</td>
<td>11</td>
<td>source OR (NOT destination)</td>
</tr>
<tr>
<td>NOT_S</td>
<td>12</td>
<td>NOT source</td>
</tr>
<tr>
<td>NOTS_OR_D</td>
<td>13</td>
<td>(NOT source) OR destination</td>
</tr>
<tr>
<td>NOT_SANDD</td>
<td>14</td>
<td>NOT (source AND destination)</td>
</tr>
<tr>
<td>ALL_BLACK</td>
<td>15</td>
<td>All ones.</td>
</tr>
</tbody>
</table>

pxy points to an array of eight WORDs. pxy[0–3] contains the bounding rectangle of the source block. pxy[4–7] contains the bounding rectangle of the destination block. src and dest each point to an MFDB structure which describes the source and destination memory form. MFDB is defined as follows:

typedef struct
{

**The Atari Compendium**
/ Memory address (NULL = current screen). If you specify a value of NULL, the rest of the structure will be filled out for you. */
VOID *fd_addr;

/* Form width in pixels */
WORD fd_width;

/* Form height in pixels */
WORD fd_height;

/* Form width in WORDs (fd_width + 15)/16 */
WORD fd_wdwidth;

/* Format (0 = device-specific, 1 = VDI format) */
WORD fd_stand;

/* Number of memory planes */
WORD fd_planes;

/* Reserved (set to 0) */
WORD reserved1;
WORD reserved2;
WORD reserved3;
}

BINDING

contrl[0] = 109;
contrl[1] = 4;
contrl[3] = 1;
contrl[6] = handle;
contrl[7] = (WORD)((LONG)src >> 16);
contrl[8] = (WORD)src;
contrl[9] = (WORD)((LONG)dest >> 16);
contrl[10] = (WORD)dest;

intin[0] = mode;

ptsin[0] = pxy[0];
ptsin[1] = pxy[1];
ptsin[2] = pxy[2];
ptsin[3] = pxy[3];
ptsin[4] = pxy[4];
ptsin[5] = pxy[5];
ptsin[6] = pxy[6];
ptsin[7] = pxy[7];

vdib();

COMMENTS

To ‘blit’ a single-plane form to a multi-plane destination, use vrt_cpyfm().

SEE ALSO

vr_trnfm(), vrt_cpyfm()
vrq_choice()

VOID vrq_choice( handle, start, final )
WORD handle, start;
WORD *final;

vrq_choice() accepts input from the ‘choice’ device in request mode.

OPCODE 30

AVAILABILITY This call is not guaranteed to be available with any driver and its use should therefore be restricted.

PARAMETERS handle specifies a valid workstation handle. start indicates the starting value for the choice device (1–???). final points to a WORD which will be filled in upon exit with the results of the request.

BINDING

contrl[0] = 30;
contrl[1] = 0;
contrl[3] = 1;
contrl[6] = handle;
intin[0] = start;
vdri();

*final = intout[0];

COMMENTS Input is sampled until a key is pressed.

SEE ALSO vsm_choice(), vsin_mode()

vrq_locator()

VOID vrq_locator( handle, mx, my, xout, yout, term )
WORD handle, mx, my;
WORD *xout, *yout, *term;

vrq_locator() inputs information from the ‘locator’ device in request mode.

OPCODE 28

AVAILABILITY This call is not guaranteed to be available with any driver and its use should therefore be restricted.
parameters

`handle` specifies a valid workstation handle. To start, the mouse cursor is displayed at the location given by `mx` and `my`. When a key or mouse button is pressed, the call returns. The final location of the mouse pointer is filled into the 2 `WORD`s pointed to by `xout` and `yout`. The `WORD` pointed to by `term` is filled in with the ASCII key of the character that terminated input, 32 (0x20) if the left mouse button was struck, or 33 (0x21) if the right mouse button was struck.

binding

```c
contrl[0] = 28;
contrl[1] = 1;
contrl[3] = 0;
contrl[6] = handle;

ptsin[0] = mx;
ptsin[1] = my;

vd1();

*term = intout[0];

*xout = ptsout[0];
*yout = ptsout[1];
```

comments

Using this function will confuse the AES’s mouse input functions.

see also

`vsm_locator()`, `vsin_mode()`

---

**vrq_string()**

VOID vrq_string( handle, maxlen, echo, outxy, str )

WORD handle, maxlen, echo;
WORD *outxy;
char *str;

vrq_string() waits for input from the ‘string’ device in request mode.

opcode

31

availability

This call is not guaranteed to be available with any driver and its use should therefore be restricted.

parameters

`handle` specifies a valid workstation handle. This call inputs characters from the keyboard into the buffer pointed to by `str` up to `maxlen + 1` characters. If `echo` is set to 1, characters are echoed to the screen at the location given by the two `WORD`s pointed to by `outxy`. If echo is set to 0, no echoing is performed.

binding

```c
WORD i;

contrl[0] = 31;
```
vrq_valuator() – 7.123

contrl[1] = 1;
contrl[3] = 2;
contrl[6] = handle;

intin[0] = maxlen;
intin[1] = echo;

ptsin[0] = outxy[0];
ptsin[1] = outxy[1];

vdi();

for(i = 0; i < contrl[4]; i++)
    str[i] = (char)intout[i];

CAVEATS

The echo parameter is not functional. Character output is never echoed. However, outxy must point to valid memory space or a crash will occur.

COMMENTS

Though this binding does not allow for it, if maxlen is specified as negative, then as many as \(|maxlen| + 1\) characters will be read as keycodes rather than ASCII codes. The values in intout will occupy the full WORD rather than just the lower eight bits. A custom binding could be used to take advantage of this.

SEE ALSO

vsin_mode(), vsm_string()

vrq_valuator()

VOID vrq_valuator( handle, start, *final, *term )
WORD handle, start;
WORD *final, *term;

vrq_valuator() accepts for input from the valuator device until a terminating character is entered in request mode.

OPCODE

29

AVAILABILITY

This call is not guaranteed to be available with any driver and its use should therefore be restricted.

PARAMETERS

handle specifies a valid workstation handle. start specifies the initial value of the valuator device (1–100). When a terminating character has been struck, the WORD pointed to by final will be filled in with the final value of the valuator and the WORD pointed to by term will be filled in with whatever ASCII character caused termination.

BINDING

contrl[0] = 29;
contrl[1] = 0;
contrl[3] = 1;
contrl[6] = handle;
intin[0] = start;

vdi();

*final = intout[0];
*term = intout[1];

**COMMENTS**
The ‘valuator’ is typically the up and down arrow keys. Each key increments or decrements the value by 10 unless the shift key is held in which case it is incremented or decremented by 1.

**SEE ALSO**
vsin_mode(), vsm_valuator()

---

**vrtn_cpyfm()**

VOID vrt_cpyfm( handle, mode, pxy, src, dest, colors )

WORD handle, mode;

WORD *pxy;

MFDB *src, *dest;

WORD *colors;

**vrtn_cpyfm()** ‘blits’ a single-plane source form to a multiple-plane destination.

**OPCODE**
121

**AVAILABILITY**
Supported by all screen drivers.

**PARAMETERS**
handle specifies a valid workstation handle. mode specifies the writing mode (1–4, see vswr_mode()), pxy, src, and dest are defined the same as in vro_cpyfm().

**colors** points to a 2 WORD array which specifies the colors to apply to the ‘blitted’ image. colors[0] is applied to all set bits in the source image and colors[1] is applied to all of the cleared bits.

**BINDING**

control[0] = 121;
control[1] = 4;
control[3] = 3;
control[6] = handle;
control[7] = (WORD)((LONG)src >> 16);
control[8] = (WORD)src;
control[9] = (WORD)((LONG)dest >> 16);
control[10] = (WORD)dest;

intin[0] = mode;
intin[1] = colors[0];
intin[2] = colors[1];

ptsin[0] = pxy[0];
ptsin[1] = pxy[1];
void vs_clip(handle, flag, pxy)

VOID vs_clip( handle, flag, pxy )
WORD handle, flag;
WORD *pxy;

vs_clip() defines the global clipping rectangle and state for the specified workstation.

OPCODE 129

AVAILABILITY Supported by all drivers.

PARAMETERS handle specifies a valid workstation handle. flag is set to CLIP_OFF (0) to turn off clipping or CLIP_ON (1) to enable clipping. If flag is CLIP_ON (1) then pxy should point to a 4 WORD array containing a VDI format rectangle which will serve as the clipping rectangle, otherwise, pxy can be NULL.

BINDING

cntl[0] = 129;
cntl[1] = 2;
cntl[3] = 1;
cntl[6] = handle;

if(intin[0] = flag) {
    pxy[0] = pxy[0];
    pxy[1] = pxy[1];
    pxy[2] = pxy[2];
    pxy[3] = pxy[3];
}

vdi();

COMMENTS All VDI calls are clipped to that workstation's current clipping rectangle.
**vs_color()**

VOID vs_color( handle, color, rgb )  
WORD handle, color;  
WORD *rgb;

vs_color() sets the color of a palette index.

**OPCODE**  
14

**AVAILABILITY**  
Supported by all devices.

**PARAMETERS**  
handle specifies a valid workstation handle. color specifies the color register of the color to modify. rgb points to an array of three WORDs which contain the red, green, and blue values respectively (0–1000) which will be used to map the color index to the closest color value possible.

**BINDING**  
contr1[0] = 14;  
contr1[1] = 0;  
contr1[3] = 4;  
contr1[6] = handle;

intin[0] = color;  
intin[1] = rgb[0];  
intin[2] = rgb[1];  
intin[3] = rgb[2];

vdii();

**SEE ALSO**  
Esetcolor(), Setcolor()

---

**vs_curaddress()**

VOID vs_curaddress( handle, row, column )  
WORD handle, row, column;

vs_curaddress() sets the position of the alpha screen text cursor.

**OPCODE**  
5

**SUB-OPCODE**  
11

**AVAILABILITY**  
Supported by all screen drivers.

**PARAMETERS**  
handle specifies a valid workstation handle. row and column specify the new
coordinates of the text cursor.

**BINDING**

\[
\begin{align*}
\text{ctrl}[0] &= 5; \\
\text{ctrl}[1] &= 0; \\
\text{ctrl}[3] &= 2; \\
\text{ctrl}[5] &= 11; \\
\text{ctrl}[6] &= \text{handle}; \\
\end{align*}
\]

\[
\begin{align*}
\text{intin}[0] &= \text{row}; \\
\text{intin}[1] &= \text{column}; \\
\end{align*}
\]

\text{vdi();}

**COMMENTS**  
This call is equivalent to the ESC-Y VT-52 code.

**SEE ALSO**  
\text{vq_curaddress()}

---

**vs_palette()**

\text{VOID vs_palette( handle, mode )}

\text{WORD handle, mode;}

\text{vs_palette()} selects a CGA palette.

**OPCODE**  
5

**SUB-OPCODE**  
60

**AVAILABILITY**  
This call was originally designed for use on IBM CGA-based computers. Its usefulness and availability are not guaranteed under any driver so it should thus be avoided.

**PARAMETERS**  
\text{handle} specifies a valid workstation handle. A \text{mode} value of 0 selects a palette of red, green, and blue. A \text{mode} value of 1 selects a palette of cyan, magenta, and white.

**BINDING**

\[
\begin{align*}
\text{ctrl}[0] &= 5; \\
\text{ctrl}[1] &= 0; \\
\text{ctrl}[3] &= 1; \\
\text{ctrl}[5] &= 60; \\
\text{ctrl}[6] &= \text{handle}; \\
\end{align*}
\]

\[
\begin{align*}
\text{intin}[0] &= \text{mode}; \\
\text{vdi();} \\
\end{align*}
\]
vsc_form()

VOID vsc_form( handle, newform )
MFORM *newform;

vsc_form() alters the appearance of the mouse pointer.

_OPCODE 111

_AVAILABILITY Supported by all screen drivers.

_PARAMETERS handle specifies a valid workstation handle. newform points to a MFORM structure defined as follows:

typedef struct
{
    WORD mf_xhot; /* X 'hot spot' */
    WORD mf_yhot; /* Y 'hot spot' */
    WORD mf_nplanes; /* Number of planes (must be 1) */
    WORD mf_fg; /* Foreground color (should be 0) */
    WORD mf_bg; /* Background color (should be 1) */
    WORD mf_mask[16]; /* 16 WORDs of mask*/
    WORD mf_data[16]; /* 16 WORDs of data */
} MFORM;

_BINDING WORD i;

contrl[0] = 111;
contrl[1] = 0;
contrl[3] = 37;
contrl[6] = handle;

for(i = 0;i < 37;i++)
    intin[i] = ((WORD *)newform)[i];

vdi();

SEE ALSO graf_mouse()

vsf_color()

WORD vsf_color( handle, color )
WORD handle, color;

vsf_color() changes the current fill color.
vsf_interior()

WORD vsf_interior( handle, interior )
WORD handle, interior;

vsf_interior() sets the interior type for filled objects.

Opcode 25

Availability Supported by all drivers.

Parameters handle specifies a valid workstation handle. color specifies the new fill color index.

Binding

```
contrl[0] = handle;
contrl[1] = 0;
contrl[3] = 1;
contrl[6] = handle;
intin[0] = color;
vdii();
```

Return Value vsf_color() returns the actual color set (within bounds).

See Also vst_color(), vsm_color(), vsl_color(), vsf_attributes()

<table>
<thead>
<tr>
<th>Name</th>
<th>interior</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>FIS_HOLLOW</td>
<td>0</td>
<td>Hollow interior (color index 0).</td>
</tr>
<tr>
<td>FIS_SOLID</td>
<td>1</td>
<td>Solid interior (as set by vsf_color() ).</td>
</tr>
<tr>
<td>FIS_PATTERN</td>
<td>2</td>
<td>Patterned fill. (style set by vsf_style() ).</td>
</tr>
<tr>
<td>FIS_HATCH</td>
<td>3</td>
<td>Hatched fill. (style set by vsf_style() ).</td>
</tr>
<tr>
<td>FIS_USER</td>
<td>4</td>
<td>User-defined fill (as set by vsf_udpat() ).</td>
</tr>
</tbody>
</table>

Opcode 23

Availability Supported by all drivers.

Parameters handle specifies a valid workstation handle. interior specifies the interior type as follows:
int in[0] = interior;

vdi();

RETURN VALUE
This call returns the color value actually set (within bounds).

SEE ALSO
vsf_style()

vsf_perimeter()

WORD vsf_perimeter( handle, flag )

WORD handle, flag;

vsf_perimeter() sets whether a border will be drawn around most VDI objects.

OPCODE
104

AVAILABILITY
Supported by all drivers.

PARAMETERS
handle specifies a valid workstation handle. flag is set to PERIMETER_OFF (0) to turn off perimeter drawing and PERIMETER_ON (1) to enable it.

BINDING
contrl[0] = 104;
contrl[1] = 0;
contrl[3] = 1;
contrl[6] = handle;

vdi();

RETURN VALUE
This function returns the new value of the perimeter visibility flag.

vsf_style()

WORD vsf_style( handle, style )

WORD handle, style;

vsf_style() defines the style of fill pattern applied to filled objects.

OPCODE
24

AVAILABILITY
Supported by all drivers.

PARAMETERS
handle specifies a valid workstation handle. style specifies the pattern or hatch index depending upon the last setting of vsf_interior(). Valid pattern indexes are
as follows:

| 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 10 | 11 | 12 | 13 | 14 | 15 | 16 | 17 | 18 | 19 | 20 | 21 | 22 | 23 | 24 |
|---|---|---|---|---|---|---|---|---|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|

Valid hatch indexes are as follows:

<table>
<thead>
<tr>
<th>1</th>
<th>2</th>
<th>3</th>
<th>4</th>
<th>5</th>
<th>6</th>
<th>7</th>
<th>8</th>
<th>9</th>
<th>10</th>
<th>11</th>
<th>12</th>
</tr>
</thead>
</table>

**Binding**

```c
contrl[0] = 24;
contrl[1] = 0;
contrl[3] = 1;
contrl[6] = handle;

intin[0] = style;

vdi();
```

**Return Value**

This call returns the actual style set by the call.

**Comments**

The interior type should be set first with `vsf_interior()`.

**See Also**

`vsf_interior()`
vsf_udpat()

VOID vsf_udpat( handle, pattern, planes )
WORD handle;
WORD *planes;
WORD planes;

vsf_udpat() creates the user-defined fill pattern.

OPCODE
112

AVAILABILITY
Supported by all drivers.

PARAMETERS
handle specifies a valid workstation handle. In palette-based modes, pattern points to an array of (16 * planes) WORDs which provide the bit pattern for the fill.

In true-color modes, pattern points to a 16x16 array of LONGs (256 in total) which each contain 32-bit color information. planes specifies the number of color planes for the fill. Use 1 for a monochrome fill on any display, a value equal to the number of planes on the current device for a palette-based color fill or 32 for a true-color display.

BINDING
WORD i;

contrl[0] = 112;
contrl[1] = 0;
contrl[3] = (16 * planes);
contrl[6] = handle;

for(i = 0; i < (16 * planes); i++)
    intin[i] = pattern[i];

vdli();

SEE ALSO
vsf_interior()

vsin_mode()

WORD vsin_mode( handle, device, mode )
WORD handle, device, mode;

vsin_mode() chooses between request or sample mode for the specified device.
vsl_color() – 7.133

**Opcode**

33

**Availability**

Supported in ROM by all Atari computers.

**Parameters**

`handle` specifies a valid workstation handle. A `mode` value of `REQUEST_MODE` (1) sets the device to operate in request mode whereas a value of `SAMPLE_MODE` (2) operates the device in sample mode. Valid devices are:

<table>
<thead>
<tr>
<th>Name</th>
<th>device</th>
<th>Device</th>
</tr>
</thead>
<tbody>
<tr>
<td>LOCATOR</td>
<td>1</td>
<td>Locator</td>
</tr>
<tr>
<td>VALUATOR</td>
<td>2</td>
<td>Valuator</td>
</tr>
<tr>
<td>CHOICE</td>
<td>3</td>
<td>Choice</td>
</tr>
<tr>
<td>STRING</td>
<td>4</td>
<td>String</td>
</tr>
</tbody>
</table>

**Binding**

```c
contrl[0] = 33;
contrl[1] = 0;
contrl[3] = 2;
contrl[6] = handle;

intin[0] = device;
intin[1] = mode;

vdi();

return intout[0];
```

**Return Value**

`vsin_mode()` returns `mode`.

**Comments**

Using this function will cause the AES to function improperly.

**See Also**

`vrq_valuator()`, `vrq_string()`, `vrq_choice()`, `vrq_locator()`, `vsm_valuator()`, `vsm_string()`, `vsm_choice()`, `vsm Locator()`

---

vsl_color()

```c
WORD vsl_color( handle, color )
WORD handle, color;

vsl_color() sets the color for line-drawing functions and objects with perimeters.
```

**Opcode**

17

**Availability**

Supported by all drivers.
PARAMETERS  

handled specifies a valid workstation handle. color specifies the new color to define for line-drawing.

BINDING  

\[
\begin{align*}
\text{contrl}[0] &= 17; \\
\text{contrl}[1] &= 0; \\
\text{contrl}[3] &= 1; \\
\text{contrl}[6] &= \text{handle};
\end{align*}
\]

\[
\begin{align*}
\text{intin}[0] &= \text{color}; \\
vdi(); \\
\text{return intout}[0];
\end{align*}
\]

RETURN VALUE  

This function returns the new color set (within bounds).

SEE ALSO  

vst_color(), vsm_color(), vsf_color()

---

### vsl_ends()

VOID vsl_ends(handle, start, end)

WORD handle, start, end;

vsl_ends() sets the style of end point for the starting and ending points of lines drawn by the VDI in line-drawing functions and perimeter drawing.

OPCODE  

108

AVAILABILITY  

Supported by all drivers.

PARAMETERS  

handle specifies a valid workstation handle. start and end specify the type of end cap to use at the start and end of lines respectively as follows:

<table>
<thead>
<tr>
<th>Name</th>
<th>start/end</th>
<th>Shape</th>
</tr>
</thead>
<tbody>
<tr>
<td>SQUARE</td>
<td>0</td>
<td></td>
</tr>
<tr>
<td>ARROWED</td>
<td>1</td>
<td></td>
</tr>
<tr>
<td>ROUND</td>
<td>2</td>
<td></td>
</tr>
</tbody>
</table>

BINDING  

\[
\begin{align*}
\text{contrl}[0] &= 108; \\
\text{contrl}[1] &= 0; \\
\text{contrl}[3] &= 2; \\
\text{contrl}[6] &= \text{handle};
\end{align*}
\]
intin[0] = start;
intin[1] = end;

vdi();

SEE ALSO vsl_type()

---

vsl_type()

WORD vsl_type( handle, type )
WORD handle, type;

vsl_type() defines the style of line used in line-drawing functions and perimeter drawing.

OPCODE 15

AVAILABILITY Supported by all drivers.

PARAMETERS handle specifies a valid workstation handle. type defines the style of line as follows:

<table>
<thead>
<tr>
<th>Name</th>
<th>type</th>
<th>Style</th>
</tr>
</thead>
<tbody>
<tr>
<td>SOLID</td>
<td>0</td>
<td><img src="image" alt="Solid Line" /></td>
</tr>
<tr>
<td>LDASHED</td>
<td>1</td>
<td><img src="image" alt="LDashed Line" /></td>
</tr>
<tr>
<td>DOTTED</td>
<td>2</td>
<td><img src="image" alt="Dotted Line" /></td>
</tr>
<tr>
<td>DASHDOT</td>
<td>3</td>
<td><img src="image" alt="Dash Dot Line" /></td>
</tr>
<tr>
<td>DASH</td>
<td>4</td>
<td><img src="image" alt="Dashed Line" /></td>
</tr>
<tr>
<td>DASHDOTDOT</td>
<td>5</td>
<td><img src="image" alt="Dash Dot Dot Line" /></td>
</tr>
<tr>
<td>USERLINE</td>
<td>6</td>
<td>User-defined with vsl_udsty().</td>
</tr>
</tbody>
</table>

BINDING

ctrl[0] = 15;
ctrl[1] = 0;
ctrl[3] = 1;
ctrl[6] = handle;
intin[0] = type;

vdi();
return intout[0];

RETURN VALUE

vsl_style() returns the newly set line type.

SEE ALSO

vsl_udsty()

vsl_udsty()

VOID vsl_udsty( handle, pattern )
WORD handle, pattern;

vsl_udsty() sets the user-defined line type.

OPCODE

113

AVAILABILITY

Supported by all drivers.

PARAMETERS

handle specifies a valid workstation handle. pattern is a WORD which defines the USERLINE style. It is essentially a bit mask which is applied to a solid line and repeated along the length of the line. A value of 0xFFFF would create a solid line. A value of 0xAAAA would produce a line where every other pixel was set.

BINDING

contrl[0] = 113;
contrl[1] = 0;
contrl[3] = 1;
contrl[6] = handle;

intin[0] = pattern;

vdi();

COMMENTS

You must call vsl_style( handle, 6 ) to actually utilize this style.

SEE ALSO

vsl_style()
vsl_width()

VOID vsl_width( handle, width )
WORD handle, width;

\textit{vsl_width}() determines the width of lines drawn with line-drawing functions and as perimeters to other objects.

\textbf{OPCODE} \hfill 16

\textbf{AVAILABILITY} \hfill Supported by all drivers.

\textbf{PARAMETERS} \hfill \textit{handle} specifies a valid workstation handle. \textit{width} specifies the width future lines drawn will be.

\textbf{BINDING} \hfill
\begin{verbatim}
contrl[0] = 16;
contrl[1] = 0;
contrl[3] = 1;
contrl[6] = handle;
intin[0] = width;
\end{verbatim}
\begin{verbatim}
vdi();
\end{verbatim}

\textbf{COMMENTS} \hfill The \textit{VDI} is only capable of drawing lines an odd number of pixels thick. Values will be rounded down to the first odd number.

Setting a line width higher than 1 may nullify line styles other than solid. Check \texttt{vq_extnd()} for details.

\textbf{SEE ALSO} \hfill \texttt{vq_extnd()}

vsm_choice()

WORD vsm_choice( handle, xout )
WORD handle;
WORD *xout;

\texttt{vsm_choice()} returns the current value of the ‘choice’ device.

\textbf{OPCODE} \hfill 30

\textbf{AVAILABILITY} \hfill This call is not guaranteed to be available with any driver and its use should therefore be restricted.
PARAMETERS  

.handle specifies a valid workstation handle. .xout points to a .WORD which is filled in on function exit with the current value of the choice device.

BINDING  

```c
contr1[0] = 30;
contr1[1] = contr1[3] = 0;
contr1[6] = handle;

di();
*xout = intout[0];
return contr1[4];
```

RETURN VALUE  

.vsm_choice() returns 1 if an input from the ‘choice’ device was made or 0 otherwise.

SEE ALSO  

.vsin_mode(), .vrq_choice()

---

### vsm_color()

**WORD vsm_color( handle, color )**  
**WORD handle, color;**

.vsm_color() defines the color used to render markers.

OPCODE  

20

AVAILABILITY  

Supported by all drivers.

PARAMETERS  

.handle specifies a valid workstation handle. .color specifies the new color to define for markers.

BINDING  

```c
contr1[0] = 20;
contr1[1] = 0;
contr1[3] = 1;
contr1[6] = handle;

di();

return intout[0];
```

RETURN VALUE  

.vsm_color() returns the new marker color actually set (within bounds).

SEE ALSO  

.v_pmarker(), .vsl_color(), .vst_color(), .vsf_color()
**vsm_height()**

WORD vsm_height( handle, size )
WORD handle, size;

vsm_height() sets the height of markers.

**OPCODE**

19

**AVAILABILITY**

Supported by all drivers.

**PARAMETERS**

handle specifies a valid workstation handle. size specifies the height (and width) of markers to draw in pixels.

**BINDING**

```
contrl[0] = 19;
contrl[1] = 0;
contrl[3] = 1;
contrl[6] = handle;
intin[0] = size;

vdi();
return intout[0];
```

**RETURN VALUE**

vsm_height() returns the marker height actually set.

**COMMENTS**

The DOT marker is not affected by this call. It is always one pixel high and wide.

**SEE ALSO**

v_pmarker()

---

**vsm_locator()**

WORD vsm_locator( handle, mx, my, xout, yout, term )
WORD handle, mx, my;
WORD *xout, *yout, *term;

vsm_locator() receives data from the ‘locator’ device in sample mode.

**OPCODE**

28

**AVAILABILITY**

This call is not guaranteed to be available with any driver and its use should therefore be restricted.

**PARAMETERS**

handle specifies a valid workstation handle. The mouse pointer is initially drawn
at location \((mx, my)\). The call returns with the final position of the mouse in the \texttt{WORD}s pointed to by \texttt{xout} and \texttt{yout}.

The \texttt{WORD} pointed to by \texttt{term} will be filled in with a value which specifies the ASCII value of the key pressed. \texttt{term} will be set to 0x20 if the left mouse button was pressed or 0x21 if the right mouse button was pressed.

\begin{verbatim}
BINDING

contrl[0] = 28;
contrl[1] = 1;
contrl[3] = 0;
contrl[6] = handle;

ptsin[0] = mx;
ptsin[1] = my;

vdi();

*xout = ptsout[0];
*yout = ptsout[1];
*term = intout[0];
return ((contrl[4] << 1) | contrl[2]);
\end{verbatim}

\textbf{RETURN VALUE} \texttt{vsm_locator()} returns one of the following based on its result:

\begin{table}[h]
\begin{tabular}{|c|c|}
\hline
Return Value & Meaning \\
\hline
0 & Mouse has not moved nor was any key pressed. \\
1 & Mouse has been moved (\texttt{xout} and \texttt{yout} are valid). \\
2 & Key or mouse button has been struck (\texttt{term} is valid). \\
3 & Mouse has moved and a key or mouse button has been struck (\texttt{xout}, \texttt{yout}, and \texttt{term} are valid). \\
\hline
\end{tabular}
\end{table}

\textbf{CAVEATS} Using this call will confuse the AES.

\textbf{SEE ALSO} \texttt{vrq_locator()}, \texttt{vsin_mode()}

\begin{verbatim}
\textbf{vsm_string()}

WORD vsm_string( handle, maxlen, echo, echoxy, str )
WORD handle, maxlen, echo;
WORD *echoxy;
char *str;

vsm_string() retrieves input from the 'string' device.
\end{verbatim}

\textbf{OPCODE} 31
The Atari Compendium

Availability
This call is not guaranteed to be available with any driver and its use should therefore be restricted.

Parameters
handle specifies a valid workstation handle. This call inputs characters from the keyboard into the buffer pointed to by str up to (maxlen + 1) characters. If echo is set to 1, characters are echoed to the screen at the location given by the two WORDs pointed to by outxy. If echo is set to 0, no echoing is performed.

Binding

```c
WORD i;

contrl[0] = 31;
contrl[1] = 1;
contrl[3] = 2;
contrl[6] = handle;

intin[0] = maxlen;
intin[1] = echo;

ptsin[0] = echoxy[0];
ptsin[1] = echoxy[1];

vdi();

for(i = 0; i < contrl[4]; i++)
    str[i] = (char)intout[i];

return contrl[4];
```

Return Value
vsm_string() returns the number of characters actually read.

Caveats
Using this function will confuse the AES.

Comments
Though this binding does not allow for it, if maxlen is specified as negative, then as many as (|maxlen| + 1) characters will be read as keycodes rather than ASCII codes. The values in intout will occupy the full WORD rather than just the lower eight bits. A custom binding could be used to take advantage of this.

See Also
vsm_mode()

vsm_type()

WORD vsm_type( handle, type )
WORD handle, type;

vsm_type() sets the current type of marker.

Opcode
18
**Availability**

Supported by all drivers.

**Parameters**

*handle* specifies a valid workstation handle. *type* changes the marker type as follows:

<table>
<thead>
<tr>
<th>Name</th>
<th>type</th>
<th>Shape</th>
</tr>
</thead>
<tbody>
<tr>
<td>MRKR_DOT</td>
<td>1</td>
<td>Single Pixel</td>
</tr>
<tr>
<td>MRKR_PLUS</td>
<td>2</td>
<td>+</td>
</tr>
<tr>
<td>MRKR_ASTERISK</td>
<td>3</td>
<td>*</td>
</tr>
<tr>
<td>MRKR_BOX</td>
<td>4</td>
<td>□</td>
</tr>
<tr>
<td>MRKR_CROSS</td>
<td>5</td>
<td>×</td>
</tr>
<tr>
<td>MRKR_DIAMOND</td>
<td>6</td>
<td>◊</td>
</tr>
<tr>
<td>—</td>
<td>7...</td>
<td>Device Dependent</td>
</tr>
</tbody>
</table>

**Binding**

```c
ctrl[0] = 18;
ctrl[1] = 0;
ctrl[3] = type;
ctrl[6] = handle;

intin[0] = type;

vdi();
```

**Return Value**

*vsm_type()* returns the type of marker actually set.
vsm_valuator() – 7.143

SEE ALSO  v_pmarker()

vsm_valuator()

VOID vsm_valuator( handle, x, xout, term, status )
WORD handle, x;
WORD *xout, *term, *status;

vsm_valuator() retrieves input from the ‘valuator’ device in sample mode.

OPCODE 29

AVAILABILITY This call is not guaranteed to be available with any driver and its use should therefore be restricted.

PARAMETERS handle specifies a valid workstation handle. x sets the initial value of the ‘valuator’. The WORD pointed to by xout is filled in with the final value of the device. If a key was pressed its ASCII code is returned in the WORD pointed to by term. The WORD pointed to by status contains a value as follows:

<table>
<thead>
<tr>
<th>status</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>No input was taken.</td>
</tr>
<tr>
<td>1</td>
<td>Valuator changed.</td>
</tr>
<tr>
<td>2</td>
<td>Key press occurred.</td>
</tr>
</tbody>
</table>

BINDING

contrl[0] = 29;
contrl[1] = 0;
contrl[3] = 1;
contrl[6] = handle;

intin[0] = x;
vdi();

*xout = intout[0];
*term = intout[1];
*status = contrl[4];

SEE ALSO  vsin_mode(), vrq_valuator()
vsp_message()

VOID vsp_message( handle )
WORD handle;

calls the suppression of palette driver messages from the screen.

OPCODE 5
SUB-OPCODE 95

Availability Supported by all camera drivers.

Parameters handle specifies a valid workstation handle.

Binding
contrl[0] = 5;
contrl[5] = 95;
contrl[6] = handle;

vsp_save()

VOID vsp_save( handle )
WORD handle;

 calls the suppression of palette driver messages from the screen.

OPCODE 5
SUB-OPCODE 94

Availability Supported by all camera drivers.

Parameters handle specifies a valid workstation handle.

Binding
contrl[0] = 5;
contrl[5] = 94;
contrl[6] = handle;

vsp_save() saves the current state of the driver to disk.
vsp_state()

VOID vsp_state( handle, port, film, lightness, interlace, planes, indexes )
WORD handle, port, film, lightness, interlace, planes;
WORD *indexes;

vsp_state() sets the palette driver state.

OPCODE 5
SUB-OPCODE 93
AVAILABILITY Supported by all camera drivers.

PARAMETERS handle specifies a valid workstation handle. port specifies the communication port number of the camera device. film specifies the index of the desired type of film (0–4).

lightness specifies the modification to apply to the camera’s default f-stop setting (-3–3). A value of 0 uses the default setting. A value of -3 results in an exposure of half of the default length whereas a value of 3 doubles the exposure time. interlace is set to 0 for non-interlaced or 1 for interlaced output.

planes specifies the number of planes to output (1–4). indexes points to an array of 16 WORDs which define the color codes for the palette.

BINDING

WORD i;

contrl[0] = 5;
contrl[1] = 0;
contrl[3] = 20;
contrl[5] = 93;
contrl[6] = handle;

intin[0] = port;
intin[1] = film;
intin[2] = lightness;
intin[3] = interlace;
for(i = 0;i < 16;i++)
    intin[i + 5] = indexes[i];

vdif();

SEE ALSO vqp_state()
**vst_alignment()**

VOID vst_alignment( handle, halign, valign, *hout, *vout )
WORD handle, halign, valign;
WORD *hout, *vout;

**vst_alignment()** affects the vertical and horizontal alignment of normal and justified text.

**OPCODE**
39

**AVAILABILITY**
Supported by all drivers.

**PARAMETERS**
*handle* specifies a valid workstation handle. *halign* and *valign* affects where the coordinate specified by *v_gtext()* or *v_justified()* actually applies to as follows:

<table>
<thead>
<tr>
<th>valign:</th>
<th>Top (5)</th>
<th>Ascent Line (2)</th>
<th>Half Line (1)</th>
<th>Base Line (0)</th>
<th>Descent (4)</th>
<th>Bottom (3)</th>
</tr>
</thead>
<tbody>
<tr>
<td>halign:</td>
<td>Left Justified (0)</td>
<td>Center Justified (1)</td>
<td>Right Justified (2)</td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

On return, the **WORDS** pointed to by *hout* and *vout* are filled in with the values actually set.

**BINDING**

```
ctrl[0] = 39;
ctrl[1] = 0;
ctrl[3] = 2;
ctrl[6] = handle;

intin[0] = halign;
intin[1] = valign;

vdi();

*hout = intout[0];
*vout = intout[1];
```

**SEE ALSO**
*v_gtext(), v_justified()*
vst_arbpt()

WORD vst_arbpt( handle, point, wchar, hchar, wcell, hcell )
WORD handle;
WORD point;
WORD *wchar, *hchar, *wcell, *hcell;

vst_arbpt() selects any point size for an outline font.

OPCODE
246

AVAILABILITY
Available only with FSMGDOS or SpeedoGDOS.

PARAMETERS
handle specifies a valid workstation handle. point specifies the point size at which to render outline text.

Upon return, the WORDs pointed to by wchar, hchar, wcell, and hcell will be filled in with the width and height of the character and the width and height of the character cell respectively.

BINDING
contrl[0] = 246;
contrl[1] = 0;
contrl[3] = 1;
contrl[6] = handle;

intin[0] = point;

vdi();

*wchar = ptsout[0];
*hchar = ptsout[1];
*wcell = ptsout[2];
*hcell = ptsout[3];

return intout[0];

RETURN VALUE
vst_arbpt() returns the point size actually selected.

COMMENTS
This call only works with outline fonts, however, it is not restricted by the point sizes listed in the ‘ASSIGN.SYS’ file.

To specify a fractional point size, use vst_arbpt32().

SEE ALSO
vst_arbpt32(), vst_point(), vst_height()
vst_arbpt32()

fix31 vst_arbpt( handle, point, wchar, hchar, wcell, hcell )
WORD handle;
fix31 point;
WORD *wchar, *hchar, *wcell, *hcell;

vst_arbpt32() selects a fractional point size for an outline font.

OPCODE
246

AVAILABILITY
Available only with FSMGDOS or SpeedoGDOS.

PARAMETERS
handle specifies a valid workstation handle. point specifies the point size at which to render outline text as a fix31 value.
Upon return, the WORDs pointed to by wchar, hchar, wcell, and hcell will be filled in with the width and height of the character and the width and height of the character cell respectively.

BINDING
contr1[0] = 246;
contr1[1] = 0;
contr1[3] = 2;
contr1[6] = handle;
intin[0] = (WORD)(point >> 16);
intin[1] = (WORD)(point & 0xFFFF);

vdi();

*wchar = ptsout[0];
*hchar = ptsout[1];
*wcell = ptsout[2];
*hcell = ptsout[3];

return ((fix31)intout[0] << 16) | (fix31)intout[1];

RETURN VALUE
vst_arbpt32() returns the point size actually selected.

COMMENTS
This call only works with outline fonts, however, it is not restricted by the point sizes listed in the ‘ASSIGN.SYS’ file.

SEE ALSO
vst_arbpt(), vst_point(), vst_height()
vst_charmap()

VOID vst_charmap( handle, mode )
WORD handle, mode;

vst_charmap() chooses between the standard Atari ASCII interpretation of text strings or translation of Bitstream character indexes.

Opcode

236

Availability

Available only with SpeedoGDOS.

Parameters

handle specifies a valid workstation handle. mode should be MAP_ATARI (1) to specify Atari ASCII characters or MAP_BITSTREAM (0) for Bitstream mappings.

Binding

contrl[0] = 236;
contrl[1] = 0;
contrl[3] = 1;
contrl[6] = handle;
intin[0] = mode;
vdli();

Comments

Bitstream character indexes are WORD sized rather than BYTE sized. A list of Bitstream character mappings can be found in Appendix G.

vst_color()

WORD vst_color( handle, color )
WORD handle, color;

vst_color() sets the current text color.

Opcode

22

Availability

Supported by all drivers.

Parameters

handle specifies a valid workstation handle. color specifies the new color to apply to text.

Binding

contrl[0] = 22;
contrl[1] = 0;
contrl[3] = 1;
contrl[6] = handle;
intin[0] = color;

vdi();

return intout[0];

**RETURN VALUE**

`vst_color()` returns the text color actually set (within bounds).

**SEE ALSO**

`vsl_color()`, `vsm_color()`, `vsf_color()`

---

**vst_effects()**

```c
WORD vst_effects( handle, effects )
WORD handle, effects;
```

`vst_effects()` defines which special effects are to be applied to text.

**OPCODE**

106

**AVAILABILITY**

Supported by all drivers.

**PARAMETERS**

`handle` specifies a valid workstation handle. `effects` is a bit mask which specifies one or more special effects to apply to text as follows:

<table>
<thead>
<tr>
<th>Name</th>
<th>Bit</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>THICKENED</td>
<td>0</td>
<td>Thickened</td>
</tr>
<tr>
<td>LIGHT</td>
<td>1</td>
<td>Lightened</td>
</tr>
<tr>
<td>SKEWED</td>
<td>2</td>
<td>Skewed</td>
</tr>
<tr>
<td>UNDERLINED</td>
<td>3</td>
<td>Underlined</td>
</tr>
<tr>
<td>OUTLINED</td>
<td>4</td>
<td>Outlined</td>
</tr>
<tr>
<td>SHADOWED</td>
<td>5</td>
<td>Shadowed (not currently supported)</td>
</tr>
</tbody>
</table>

**BINDING**

```c
contr1[0] = 106;
contr1[1] = 0;
contr1[3] = 1;
contr1[6] = handle;

intin[0] = effects;

vdi();

return intout[0];
```

**RETURN VALUE**

`vst_effects()` returns the actual effects set by the call.

**COMMENTS**

Special effects do not, in general, work well with outline text (besides...
underlining). To compensate, most type families have bold and italic faces in addition to the vst_skew() call.

SEE ALSO vst_skew()

---

**vst_error()**

VOID vst_error( handle, mode, error )

WORD handle, mode;

WORD *error;

vst_error() provides a method to obtain errors from GDOS and suppress text messages on screen.

**OPCODE**

245

**AVAILABILITY**

Available only with FONTGDOS, FSM, or SpeedoGDOS.

**PARAMETERS**

*handle* specifies a valid workstation handle. *mode* specifies the error reporting mode. A value of SCREEN_ERROR (1) (default) causes error messages to be outputted to the screen as text.

A value of APP_ERROR (0) suppresses these messages and instead places an error code in the WORD pointed to by *error* whenever an error occurs leaving it up to the application to process errors correctly. Prior to making this call and after each reported error, the application is responsible for resetting the value pointed to by *error* to 0. The following is a list of possible error codes:

<table>
<thead>
<tr>
<th>Name</th>
<th>error</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>NO_ERROR</td>
<td>0</td>
<td>No error.</td>
</tr>
<tr>
<td>CHAR_NOT_FOUND</td>
<td>1</td>
<td>Character not found in font.</td>
</tr>
<tr>
<td>FILE_READERR</td>
<td>8</td>
<td>Error reading file.</td>
</tr>
<tr>
<td>FILE_OPENERR</td>
<td>9</td>
<td>Error opening file.</td>
</tr>
<tr>
<td>BAD_FORMAT</td>
<td>10</td>
<td>Bad file format.</td>
</tr>
<tr>
<td>CACHE_FULL</td>
<td>11</td>
<td>Out of memory/cache full.</td>
</tr>
<tr>
<td>MISC_ERROR</td>
<td>-1</td>
<td>Miscellaneous error.</td>
</tr>
</tbody>
</table>

**Binding**

```
contr1[0] = 245;
contr1[1] = 0;
contr1[3] = 3;
contr1[6] = handle;

intin[0] = mode;
*(LONG *)&intin[1] = (LONG)error;
```
Once setting the error mode to 0, an application should check the error variable after each of the following calls:

- `v_gtext()`
- `v_justified()`
- `vst_point()`
- `vst_height()`
- `vst_font()`
- `vst_arbpt()`
- `vqt_advance()`
- `vst_setsize()`
- `vqt_fontinfo()`
- `vqt_name()`
- `vqt_width()`
- `vqt_extent()`
- `v_opnwk()`
- `v_opnvwk()`
- `vst_load_fonts()`
- `vst_unload_fonts()`
- `v_ftext()`
- `vqt_f_extent()`

**vst_font()**

```c
WORD vst_font(
    handle, index
)

WORD handle, index;
```

**vst_font()** sets the current text font.

**OPCODE**

21

**AVAILABILITY**

Supported by all drivers.

**PARAMETERS**

- `handle` specifies a valid workstation handle.
- `index` specifies the index (as returned by `vqt_name()`) of the font to enable.

**BINDING**

```c
contrl[0] = 21;
contrl[1] = 0;
contrl[3] = 1;
contrl[6] = handle;

intin[0] = index;

vdi();

return intout[0];
```

**RETURN VALUE**

`vst_font()` returns the index of the font actually set.

**SEE ALSO**

`vqt_name()`
vst_height()

VOID vst_height( handle, height, wchar, hchar, wcell, hcell )
WORD handle, height;
WORD *wchar, *hchar, *wcell, *hcell;

vst_height() sets the height of the current text face (in pixels).

O P C O D E

12

A V A I L A B I L I T Y

Supported by all drivers.

P A R A M E T E R S

handle specifies a valid workstation handle. height specifies the height (in pixels) at which to render text. Upon return, the WORDs pointed to by wchar, hchar, wcell, and hcell will be filled in with the width and height of the character and the width and height of the character cell respectively.

B I N D I N G

ctrl[0] = 12;
ctrl[1] = 1;
ctrl[3] = 0;
ctrl[6] = handle;

ptsin[0] = 0;
ptsin[1] = height; /* Passed in ptsin[1] because of VDI bug. */

vdi();

*wchar = ptsout[0];
*hchar = ptsout[1];
*wcell = ptsout[2];
*hcell = ptsout[3];

C O M M E N T S

vst_height() works on both bitmap and outline fonts. The font will be scaled to fit within the height given. This doesn’t always give good results with bitmap text.

S E E A L S O

vst_point(), vst_arbpt()

vst_kern()

VOID vst_kern( handle, tmode, pmode, tracks, pairs )
WORD handle, tmode, pmode;
WORD *tracks, *pairs;

vst_kern() sets the track and pair kerning values.
OPCODE 237

AVAILABILITY Available only with SpeedoGDOS.

PARAMETERS \( \text{handle} \) specifies a valid workstation handle. \( \text{tmode} \) specifies the track kerning mode as follows:

<table>
<thead>
<tr>
<th>Name</th>
<th>( \text{tmode} )</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>TRACK_NONE</td>
<td>0</td>
<td>No track kerning</td>
</tr>
<tr>
<td>TRACK_NORMAL</td>
<td>1</td>
<td>Normal track kerning</td>
</tr>
<tr>
<td>TRACK_TIGHT</td>
<td>2</td>
<td>Tight track kerning</td>
</tr>
<tr>
<td>TRACK_VERYTIGHT</td>
<td>3</td>
<td>Very tight track kerning</td>
</tr>
</tbody>
</table>

Setting \( \text{pmode} \) to \text{PAIR_ON} (1) turns pair kerning on. Setting it to \text{PAIR_OFF} (0) turns pair kerning off.

The \text{WORD} pointed to by \text{tracks} is filled in with the track kerning mode actually set. \text{pairs} points to a \text{WORD} which is filled in with the number of defined character kerning pairs.

BINDING

```c
contrl[0] = 237;
contrl[1] = 0;
contrl[3] = 2;
contrl[6] = handle;

intin[0] = tmode;
intin[1] = pmode;

vdi();

*tracks = intout[0];
*pairs = intout[1];
```

SEE ALSO \text{vqt_trackkern()}, \text{vqt_pairkern()}

---

**vst_load_fonts()**

\text{WORD vst_load_fonts( handle, rsrvd )}

\text{WORD handle, rsrvd;}

\text{vst_load_fonts()} loads disk-based font information into memory.

OPCODE 119

AVAILABILITY Available with any form of GDOS.
**vst_point()**

WORD vst_point( handle, point, wchar, hchar, wcell, hcell )
WORD handle, height;
WORD *wchar, *hchar, *wcell, *hcell;

vst_point() sets the height of the current text face in points (1/72 inch).

**OPCODE**

107

**AVAILABILITY**

Supported by all drivers.

**PARAMETERS**

*handle* specifies a valid workstation handle. *point* specifies a valid point size to set the current text face to. This means an appropriate bitmap font or a point size enumerated in the ‘EXTEND.SYS’ file.

Upon return, the WORDs pointed to by *wchar*, *hchar*, *wcell*, and *hcell* will be filled in with the width and height of the character and the width and height of the character cell respectively.

**BINDING**

contrl[0] = 107;
contrl[1] = 0;
contrl[3] = 1;
contrl[6] = handle;
intin[0] = point;

vdi();
*wchar = ptsout[0];
*hchar = ptsout[1];
*wcell = ptsout[2];
*hcell = ptsout[3];

return intout[0];

**RETURN VALUE**  
*vst_point()* returns the point size actually set.

**COMMENTS**  
If a point size which doesn’t exist for the current face is selected, the next valid size down is selected.

**SEE ALSO**  
vst_arbpt(), vst_height()

---

**vst_rotation()**

WORD vst_rotation( handle, angle )  
WORD handle, angle;

*vst_rotation()* sets the angle at which graphic text is drawn.

**OPCODE**  
13

**AVAILABILITY**  
Supported by all drivers. For specific character rotation abilities, check the values returned in *vq_extnd()*.

**PARAMETERS**  
*handle* specifies a valid workstation handle. *angle* specifies the angle at which to rotate text in tenths of degrees as follows:

```
0       900
1800     2700
```

**BINDING**  
```c
contrl[0] = 13;
contrl[1] = 0;
contrl[3] = 1;
contrl[6] = handle;

intin[0] = angle;

vdi();

return intout[0];
```
**Return Value**  
`vst_rotation()` returns the value of rotation actually set.

**Comments**  
Bitmap fonts may only be rotated at 0, 90, and 270 degrees. Outline fonts may be rotated at any angle with **FSM**.

---

### vst_scratch()

**VOID vst_scratch( handle, mode )**

**WORD handle, mode;**

`vst_scratch()` allows **FSMGDOS** or **SpeedoGDOS** to change its method of allocating a scratch buffer for better efficiency.

**Opcode**  
244

**Availability**  
Available only with **FSMGDOS** or **SpeedoGDOS**.

**Parameters**  
`handle` specifies a valid workstation handle. `mode` specifies the scratch buffer allocation mode as follows:

<table>
<thead>
<tr>
<th>Name</th>
<th>mode</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>SCRATCH_BOTH</td>
<td>0</td>
<td>Scratch buffers should be allocated which are large enough for <strong>FSM/Speedo</strong> and bitmap fonts with any combination of special effects.</td>
</tr>
<tr>
<td>SCRATCH_BITMAP</td>
<td>1</td>
<td>Scratch buffers should be allocated which are large enough for <strong>FSM/Speedo</strong> fonts with no effects and bitmap fonts with effects.</td>
</tr>
<tr>
<td>SCRATCH_NONE</td>
<td>2</td>
<td>Scratch buffers should be allocated which are large enough for <strong>FSM/Speedo</strong> fonts and bitmap fonts with no special effects.</td>
</tr>
</tbody>
</table>

**Binding**

```c
contrl[0] = 244;
contrl[1] = 0;
contrl[3] = 1;
contrl[6] = handle;
intin[0] = mode;
vdli();
```

**Comments**  
Atari recommends that at least mode 1 be set prior to a `vst_load_fonts()` call to prevent scratch buffer overruns.

The size of the scratch buffer is based on the size of the largest point size specified in the ‘EXTEND.SYS’ file. Attempting to add effects to a character higher in point size than this will cause a buffer overrun.
vst_setsize()

WORD vst_setsize( handle, point, wchar, hchar, wcell, hcell )
WORD handle;
WORD point;
WORD * wchar, * hchar, * wcell, * hcell;

vst_setsize() sets the width of outline characters.

OPCODE
252

AVAILABILITY
Available only with FSMGDOS or SpeedoGDOS.

PARAMETERS
handle specifies a valid workstation handle.

point specifies the width of the character in points (1/72 inch). A value for point equivalent to the same point size specified in vst_arbpt() will result in a correctly proportioned character.

Upon return, the WORDs pointed to by wchar, hchar, wcell, and hcell will be filled in with the width and height of the character and the width and height of the character cell respectively.

BINDING

```
contrl[0] = 252;
contrl[1] = 0;
contrl[3] = 1;
contrl[6] = handle;

intin[0] = point;

vdi();

*wchar = ptsout[0];
*hchar = ptsout[1];
*wcell = ptsout[2];
*hcell = ptsout[3];

return intout[0];
```

RETURN VALUE

vst_setsize() returns the size actually set.

COMMENTS

This call only works with outline fonts. At the next vst_point(), vst_height(), or vst_arbpt() the size will be reset to the correct proportions (width in points = height in points).

To set a fractional size, use vst_setsize32().

THE ATARI COMPENDIUM
vst_setsize32() sets the width of outline characters as a fix31 fractional value.

**OPCODE**

252

**AVAILABILITY**

Available only with SpeedoGDOS.

**PARAMETERS**

*handle* specifies a valid workstation handle.

*point* specifies the width of the character in points (1/72 inch). A value for *point* equivalent to the same point size specified in *vst_arbpt()* will result in a correctly proportioned character.

Upon return, the WORDs pointed to by *wchar*, *hchar*, *wcell*, and *hcell* will be filled in with the width and height of the character and the width and height of the character cell respectively.

**BINDING**

```c
contrl[0] = 252;
contrl[1] = 0;
contrl[3] = 2;
contrl[6] = handle;

intin[0] = (WORD)(point >> 8);
intin[1] = (WORD)point;

vdi();

*wchar = ptsout[0];
*hchar = ptsout[1];
*wcell = ptsout[2];
*hcell = ptsout[3];

return ((fix31)intout[0] << 16) | (fix31)intout[1];
```

**RETURN VALUE**

vst_setsize32() returns the size actually set.

**COMMENTS**

This call only works with outline fonts. At the next vst_point(), vst_height(), or vst_arbpt() the size will be reset to the correct proportions (width in points = height in points).
vst_skew()

WORD vst_skew( handle, skew )

vst_skew() sets the skew amount for fonts.

OPCODE 253

AVAILABILITY Available only with FSMGDOS or SpeedoGDOS.

PARAMETERS handle specifies a valid workstation handle. skew specifies the amount to skew in tenths of degrees from -900 to 900. Negative values skew to the left and positive values skew to the right. skew values of -900 or 900 will result in a flat line.

BINDING

cntl[0] = 253;
cntl[1] = 0;
cntl[3] = 1;
cntl[6] = handle;

intin[0] = skew;

vdi();

return intout[0];

RETURN VALUE vst_skew() returns the skew value actually set.

COMMENTS This call should only be used with outline fonts. Note that this call generates a true ‘skew’ effect independent of that generated by vst_effects() which is an algorithmic ‘skew’. The algorithmic ‘skew’ may be used on bitmap fonts but is rather unpleasant applied to outline fonts.

SEE ALSO vst_effects()
**vswr_mode()**

WORD vswr_mode(handle, mode)

WORD handle, mode;

vswr_mode() defines the writing mode for rendering VDI objects.

**Opcode**

32

**Availability**

Supported by all drivers.

**Parameters**

handle specifies a valid workstation handle. mode specifies a writing mode as follows:

<table>
<thead>
<tr>
<th>Name</th>
<th>mode</th>
<th>Example</th>
</tr>
</thead>
<tbody>
<tr>
<td>MD_REPLACE</td>
<td>1</td>
<td><img src="image1.png" alt="Example" /></td>
</tr>
<tr>
<td></td>
<td></td>
<td><img src="image2.png" alt="Example" /></td>
</tr>
<tr>
<td>MD_TRANS</td>
<td>2</td>
<td><img src="image3.png" alt="Example" /></td>
</tr>
<tr>
<td></td>
<td></td>
<td><img src="image4.png" alt="Example" /></td>
</tr>
</tbody>
</table>
**vt_alignment()**

VOID vt_alignment(handle, dx, dy)

WORD handle, dx, dy;

vt_alignment() allows an offset to be specified that will be applied to all coordinates output from the graphics tablet.

**OPC ODE**

5

**SUB-O P C O D E**

85

**A VAILABILITY**

Supported by all tablet drivers.

**PARAMETERS**

handle specifies a valid workstation handle. dx and dy are the delta offsets from

---

**BINDING**

contrl[0] = 32;
contrl[1] = 0;
contrl[3] = 1;
contrl[6] = handle;

intin[0] = mode;

vdi();

return intout[0];

**RE TUR N V ALUE**

vswr_mode() returns the writing mode set.

**COMMENTS**

In true-color modes, MD_ERASE and MD_TRANS work a little differently, they write (or avoid writing on) whatever color is currently held in VDI color 0 (as opposed to the actual register reference of 0).
vt_axis() – 7.163

( 0, 0 ) to apply to values from the graphics tablet.

**BINDING**

```plaintext
contrl[0] = 5;
contrl[1] = 0;
contrl[3] = 2;
contrl[5] = 85;
contrl[6] = handle;
```

```plaintext
intin[0] = dx;
intin[1] = dy;
```

```plaintext
vdi();
```

**COMMENTS**

This call is used to ‘fine-tune’ the true starting point of the tablet.

**SEE ALSO**

vt_origin()

---

**vt_axis()**

**VOID vt_axis( handle, xres, yres, *xout, *yout )**

**WORD handle, xres, yres;**

**WORD *xout, *yout;**

vt_axis() sets the horizontal and vertical resolution for the graphics tablet (in lines).

**OPCODE**

5

**SUB-OPCODE**

82

**AVAILABILITY**

Supported by all tablet drivers.

**PARAMETERS**

handle specifies a valid workstation handle. xres and yres specify the new horizontal and vertical resolution of the tablet respectively. Upon return, the WORDs pointer to by xout and yout are filled in with the resolution actually set.

**BINDING**

```plaintext
contrl[0] = 5;
contrl[1] = 0;
contrl[3] = 2;
contrl[5] = 82;
contrl[6] = handle;
```

```plaintext
intin[0] = xres;
intin[1] = yres;
```

```plaintext
vdi();
```

```plaintext
*xout = intout[0];
*yout = intout[1];
```
SEE ALSO  vt_alignment(), vt_origin()

---

**vt_origin()**

VOID vt_origin( handle, xorigin, yorigin )
WORD handle, xorigin, yorigin;

vt_origin() sets the origin point for the tablets’ upper-left point.

**OPCODE**  5

**SUB-OPCODE**  83

**AVAILABILITY**  Supported by all tablet drivers.

**PARAMETERS**  handle specifies a valid workstation handle. xorigin and yorigin specify the new upper-left point recognized by the tablet.

**BINDING**

```c
contrl[0] = 5;
contrl[1] = 0;
contrl[3] = 2;
contrl[5] = 83;
contrl[6] = handle;

intin[0] = xorigin;
intin[1] = yorigin;

vt_origin();
```

SEE ALSO  vt_axis(), vt_alignment()

---

**vt_resolution()**

VOID vt_resolution( handle, xres, yres, *xout, *yout )
WORD xres, yres;
WORD *xout, *yout;

vt_resolution() sets the horizontal and vertical resolution of the graphics tablet (in lines per inch).

**OPCODE**  5

**SUB-OPCODE**  81

---

THE ATARI COMPENDIUM
**Availability**

Supported by all tablet drivers.

**Parameters**

`handle` specifies a valid workstation handle. `xres` and `yres` specify the new horizontal and vertical resolution values for the tablet respectively. Upon return, the **words** pointed to by `xout` and `yout` are filled in with the values actually set.

**Binding**

```c
consr[0] = 5;
consr[1] = 0;
consr[3] = 2;
consr[5] = 81;
consr[6] = handle;

intin[0] = xres;
intin[1] = yres;

vdli();

*xout = intout[0];
*yout = intout[1];
```

**See Also**

`vt_axis()`
Overview

The **Line-A** portion of the operating system is so named because it uses a special exception vector of 680x0 processors triggered when the first nibble of the a command word is $A. On Atari systems this vector is routed to the operating system ROMs and provides a low-level yet high-speed graphics interface.

The **Line-A** system is included in this document for completeness only. It is recommended that its use be avoided and that the counterpart **VDI** calls be used instead. Atari has not guaranteed that it will maintain **Line-A** compatibility in future systems. Its functionality has already been limited as video capabilities have advanced beyond its design.

The Line-A Variable Table

The **Line-A** opcode $A000 will return a pointer to an internal variable table in D0 and A0. This table is used by the **Line-A** functions as a parameter passing mechanism as opposed to using the stack or internal registers.

Members of the **Line-A** variable table are accessed via offsets from the base address. The function, location, and size of documented variables are as follows:

<table>
<thead>
<tr>
<th>Name</th>
<th>Offset</th>
<th>Size</th>
<th>Contents</th>
</tr>
</thead>
<tbody>
<tr>
<td>RESERVED</td>
<td>-910</td>
<td>LONG</td>
<td>Reserved for future use.</td>
</tr>
<tr>
<td>CUR_FONT</td>
<td>-906</td>
<td>LONG</td>
<td>Pointer to the current font header.</td>
</tr>
<tr>
<td>RESERVED</td>
<td>-902</td>
<td>92 BYTEs</td>
<td>Reserved for future use.</td>
</tr>
<tr>
<td>M_POS_HX</td>
<td>-856</td>
<td>WORD</td>
<td>X Offset into the mouse form of the 'hot spot'.</td>
</tr>
<tr>
<td>M_POS_HY</td>
<td>-854</td>
<td>WORD</td>
<td>Y Offset into the mouse form of the 'hot spot'.</td>
</tr>
<tr>
<td>M_PLANES</td>
<td>-852</td>
<td>WORD</td>
<td>Writing mode for the mouse pointer (1 = <strong>VDI</strong> Mode, -1 = XOR Mode). Defaults to <strong>VDI</strong> mode.</td>
</tr>
<tr>
<td>M_CDB_BG</td>
<td>-850</td>
<td>WORD</td>
<td>Mouse pointer background color.</td>
</tr>
<tr>
<td>M_CDB_FG</td>
<td>-848</td>
<td>WORD</td>
<td>Mouse pointer foreground color.</td>
</tr>
<tr>
<td>MASK_FORM</td>
<td>-846</td>
<td>32 WORDs</td>
<td>Image and Mask for the mouse pointer. Data is stored in the following format:</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>Line 0 Mask</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>Line 0 Image</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>Line 1 Mask</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>Line 1 Image</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>etc.</td>
</tr>
<tr>
<td>INQ_TAB</td>
<td>-782</td>
<td>46 WORDs</td>
<td>This area contains 45 WORDs of information returned from a <strong>vq_extnd()</strong> of the physical screen workstation plus one extra reserved WORD.</td>
</tr>
<tr>
<td>DEV_TAB</td>
<td>-692</td>
<td>46 WORDs</td>
<td>This area contains the first 45 WORDs of information returned from a <strong>v_opnw()</strong> of the physical screen workstation plus one extra reserved WORD.</td>
</tr>
<tr>
<td>GCURX</td>
<td>-602</td>
<td>WORD</td>
<td>Current mouse pointer X position.</td>
</tr>
<tr>
<td>GCURY</td>
<td>-600</td>
<td>WORD</td>
<td>Current mouse pointer Y position.</td>
</tr>
<tr>
<td>Variable</td>
<td>Offset</td>
<td>Type</td>
<td>Description</td>
</tr>
<tr>
<td>------------------</td>
<td>--------</td>
<td>--------</td>
<td>--------------------------------------------------------------------------------------------------</td>
</tr>
<tr>
<td>M_HID_CT</td>
<td>-598</td>
<td>WORD</td>
<td>Current mouse ‘hide’ count (number of times mouse has been hidden, 0 = visible).</td>
</tr>
<tr>
<td>MOUSE_BT</td>
<td>-596</td>
<td>WORD</td>
<td>Bitmap of the current mouse button status.</td>
</tr>
<tr>
<td>REQ_COL</td>
<td>-594</td>
<td>48 WORDs</td>
<td>Contains 48 WORDs of RGB data for the first 16 VDI color registers as would be returned by <code>vq_color()</code>.</td>
</tr>
<tr>
<td>SIZ_TAB</td>
<td>-498</td>
<td>15 WORDs</td>
<td>This table contains the final 12 WORDs of information returned from a <code>v_opnwk()</code> of the physical screen workstation plus 3 reserved WORDs.</td>
</tr>
<tr>
<td>RESERVED</td>
<td>-468</td>
<td>WORD</td>
<td>Reserved for future use.</td>
</tr>
<tr>
<td>RESERVED</td>
<td>-466</td>
<td>WORD</td>
<td>Reserved for future use.</td>
</tr>
<tr>
<td>CUR_WORK</td>
<td>-464</td>
<td>LONG</td>
<td>Pointer to the current VDI workstation attribute table.</td>
</tr>
<tr>
<td>DEF_FONT</td>
<td>-460</td>
<td>LONG</td>
<td>Pointer to the default font header.</td>
</tr>
<tr>
<td>FONT_RING</td>
<td>-456</td>
<td>4 LONGs</td>
<td>This area contains three pointers and a NULL. The first two pointers point to linked lists of system font headers. The third pointer points to the linked list of GDOS based fonts.</td>
</tr>
<tr>
<td>FONT_COUNT</td>
<td>-440</td>
<td>WORD</td>
<td>Total number of fonts pointed to by the FONT_RING pointers.</td>
</tr>
<tr>
<td>RESERVED</td>
<td>-438</td>
<td>90 BYTEs</td>
<td>Reserved for future use.</td>
</tr>
<tr>
<td>CUR_MS_STAT</td>
<td>-348</td>
<td>BYTE</td>
<td>Bitmap of mouse status since the last interrupt as follows:</td>
</tr>
<tr>
<td>RESERV GED</td>
<td>-347</td>
<td>BYTE</td>
<td>Reserved for future use.</td>
</tr>
<tr>
<td>V_HID_CNT</td>
<td>-346</td>
<td>WORD</td>
<td>Number of times the text cursor has been hidden (0 = visible).</td>
</tr>
<tr>
<td>CUR_X</td>
<td>-344</td>
<td>WORD</td>
<td>X position where mouse pointer will be drawn.</td>
</tr>
<tr>
<td>CUR_Y</td>
<td>-342</td>
<td>WORD</td>
<td>Y position where mouse pointer will be drawn.</td>
</tr>
<tr>
<td>CUR_FLAG</td>
<td>-340</td>
<td>BYTE</td>
<td>Mouse redraw flag (if non-zero, mouse pointer will be redrawn at the next vertical blank interrupt).</td>
</tr>
<tr>
<td>MOUSE_FLAG</td>
<td>-339</td>
<td>BYTE</td>
<td>Mouse interrupt flag (0=disable interrupts)</td>
</tr>
<tr>
<td>RESERVED</td>
<td>-338</td>
<td>LONG</td>
<td>Reserved for future use.</td>
</tr>
<tr>
<td>V_SAV_XY</td>
<td>-334</td>
<td>2 WORDs</td>
<td>X and Y position of the text cursor as saved by the VT-52 emulator.</td>
</tr>
<tr>
<td>SAVE_LEN</td>
<td>-330</td>
<td>WORD</td>
<td>Height of the form saved in SAVE_AREA in pixels.</td>
</tr>
<tr>
<td>SAVE_ADDR</td>
<td>-328</td>
<td>LONG</td>
<td>Address of the first WORD of screen data contained in SAVE_AREA.</td>
</tr>
<tr>
<td>SAVE_STAT</td>
<td>-324</td>
<td>LONG</td>
<td>Save status flag as follows:</td>
</tr>
<tr>
<td>SAVE_AREA</td>
<td>-322</td>
<td>256 BYTEs</td>
<td>Save buffer for the mouse pointer.</td>
</tr>
</tbody>
</table>

**Bit** | **Meaning**  
--- | ---  
0 | Left mouse status (0=up)  
1 | Right mouse status (0=up)  
2 | Reserved  
3 | Reserved  
4 | Reserved  
5 | Mouse move flag (1=moved)  
6 | Right mouse status flag (0=hasn’t changed)  
7 | Left mouse status flag (0=hasn’t changed)  
0 | Save buffer valid? (0=no)  
1 | Width of save (0=16 bits, 1=32 bits)
<table>
<thead>
<tr>
<th>Variable</th>
<th>Offset</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>USER_TIM</td>
<td>-66</td>
<td>LONG</td>
<td>Pointer to a routine which occurs at each timer tick. (use <code>vex_timv()</code>) instead. Routine ends by jumping to function pointed to by NEXT_TIM.</td>
</tr>
<tr>
<td>NEXT_TIM</td>
<td>-62</td>
<td>LONG</td>
<td>See above.</td>
</tr>
<tr>
<td>USER_BUT</td>
<td>-58</td>
<td>LONG</td>
<td>Pointer to a routine called each time a mouse button is pressed (use <code>vex_butv()</code>) instead.</td>
</tr>
<tr>
<td>USER_CUR</td>
<td>-54</td>
<td>LONG</td>
<td>Pointer to a routine called each time the mouse needs to be rendered (use <code>vex_curv()</code>) instead.</td>
</tr>
<tr>
<td>USER_MOT</td>
<td>-50</td>
<td>LONG</td>
<td>Pointer to routine called each time the mouse is moved (use <code>vex_motv()</code>) instead.</td>
</tr>
<tr>
<td>V_CEL_HT</td>
<td>-46</td>
<td>WORD</td>
<td>Current text cell height.</td>
</tr>
<tr>
<td>V_CEL_MX</td>
<td>-44</td>
<td>WORD</td>
<td>Number of text columns – 1.</td>
</tr>
<tr>
<td>V_CEL_MY</td>
<td>-42</td>
<td>WORD</td>
<td>Number of text rows – 1.</td>
</tr>
<tr>
<td>V_CEL_WR</td>
<td>-40</td>
<td>WORD</td>
<td>Number of bytes between character cells.</td>
</tr>
<tr>
<td>V_CEL_BG</td>
<td>-38</td>
<td>WORD</td>
<td>Text background color.</td>
</tr>
<tr>
<td>V_COL_FG</td>
<td>-36</td>
<td>WORD</td>
<td>Text foreground color.</td>
</tr>
<tr>
<td>V_CUR_AD</td>
<td>-34</td>
<td>LONG</td>
<td>Text cursor physical address.</td>
</tr>
<tr>
<td>V_CUR_OF</td>
<td>-30</td>
<td>WORD</td>
<td>Offset (in bytes) from physical screen address to the top of the first text character.</td>
</tr>
<tr>
<td>V_CUR_XY</td>
<td>-28</td>
<td>2 WORDs</td>
<td>X and Y character position of the text cursor.</td>
</tr>
<tr>
<td>V_PERIOD</td>
<td>-24</td>
<td>BYTE</td>
<td>Current cursor blink rate.</td>
</tr>
<tr>
<td>V_CUR_CT</td>
<td>-23</td>
<td>BYTE</td>
<td>Countdown timer to next blink.</td>
</tr>
<tr>
<td>V_FNT_AD</td>
<td>-22</td>
<td>LONG</td>
<td>Pointer to system font data (monospaced).</td>
</tr>
<tr>
<td>V_FNT_ND</td>
<td>-18</td>
<td>WORD</td>
<td>Last ASCII character in font.</td>
</tr>
<tr>
<td>V_FNT_ST</td>
<td>-16</td>
<td>WORD</td>
<td>First ASCII character in font.</td>
</tr>
<tr>
<td>V_REZ_HZ</td>
<td>-14</td>
<td>WORD</td>
<td>Width of the system font form in bytes.</td>
</tr>
<tr>
<td>V_REZ_VT</td>
<td>-12</td>
<td>WORD</td>
<td>Horizontal pixel resolution.</td>
</tr>
<tr>
<td>V_OFF_AD</td>
<td>-10</td>
<td>LONG</td>
<td>Pointer to font offset table.</td>
</tr>
<tr>
<td>RESERVED</td>
<td>-6</td>
<td>WORD</td>
<td>Reserved for future use.</td>
</tr>
<tr>
<td>V_REZ_VT</td>
<td>-4</td>
<td>WORD</td>
<td>Vertical pixel resolution.</td>
</tr>
<tr>
<td>BYTES_LIN</td>
<td>-2</td>
<td>WORD</td>
<td>Bytes per screen line.</td>
</tr>
<tr>
<td>PLANES</td>
<td>0</td>
<td>WORD</td>
<td>Number of planes in the current resolution.</td>
</tr>
<tr>
<td>WIDTH</td>
<td>2</td>
<td>WORD</td>
<td>Width of the destination form in bytes.</td>
</tr>
<tr>
<td>CONTRL</td>
<td>4</td>
<td>LONG</td>
<td>Pointer to the CONTRL array.</td>
</tr>
<tr>
<td>INTIN</td>
<td>8</td>
<td>LONG</td>
<td>Pointer to the INTIN array.</td>
</tr>
<tr>
<td>PTSIN</td>
<td>12</td>
<td>LONG</td>
<td>Pointer to the PTSIN array.</td>
</tr>
<tr>
<td>INTOUT</td>
<td>16</td>
<td>LONG</td>
<td>Pointer to the INTOUT array.</td>
</tr>
<tr>
<td>PTSOUT</td>
<td>20</td>
<td>LONG</td>
<td>Pointer to the PTSOUT array.</td>
</tr>
<tr>
<td>COLBIT0</td>
<td>24</td>
<td>WORD</td>
<td>Color bit value used for plane 0.</td>
</tr>
<tr>
<td>COLBIT1</td>
<td>26</td>
<td>WORD</td>
<td>Color bit value used for plane 1.</td>
</tr>
<tr>
<td>COLBIT2</td>
<td>28</td>
<td>WORD</td>
<td>Color bit value used for plane 2.</td>
</tr>
<tr>
<td>COLBIT3</td>
<td>30</td>
<td>WORD</td>
<td>Color bit value used for plane 3.</td>
</tr>
<tr>
<td>LSTLIN</td>
<td>32</td>
<td>WORD</td>
<td>Last pixel draw flag (0=draw, 1=don’t draw). Used to prevent the last pixel in a polyline segment drawn in XOR mode from overwriting the first pixel in the next line.</td>
</tr>
<tr>
<td>LNMASK</td>
<td>34</td>
<td>WORD</td>
<td>Line draw pattern mask.</td>
</tr>
<tr>
<td>WMODE</td>
<td>36</td>
<td>WORD</td>
<td>VDI writing mode.</td>
</tr>
<tr>
<td>X1</td>
<td>38</td>
<td>WORD</td>
<td>X coordinate for point 1.</td>
</tr>
<tr>
<td>Y1</td>
<td>40</td>
<td>WORD</td>
<td>Y coordinate for point 1.</td>
</tr>
<tr>
<td>X2</td>
<td>42</td>
<td>WORD</td>
<td>X coordinate for point 2.</td>
</tr>
<tr>
<td>Y2</td>
<td>44</td>
<td>WORD</td>
<td>Y coordinate for point 2.</td>
</tr>
<tr>
<td>PATPTR</td>
<td>46</td>
<td>LONG</td>
<td>Fill-pattern pointer.</td>
</tr>
<tr>
<td>Variable</td>
<td>Size</td>
<td>Type</td>
<td>Description</td>
</tr>
<tr>
<td>------------</td>
<td>------</td>
<td>-------</td>
<td>-----------------------------------------------------------------------------</td>
</tr>
<tr>
<td>PATMSK</td>
<td>50</td>
<td>WORD</td>
<td>This value is AND'ed with the value in Y1 to give an index into the current fill pattern for the current line.</td>
</tr>
<tr>
<td>MFILL</td>
<td>52</td>
<td>WORD</td>
<td>Multiplane fill pattern flag (0=Mono).</td>
</tr>
<tr>
<td>CLIP</td>
<td>54</td>
<td>WORD</td>
<td>Clipping flag (0=disabled).</td>
</tr>
<tr>
<td>XINCL</td>
<td>56</td>
<td>WORD</td>
<td>Left edge of clipping rectangle.</td>
</tr>
<tr>
<td>XMAXCL</td>
<td>58</td>
<td>WORD</td>
<td>Right edge of clipping rectangle.</td>
</tr>
<tr>
<td>YMINCL</td>
<td>60</td>
<td>WORD</td>
<td>Top edge of clipping rectangle.</td>
</tr>
<tr>
<td>YMAXCL</td>
<td>62</td>
<td>WORD</td>
<td>Bottom edge of clipping rectangle.</td>
</tr>
<tr>
<td>XDDA</td>
<td>64</td>
<td>WORD</td>
<td>Text scaling accumulator (set to $8000 prior to blitting text).</td>
</tr>
</tbody>
</table>
| DDAINC     | 66   | WORD  | Scaling increment. If SIZE1 is the actual point size and SIZE2 is the desired point size then to scale up use: \\
|            |      |       | \( DDAINC = 256 \times \frac{(SIZE2 - SIZE1)}{SIZE1} \) \) \ To scale down use: \\
|            |      |       | \( DDAINC = 256 \times \frac{SIZE2}{SIZE1} \) \) |
| SCALDIR    | 68   | WORD  | Text scaling direction (0=down, 1=up).                                      |
| MONO       | 70   | WORD  | Monospaced font flag.                                                       |
| SOURCEX    | 72   | WORD  | X coordinate of character in font form.                                     |
| SOURCEY    | 74   | WORD  | Y coordinate of character in font form.                                     |
| DESTX      | 76   | WORD  | X position on screen to output character at.                                |
| DESTY      | 78   | WORD  | Y position on screen to output character at.                                |
| DELX       | 80   | WORD  | Width of the character to output.                                           |
| DELY       | 82   | WORD  | Height of the character to output.                                          |
| FBASE      | 84   | LONG  | Pointer to the font character image block.                                  |
| FWIDTH     | 88   | WORD  | Width of the font form in bytes.                                            |
| STYLE      | 90   | WORD  | Special effects flag bitmap as follows: \\
|            |      |       | Bit | Meaning                      |
|            |      |       | 0   | Thickening                    |
|            |      |       | 1   | Lightening                    |
|            |      |       | 2   | Skewing                       |
|            |      |       | 3   | Underlining                   |
|            |      |       | 4   | Outlining (not supported by Line-A)                                        |
| LITEMASK   | 92   | WORD  | Mask to lighten text (usually $5555).                                       |
| SKEWMASK   | 94   | WORD  | Mask to skew text (usually $5555).                                          |
| WEIGHT     | 96   | WORD  | Width to thicken characters by.                                            |
| ROFF       | 98   | WORD  | Offset above baseline used for italicizing.                                 |
| LOFF       | 100  | WORD  | Offset below baseline used for italicizing.                                 |
| SCALE      | 102  | WORD  | Text scaling flag (0=no scale).                                             |
| CHUP       | 104  | WORD  | Character rotation angle in tenths of degrees (supported only in 90 degree increments). |
| TEXTFG     | 106  | WORD  | Text foreground color.                                                     |
| SCRTCHP    | 108  | LONG  | Pointer to two contiguous scratch buffers used in creating text special effects. |
| SCRPT2     | 112  | WORD  | Offset from first buffer to second (in bytes).                              |
| TEXTBG     | 114  | WORD  | Text background color.                                                     |
| COPYTRAN   | 116  | WORD  | Copy raster mode (0=Opaque, 1=Transparent).                                 |
Line-A Font Headers

Raster system and GDOS fonts are linked to form a list of font headers which contain the information needed to render text. Outline text generated by FSM is inaccessible in this manner.

Each monospaced font contains a font header, character and horizontal offset table, and font form. All data types are in “Little Endian” (Intel format) and as such must be byte-swapped before use.

The font form is a raster form with each character laid side-by-side on the horizontal plane. The first character is WORD aligned but padding within the form only occurs at the end of a scanline to force the next scanline to be WORD aligned.

Each font header contains a pointer to the next font in the list. The list is terminated by a NULL pointer. The font header format is as follows:

<table>
<thead>
<tr>
<th>Name</th>
<th>Offset</th>
<th>Type</th>
<th>Contents</th>
</tr>
</thead>
<tbody>
<tr>
<td>font_id</td>
<td>0</td>
<td>WORD</td>
<td>Font ID number (must be unique).</td>
</tr>
<tr>
<td>point</td>
<td>2</td>
<td>WORD</td>
<td>Point size of font.</td>
</tr>
<tr>
<td>name</td>
<td>4</td>
<td>32 BYTES</td>
<td>ASCII Name of font.</td>
</tr>
<tr>
<td>first_ade</td>
<td>36</td>
<td>UWORD</td>
<td>First ASCII character in font.</td>
</tr>
<tr>
<td>last_ade</td>
<td>38</td>
<td>UWORD</td>
<td>Last ASCII character in font.</td>
</tr>
<tr>
<td>top</td>
<td>40</td>
<td>UWORD</td>
<td>Distance from the top line of the font to the baseline.</td>
</tr>
<tr>
<td>ascent</td>
<td>42</td>
<td>UWORD</td>
<td>Distance from the ascent line of the font to the baseline.</td>
</tr>
<tr>
<td>half</td>
<td>44</td>
<td>UWORD</td>
<td>Distance from the half line of the font to the baseline.</td>
</tr>
<tr>
<td>descent</td>
<td>46</td>
<td>UWORD</td>
<td>Distance from the descent line of the font to the baseline.</td>
</tr>
<tr>
<td>bottom</td>
<td>48</td>
<td>UWORD</td>
<td>Distance from the bottom line of the font to the baseline.</td>
</tr>
<tr>
<td>max_char_width</td>
<td>50</td>
<td>UWORD</td>
<td>Width of the widest character in the font.</td>
</tr>
<tr>
<td>max_cell_width</td>
<td>52</td>
<td>UWORD</td>
<td>Width of the widest character cell in the font.</td>
</tr>
<tr>
<td>left_offset</td>
<td>54</td>
<td>UWORD</td>
<td>Amount character slants left when skewed.</td>
</tr>
<tr>
<td>right_offset</td>
<td>56</td>
<td>UWORD</td>
<td>Amount character slants right when skewed.</td>
</tr>
<tr>
<td>thicken</td>
<td>58</td>
<td>UWORD</td>
<td>Number of pixels to smear for thickening.</td>
</tr>
<tr>
<td>ul_size</td>
<td>60</td>
<td>UWORD</td>
<td>Size of an appropriate underline for the font.</td>
</tr>
<tr>
<td>lighten</td>
<td>62</td>
<td>UWORD</td>
<td>Mask for character lightening.</td>
</tr>
<tr>
<td>skew</td>
<td>64</td>
<td>UWORD</td>
<td>Mask for character skewing.</td>
</tr>
<tr>
<td>flags</td>
<td>66</td>
<td>UWORD</td>
<td>Font type flags.</td>
</tr>
<tr>
<td>hor_table</td>
<td>68</td>
<td>LONG</td>
<td>Pointer to the horizontal offset table. The horizontal offset table is an array of bytes with one entry per character denoting the pixel offset to the character.</td>
</tr>
</tbody>
</table>
**Line-A Function Calling Procedure**

*Line-A* functions are called by simply inserting the opcode into the instruction stream. For example, the ‘Hide Mouse’ function is called with the following assembly language instruction:

```
dc.w  $A00A
```

Generally, the *Line-A* initialization function is called ($A000) and the address of the variable and/or font header tables are stored. Prior to each *Line-A* call variables are set as explained in the *Line-A Function Reference* and the function is then called. There is no method of error reporting available.
**$A000 - Initialize**

Return pointers to the **Line-A** variable structures.

**Example Binding**

; Retrieve Line-A variable table address
; and store in A5 for other bindings

```assembly
    .dc.w $A000
    .move.l a0,a5 ; Line-A variables
    .move.l a1,a6 ; System font headers
```

**Return Value**

The initialize function returns the following information:

<table>
<thead>
<tr>
<th>Register</th>
<th>Contents</th>
</tr>
</thead>
<tbody>
<tr>
<td>D0</td>
<td>Pointer to Line-A variable table.</td>
</tr>
<tr>
<td>A0</td>
<td>Pointer to Line-A variable table.</td>
</tr>
<tr>
<td>A1</td>
<td>Pointer to a NULL terminated array of pointers to system font headers.</td>
</tr>
<tr>
<td>A2</td>
<td>Pointer to a longword array containing sixteen pointers which are addresses of the actual Line-A functions in memory. For example, JSR'ing through the pointer in the first array element has the same result as calling the Initialize instruction by an exception except that the function must be called from supervisor mode.</td>
</tr>
</tbody>
</table>

**Comments**

This call is required to return the address of the **Line-A** variable structure needed for all other **Line-A** calls. All processes (including the VDI) share this structure so don’t expect variables to remain constant between calls.

**See Also**

v_opnvwk()

---

**$A001 - Plot Pixel**

Plot a single pixel at the specified coordinates.

**Parameters**

*INTIN* points to a **WORD** containing the color register of the pixel to plot at the specified coordinates. *PTSIN* points to two **WORDS** which are the X and Y coordinates respectively.

**Example Binding**

; Plot a pixel at ( 10, 10 ) using color 1

```assembly
    move.l #intin,8(a5)
    move.l #ptsin,12(a5)
    .dc.w $A001
    .data
    intin: .dc.w 1
    ptsin:
```

---

*THE ATARI COMPENDIUM*
$A002 - Get Pixel

Get the color register of the pixel at the specified coordinates.

**PARAMETERS**

*PTSIN* points to two words which are the X and Y coordinates of the pixel to read.

**EXAMPLE**

; Read the color index of point (10, 10)

```
move.l #ptsin,12(a5)
.dc.w $A002
.data
ptsin:
.dc.w 10, 10
```

**RETURN VALUE**

The color register of the pixel is returned in D0.

**SEE ALSO**

v_getpixel()


$A003 - Arbitrary Line

Draw a line between any two coordinates.

**PARAMETERS**

*COLBIT0-4* are set appropriately to determine the line color. *LSTLIN* is a flag in which a value of 0 specifies to draw the last point in each line or a value of 1 which specifies not to. *LNMASK* specifies the pattern mask to apply to the line. *WRMODE* specifies the write mode of the function (0-3). (X1, Y1), and (X2, Y2) give the starting and ending coordinates of the line.

**EXAMPLE**

; Draw a solid line from (0, 0) to (100, 100)

```
move.w #1,24(a5) ; COLBIT 0
move.w #1,26(a5) ; COLBIT 1
move.w #1,28(a5) ; COLBIT 2
move.w #1,30(a5) ; COLBIT 3
move.w #0,32(a5) ; LSTLIN
move.w #$FFFF,34(a5) ; LNMASK
move.w #0,36(a5) ; WRMODE
move.w #0,38(a5) ; X1
move.w #0,40(a5) ; Y1
move.w #100,42(a5) ; X2
move.w #100,42(a5) ; Y2
.dc.w $A003
```
Caveats

LNMSK is modified as a result of this call.

See Also

$A004, v_pline()

$A004 - Horizontal Line

Draw a horizontal line between the specified coordinates.

Parameters

COLBIT0-3 defines the color of the line and WRMODE determines the write mode (0-3). (X1, Y1) and (X2, Y1) determine the starting and ending points of the line. PATMSK is AND’ed with Y1 to determine a line index into the pattern pointed to by PATPTR. PATMSK is normally the number of lines in the pattern (should be an even power of 2) minus one. If MFILL is non-zero, WMODE is disregarded and the fill is colored from the values in COLBIT0-3.

Example Binding

; Draw a horizontal dashed line from (0, 10) to (100, 10)

move.w #1,24(a5) ; COLBIT 0
move.w #1,26(a5) ; COLBIT 1
move.w #1,28(a5) ; COLBIT 2
move.w #1,30(a5) ; COLBIT 3
move.w #0,36(a5) ; WRMODE
move.w #0,38(a5) ; X1
move.w #0,40(a5) ; Y1
move.w #100,42(a5) ; X2
move.l #pat,46(a5) ; PATPTR
move.w #0,50(a5) ; PATMSK
move.w #0,52(a5) ; MFILL
.dc.w $A004

See Also

v_pline()

$A005 - Filled Rectangle

Draw a filled rectangle at the specified coordinates.

Parameters

CLIP is a flag which when set to 1 enables clipping and when set to 0 disables it. All output of this function is confined to the region bounded by (XMINCL, YMINCL) and (XMAXCL, YMAXCL). Other parameters are consistent with the definitions given under $A004.

Example Binding

; Draw a filled rectangle with its upper left corner at (0, 0) and its lower right corner at (100, 100). Clip the rectangle to within (10, 10) and (90, 90)

move.w #1,24(a5) ; COLBIT 0
move.w #1,26(a5); COLBIT1
move.w #1,28(a5); COLBIT2
move.w #1,30(a5); COLBIT3
move.w #0,36(a5); WRMODE
move.w #0,38(a5); X1
move.w #0,40(a5); Y1
move.w #100,42(a5); X2
move.w #100,44(a5); Y2
move.l #stipple,46(a5); PATPTR
move.w #1,50(a5); PATMSK
move.w #0,52(a5); MFILL
move.w #1,54(a5); CLIP
move.w #10,56(a5); XMINCL
move.w #10,58(a5); YMINCL
move.w #90,60(a5); XMAXCL
move.w #90,62(a5); YMAXCL.
.dc.w $A005

.stipple:
.dc.w $AAAA
.dc.w $5555

See Also v_bar(), vr_recfl()

$A006 - Filled Polygon

Draw a filled polygon line-by-line.

Parameters

PTSIN contains the X/Y coordinate pairs of the vertices of the polygon with the last point being equal to the first. CONTRL[1] specifies the number of vertices. The rest of the variables are consistent with previous usages.

Example Binding

; Draw a filled polygon with vertices at
; ( 0, 0 ), ( 319, 120 ), and ( 25, 199 ).
move.l #ptsin,12(a5); PTSIN
move.l #contrl,4(a5); CONTRL
move.w #1,24(a5); COLBIT0
move.w #1,26(a5); COLBIT1
move.w #1,28(a5); COLBIT2
move.w #1,30(a5); COLBIT3
move.w #0,36(a5); WRMODE
move.w #stipple,46(a5); PATPTR
move.w #1,50(a5); PATMSK
move.w #0,52(a5); MFILL
move.w #0,54(a5); CLIP

; loop to draw the polygon
move.w #0,40(a5); upper Y line
move.w #199,d4; lowest Y line
; - upper Y line
loop:
.dc.w $A006
addq.w #1,40(a5)
Perform a bit-block transfer.

**Parameters**

The address of a `BitBlt` parameter block is passed in register A6. That structure is defined with the following members:

<table>
<thead>
<tr>
<th>Member</th>
<th>Offset/Type</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>B_WD</td>
<td>+0 (WORD)</td>
<td>Width of block to blit (in pixels)</td>
</tr>
<tr>
<td>B_HT</td>
<td>+2 (WORD)</td>
<td>Height of block to blit (in pixels)</td>
</tr>
<tr>
<td>PLANEC_CT</td>
<td>+4 (WORD)</td>
<td>Number of bit planes to blit.</td>
</tr>
<tr>
<td>FG_COL†</td>
<td>+6 (WORD)</td>
<td>Bit array used to create index into OP_TAB. FG_COL contributes its bit #n' (where 'n' is the plane number) to bit #1 of the index used to select the operation code from OP_TAB.</td>
</tr>
<tr>
<td>BG_COL†</td>
<td>+8 (WORD)</td>
<td>Bit array used to create index into OP_TAB. BG_COL contributes its bit #n' (where 'n' is the plane number) to bit #0 of the index used to select the operation code from OP_TAB.</td>
</tr>
<tr>
<td>OP_TAB</td>
<td>+10 (LONG)</td>
<td>OP_TAB is a 4 byte array containing four logic operation codes (0 to 16) to be applied to the image. The table is indexed by using the bit in FG_COL and BG_COL corresponding to the current plane as bit #1 and bit #0 respectively yielding an offset into OP_TAB of 0-3.</td>
</tr>
<tr>
<td>S_XMIN</td>
<td>+14 (WORD)</td>
<td>X pixel offset to source upper left.</td>
</tr>
<tr>
<td>S_YMIN</td>
<td>+16 (WORD)</td>
<td>Y pixel offset to source upper left.</td>
</tr>
<tr>
<td>S_FORM</td>
<td>+18 (WORD)</td>
<td>Address of the source form.</td>
</tr>
<tr>
<td>S_NXWD</td>
<td>+22 (LONG)</td>
<td>Number of bits per pixel.</td>
</tr>
<tr>
<td>S_NXLN</td>
<td>+24 (WORD)</td>
<td>Byte width of form.</td>
</tr>
<tr>
<td>S_NXPL</td>
<td>+26 (WORD)</td>
<td>Byte offset between planes (always 2).</td>
</tr>
<tr>
<td>D_XMIN</td>
<td>+28 (WORD)</td>
<td>X pixel offset to destination upper left.</td>
</tr>
<tr>
<td>D_YMIN</td>
<td>+30 (WORD)</td>
<td>Y pixel offset to destination upper left.</td>
</tr>
</tbody>
</table>
### D_FORM
+32 (LONG) Address of the destination form.

### D_NXWD
+36 (WORD) Number of bits per pixel.

### D_NXLN
+38 (WORD) Byte width of form.

### D_NXPL
+40 (WORD) Byte offset between planes (always 2).

### P_ADDR
+42 (LONG) Address of pattern buffer (0 = no pattern).

### P_NXLN
+46 (WORD) Bytes of pattern per line (should be even).

### P_NXPL
+48 (WORD) Bytes of pattern per plane (if using a single plane fill with a multi-plane destination, this should be 0).

### P_MASK
+50 (WORD) P_MASK is found by the expression:

\[
P\_MASK = (\text{length in words} - 1) \ll n
\]

If \( P\_NXLN = 2^n \) then

\[
P\_MASK = (\text{length in words} - 1) \ll n
\]

### SPACE
+52 (WORD) 24 bytes of blank space which must be reserved as work area for the function.

†These members may be altered by this function.

**Example Binding**

```plaintext
; Perform a blit using the information located at bprmblk
lea bprmblk, a6
.dc.w $A007
```

**See Also**

vro_cpyfm(), vrt_cpyfm()

---

### $A008 - TextBlt

Blit a single character to the screen.

**Parameters**

When performing this call, the following **Line-A** variables are evaluated:

<table>
<thead>
<tr>
<th>Variable</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>WMODE</td>
<td>Writing mode (see comments below).</td>
</tr>
<tr>
<td>CLIP, XMINCL, YMINCL, XMAXCL, YMAXCL</td>
<td>Standard clipping flags and extents.</td>
</tr>
<tr>
<td>XDDA</td>
<td>Scaling accumulator (should be initialized to $8000 prior to each TextBlt call when scaling).</td>
</tr>
</tbody>
</table>
| DDAINC   | This amount specifies the fractional amount to scale the character outputted by. If scaling down, this value may be found by the formula:

\[
0 \times 100 \times \text{scaled size} / \text{actual size}
\]

If scaling up, this value may be found with the formula:

\[
0 \times 100 \times (\text{scaled size} - \text{actual size}) / \text{actual size}
\]

This variable is only evaluated if scaling is active. |
<p>| SCALDIR  | Scaling direction (1 = up, 0 = down). |</p>
<table>
<thead>
<tr>
<th>MONO</th>
<th>If 1 set to monospacing mode, if 0 set to proportional spacing mode.</th>
</tr>
</thead>
<tbody>
<tr>
<td>SOURCEX, SOURCEY</td>
<td>SOURCEX is the pixel offset into the font form of the character you wish to render. SOURCEY is usually 0 indicating that you wish to render the character from the top.</td>
</tr>
<tr>
<td>DESTX, DESTY</td>
<td>DESTX and DESTY specify the destination screen coordinates of the character.</td>
</tr>
<tr>
<td>DELX, DELEY</td>
<td>DELX and DELEY specify the width and height of the character to print.</td>
</tr>
<tr>
<td>FBASE</td>
<td>Pointer to start of font data.</td>
</tr>
<tr>
<td>FWIDTH</td>
<td>Width of font form.</td>
</tr>
<tr>
<td>STYLE</td>
<td>STYLE is a mask of the following bits indicating special effects: 0x01 = Bold 0x02 = Light 0x04 = Italic 0x08 = Underlined 0x10 = Outlined</td>
</tr>
<tr>
<td>LITEMASK</td>
<td>Mask used to lighten text (usually $5555).</td>
</tr>
<tr>
<td>SKEWMASK</td>
<td>Mask used to italicize text (usually $5555).</td>
</tr>
<tr>
<td>WEIGHT</td>
<td>Width by which to thicken boldface text (should be set from font header).</td>
</tr>
<tr>
<td>ROFF</td>
<td>Offset above character baseline when skewing (set from font header).</td>
</tr>
<tr>
<td>LOFF</td>
<td>Offset below character baseline when skewing (from font header).</td>
</tr>
<tr>
<td>SCALE</td>
<td>Scaling flag (0 = no scaling, 1 = scale text).</td>
</tr>
<tr>
<td>CHUP</td>
<td>Character rotation vector (may be 0, 900, 1800, or 2700).</td>
</tr>
<tr>
<td>TEXTFG</td>
<td>Text foreground color.</td>
</tr>
<tr>
<td>SCRTCHP</td>
<td>Pointer to start of text special effects buffer (should be twice as large as the largest distorted character and is only required when using a special effect).</td>
</tr>
<tr>
<td>SCRPT2</td>
<td>Offset of scaling buffer in SCRTCHP (midpoint).</td>
</tr>
<tr>
<td>TEXTBG</td>
<td>Text background color.</td>
</tr>
</tbody>
</table>

**EXAMPLE BINDING**

; Print a NULL-terminated string with ; no effects or clipping

```assembly
move.w #0,36(a5) ; WMODE
move.w #0,54(a5) ; CLIP
move.w #1,106(a5) ; TEXTFG
move.w #0,114(a5) ; TEXTBG
move.w #100,76(a5) ; DESTX
move.w #100,78(a5) ; DESTY
move.w #4,90(a5) ; STYLE
move.w #0,102(a5) ; SCALE
move.w #1,70(a5) ; MONO

; Find the 8x8 font
move.w 4(a6),a6 ; Address of 8x8 font
move.w 76(a6),84(a5) ; FBASE
move.w 80(a6),88(a5) ; FWIDTH
move.w 82(a6),82(a5) ; DELEY

; Print the string
lea string,a2
move.l 72(a6),a3 ; offset table
```

---

**THE ATARI COMPENDIUM**
moveq.l       #0,d0

print:
move.b       (a2)+,d0 ; Get next char
ble          end
sub.w        36(a6),d0 ; Fix offset
isl.w        #1,d0 ; Double for
               ; WORD offset
move.w       0(a3,d0),72(a5) ; SOURCEX
move.w       2(a3,d0),d0 ; x of next char
sub.w        72(a5),d0 ; get true width
move.w       d0,80(a5) ; DELX
moveq.l      #0,74(a5) ; SOURCEY
movem.l      a0-a2,-(sp) ; Save a0-a2
.dc.w        $A008
movem.l      (a7)+,a0-a2 ; Restore regs
bra          print
end:
             rts

.string:     .dc.b "The Atari Compendium",0

COMMENTS    The value for WMODE is a special case with TextBlt. Values from 0-3 translate to the standard VDI modes. Values from 4-19 translate to the BitBlt modes 0-15.

SEE ALSO    v_gtext()

$A009 - Show Mouse

Show the mouse cursor.

PARAMETERS  No parameters required. Optionally, INTIN can be made to point to a WORD value of 0 to force the mouse cursor to be displayed regardless of the number of times it was hidden.

EXAMPLE Binding

; Show the mouse regardless of the number
; of times it was hidden

move.l       #intin,8(a5) ; INTIN
.dc.w        $A009

.intin:      .dc.w 0

COMMENTS    ‘Show’ and ‘Hide’ mouse calls are nested, that is, in order to return the mouse cursor to its original state, it must be ‘shown’ the same number of times it was ‘hidden’.

SEE ALSO    v_show_c(), graf_mouse()
$A00A - Hide Mouse

Hide the mouse cursor.

**Example Binding**

; Remove the mouse from the screen

    .dc.w $A00A

**Comments**

See ‘Show Mouse’.

**See Also**

v_hide_c(), graf_mouse()

$A00B - Transform Mouse

Change the mouse’s form.

**Parameters**

On entry **INTIN** should point to a structure containing the new mouse form data. The format of the structure is defined under the entry for **vsc_form()**.

**Example Binding**

; Change the mouse form to the data held in
; the newmouse structure.

    move.b -339(a5),d0 ; Save old value
    move.b #0,-339(a5) ; Disable mouse
    move.l #newmouse,8(a5) ; INTIN
    .dc.w $A00B
    move.b d0,-339(a5) ; Restore
    ; MOUSE_FLAG

**Comments**

The old data can be saved from the information stored in the **Line-A** variable table at offset -356. To avoid ‘mouse droppings’ you should disable mouse interrupts by setting **MOUSE_FLAG** (offset -339) to 0 and restoring it when done.

**See Also**

vsc_form(), graf_mouse()

$A00C - Undraw Sprite

Undraw a previously drawn sprite.

**Parameters**

Prior to calling this function, A2 should be loaded with a pointer to the ‘sprite save block’ defined when drawing the sprite. For the format of this data, see ‘Draw Sprite’

**Example**

; ‘Undraw’ sprite previously drawn from data
$A00D - Draw Sprite

Draw a 16x16 sprite on the screen.

Parameters

Prior to calling this function, four 68x00 registers must be initialized. D0 and D1 should contain the horizontal and vertical position respectively of the coordinates of the sprite to draw. This is relative to the ‘hot spot’ of the sprite as defined in the sprite definition block.

A0 should contain a pointer to a sprite definition block defined as follows:

<table>
<thead>
<tr>
<th>Offset/Type</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>0x0000 (WORD)</td>
<td>X offset of ‘hot spot’. This value is subtracted from the value given in D0 to yield the actual screen position of the upper-left pixel.</td>
</tr>
<tr>
<td>0x0002 (WORD)</td>
<td>Y offset of ‘hot spot’. This value is subtracted from the value given in D1 to yield the actual screen position of the upper-right pixel.</td>
</tr>
<tr>
<td>0x0004 (WORD)</td>
<td>Format flag. This value specifies the mode in which the mouse pointer will be drawn. A value of 1 specifies ‘VDI mode’ whereas -1 specifies X-OR mode. The default is 1.</td>
</tr>
<tr>
<td>0x0006 (WORD)</td>
<td>Background color of sprite.</td>
</tr>
<tr>
<td>0x0008 (WORD)</td>
<td>Foreground color of sprite.</td>
</tr>
<tr>
<td>0x000A (32 WORDs)</td>
<td>Sprite form data. The bitmap data consists of two 16x16 rasters, one each for the mask and data portion of the form. The data is presented in interleaved format. The first WORD of the mask portion is first, followed by the first WORD of the data portion, and so on.</td>
</tr>
</tbody>
</table>

Register A2 is a pointer to a buffer which will be used to save the screen area where the sprite is drawn. The size of the buffer can be determined by the following formula:

\[
(10 + (VPLANES \times 64))
\]

Example Binding

; Draw a sprite at (100, 100) whose data is stored at spritedef with a valid save buffer at savebuf.

move.w #100,d0 ; X position
move.w  #100,d1  ; Y position
move.l  #sprite1,a0  ; Sprite form
move.l  #savebuf,a2  ; Save buffer
.dc.w $A00D

Caveats
Register A6 is destroyed as a result of this call.

Comments
In order to avoid the mouse form running into any sprites you draw, the mouse should be hidden before drawing and restored afterwards. It may also be advisable to call Vsync() prior to each call to avoid screen flicker.

$A00E - Copy Raster

Copy a raster form using opaque or transparent mode.

Parameters
INTIN should point to a WORD array whose first entry specifies the write mode of the operation. In transparent mode, this is a VDI standard mode (0-3), however in opaque mode the full range of BitBlt modes (0-15) are available. In transparent mode, the second and third array entries of INTIN contain the foreground and background color of the destination copy respectively.

CONTRL should point to a memory buffer which is filled in with the source and destination MFDB’s (Memory Form Definition Block’s) at offsets 14 and 18 respectively. The structure of an MFDB is discussed under vro_cpyfm().

PTSIN should point to an array of 8 WORD’s containing the pixel offsets for the blit in the order SX1, SY1, SX2, SY2, DX1, DY1, DX2, DY2.

COPYTRAN specifies the write mode. A value of 0 indicates an opaque copy while a value of 1 indicates a transparent copy.

The settings for CLIP, XMINCL, YMINCL, XMAXCL, and YMAXCL are utilized by this call.

Example Binding
; Copy a 32x32 raster form 'myrast' from a buffer in memory to the ST medium resolution screen at (100, 100) using transparent mode.

move.l  #contrl,4(a5)  ; CONTRL
move.l  #srcmfdb,contrl+14
move.l  #destmfdb,contrl+18
move.l  #intin,4(a5)  ; INTIN
move.l  #ptsin,4(a5)  ; PTSIN
move.w  #1,116(a5)  ; COPYTRAN
move.w  #0,54(a5)  ; CLIP

; Fill in some info for MFDB’s

The Atari Compendium
8.22 – Line-A Function Reference

```assembly
move.l #myrast, srcmfdb ; Source raster
move.w #$02, -(sp) ; Physbase()
trap #14
addq.l #2, sp
move.l d0, destmfdb
.data
.dc.w $A00E

contrl:
    .dc.w 0, 0, 0, 0, 0, 0, 0, 0
intin:
    .dc.w 0, 1, 0
ptsin:
    .dc.w 0, 0, 15, 15, 100, 100, 115, 115
srcmfdb:
    .dc.w 0, 0, 16, 16, 1, 0, 0, 0
destmfdb:
    .dc.w 0, 0, 320, 200, 16, 0, 2, 0
myrast:
    .dc.w $AAAA, $AAAA, $AAAA, $AAAA
    .dc.w $5555, $5555, $5555, $5555
    .dc.w $AAAA, $AAAA, $AAAA, $AAAA
    .dc.w $5555, $5555, $5555, $5555
    .dc.w $AAAA, $AAAA, $AAAA, $AAAA
    .dc.w $5555, $5555, $5555, $5555
```

**COMMENTS**

For a more indepth explanation, refer to the VDI calls parallel to these, `vro_cpyfm()` and `vrt_cpyfm()`.

**SEE ALSO**

`vro_cpyfm()`, `vrt_cpyfm()`

---

### $A00F - Seed Fill

Seed fill an irregularly shaped region.

**PARAMETERS**

`INTIN` points to a word value which specifies the mode of this function. If the value is negative, color mode is used. In color mode, the fill spreads from the initial point until it hits a color other than that of the initial point. If the value is positive, outline mode is used. It then is interpreted as the VDI color index value at which to stop the fill.

`PTSIN` points to an array of two WORDs which specify the X and Y coordinates respectively of the initial fill point.

`CUR_WORK` should point to a WORD array of 16 words with the sixteenth WORD being the fill color specified as a VDI color index.

`WMODE` specified the VDI writing mode of the fill (0-3). `PATPTR` and `PATMSK`
define the fill pattern (as defined in `Horizontal Line`).

`SEEDABORT` points to a user routine which can abort the fill, if desired, when called. This routine is called once for each line of the fill. It should zero register D0 to continue or place a non-zero value in it to abort.

**EXAMPLE BINDING**

; Seed fill an area starting at ( 100, 100 )
; in color mode with a clip region defined
; as the VDI rectangle ( 50, 50 ), ( 200, 200 ).

```assembly
move.l #intin,8(a5) ; INTIN
move.l #ptsin,12(a5) ; PTSIN
move.l #cur_work,-464(a5) ; CUR_WORK
move.l #seedabort,118(a5) ; SEEDABORT
move.w #0,36(a5) ; WMODE
move.w #stipple,46(a5) ; PATPTR
move.w #0,50(a5) ; PATMASK
move.w #0,52(a5) ; MFILL
move.w #50,56(a5) ; XMINCL
move.w #50,58(a5) ; YMINCL
move.w #200,60(a5) ; XMAXCL
move.w #200,62(a5) ; YMAXCL
.dc.w $A00F
seedabort:
    moveq.l #0, d0 ; Clear D0
    rts
.data
intin:           .dc.w -1
ptsin:           .dc.w 100, 100
cur_work:        .dc.w 0, 0, 0, 0, 0, 0, 0, 0
                 .dc.w 0, 0, 0, 0, 0, 0, 0, 1
stipple:         .dc.w $AAAA
                 .dc.w $5555

COMMENTS
The clipping variables `XMINCL`, `YMINCL`, `XMAXCL`, and `YMAXCL` must always be set as they are interpreted regardless of the clipping flag.

SEE ALSO `v_contourfill()`
Overview

The ‘Desktop’ is a GEM application that is started after the operating system is initialized and all ‘AUTO’ folder programs and desk accessories are loaded. The desktop is responsible for providing basic file management and program launching abilities to the user.

Normally, the desktop is contained in ROM, however under MultiTOS, the desktop may be soft-loaded by placing the following command line inside the ‘GEM.CNF’ file:

```
shell [new shell filename]
```

If the ‘shell’ command fails, the normal desktop is started.

If an installed shell program exits under MultiTOS, the OS will display a single menu from which programs may be launched.

MultiTOS Considerations

Messages

The desktop may be sent messages using the AES’s `shel_write()` command. The desktop currently recognizes two special messages as follows:

<table>
<thead>
<tr>
<th>Message</th>
<th>Number</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>SH_WDRAW</td>
<td>72</td>
<td>This message tells the desktop that files on a particular drive have been modified so it can update the information in any open windows.</td>
</tr>
<tr>
<td></td>
<td></td>
<td><code>msg[3]</code> should contain the drive number ( 0 = A:, 1 = B:, etc.). A value of -1 will force the desktop to update all of its open windows.</td>
</tr>
<tr>
<td>AP_DRAGDROP</td>
<td>63</td>
<td>The desktop included with AES 4.1 now accepts all drag &amp; drop messages and places the dropped object on the desktop.</td>
</tr>
</tbody>
</table>

Extendibility

The MultiTOS desktop allows the replacement of file copy, rename, and delete, and disk copy and format commands. To replace the file commands, place the filename of an application designed to replace them in the environment variable DESKCOPY. Likewise, a disk command replacement application can be placed in the environment variable DESKFMT.

The file command replacement will be called with one of three command line formats as follows:

1. Copy a file(s):  `c [-options...] [filename(s)] [destination path]`
2. Delete a file(s): `d [-options...] [filename(s)]`
3. Move a file(s):  `−m [-options...] [filename(s)] [destination path]`

The following are valid options to appear on the command line:

<table>
<thead>
<tr>
<th>Option</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>-A</td>
<td>Confirm file copies.</td>
</tr>
<tr>
<td>-B</td>
<td>Do not confirm file copies.</td>
</tr>
<tr>
<td>-C</td>
<td>Confirm file deletes.</td>
</tr>
<tr>
<td>-D</td>
<td>Do not confirm file deletes.</td>
</tr>
<tr>
<td>-E</td>
<td>Confirm file overwrites.</td>
</tr>
<tr>
<td>-F</td>
<td>Do not confirm file overwrites.</td>
</tr>
<tr>
<td>-R</td>
<td>Prompt to rename destination file(s).</td>
</tr>
</tbody>
</table>

An application which is installed to replace disk operations will receive one of two command lines as follows:

1. Format a drive (ex: A:):  `−f A:`

2. Copy a disk (ex: A: to B:):  `−c A: B:``

**TOS Application Launching**

When the user uses the desktop to launch a .TOS or .TTP application under *MultiTOS*, the desktop looks for an environment variable called TOSRUN. If it finds one, it attempts to launch whatever application is specified in that variable with the TOS filename as its parameters.

If the environment variable does not exist, it opens a pipe called ‘U:\PIPE\TOSRUN’ and writes to it the filename and any parameters separated by spaces terminated by a NULL byte.

**Desktop Files**

**DESKTOP.INF**

The desktop in TOS versions less than 2.00 place configuration defaults such as window size and position, drive icons, etc. in the DESKTOP.INF file. In addition, some control panel settings (from CONTROL.ACC, not XCONTROL.ACC) are stored in the file as well.

The DESKTOP.INF file is in standard ASCII text format. This file was not designed to be edited by the user or programmer, but, rather from the desktop itself and will not be discussed in detail.

**NEWDESK.INF**

As of TOS 2.00, the desktop now looks for a file called NEWDESK.INF rather than DESKTOP.INF. This file contains the same information as its predecessor with some additions. Icons which appear on the desktop or in windows may now be linked to icons in the DESKICON.RSC file (as described below). Other entries are still reserved and should be left unmodified.
A creative install program wishing to install custom icons may do so by adding the icons to the DESKICON.RSC file and adding information to NEWDESK.INF which points to the new icons. The install application must be careful to avoid disturbing the original information and icons and must not reorder the icons in the DESKICON.RSC file. The following two lines show example entries in NEWDESK.INF that identify an icon for a file and folder respectively.

```
#I 2C 2C 000 @ *.TXT@ @
#D 1A 1A 000 @ FOLDER@ @
```

The ‘#I’ identifies a file icon and the ‘#D’ identifies a folder icon. The next two numbers should be identical hexadecimal indexes to the icon in the DESKICON.RSC file. The entry ‘000’ is unused and should be included only as a placeholder.

The filename specified on the line can contain wildcard characters and identify the file or folder name(s) which are to be linked. All spaces and ‘@’ characters must appear exactly as above or the system may behave strangely.

**DESKICON.RSC**

The DESKICON.RSC file is a standard GEM resource file (see *Appendix C: Native File Formats*) with one object tree containing a BOX object at the ROOT (object #0) with the icons as children. The position of the icons in the object tree determine their index as referenced by the NEWDESK.INF file.

**DESKCICN.RSC**

This file is supported as of TOS 4.0 and is looked for before DESKICON.RSC. It has an identical format except that it supports the new resource file format and contains color icons rather than monochrome ones.
The Extensible Control Panel

Overview

XCONTROL is a desk accessory which provides a shell for Control Panel Extensions (CPX’s). Typical uses for CPX’s include:

- System Configuration (volume, key click, etc.)
- Hardware Configuration (serial port speed, disk access rate, etc.)
- TSR Configuration

Most CPX’s require only 512 bytes of system memory for header storage when not being executed as they are loaded only when selected by the user.

Applications, games, and other programs not used for configuration purposes should not be created as CPX’s.

CPX Executable Format

A CPX executable is identical to a standard GEMDOS executable with the exception of an additional 512 byte header which precedes the standard 28 byte GEMDOS header. When XCONTROL is initialized at boot time, the header of each CPX contained in the user’s designated CPX directory is loaded and stored. The header data contains the following information:

```c
typedef struct _cpxhead
{
    UWORD magic; /* Magic = 100 dec */
    struct {
        unsigned reserved : 13; /* Reserved */
        unsigned resident : 1; /* Resident CPX if set */
        unsigned bootinit : 1; /* Boot initialize if set */
        unsigned setonly : 1; /* Set only CPX if set */
    } flags;
    LONG cpx_id; /* CPX ID Value */
    UWORD cpx_version; /* CPX Version */
    char i_text[14]; /* Icon Text */
    UWORD sm_icon[48]; /* Icon Bitmap 32x24 */
    UWORD i_color; /* Icon Color */
    char title[18]; /* Title (16 char max) */
    UWORD t_color; /* Title text color */
    char buffer[64]; /* User-storage */
    char reserved[306]; /* Reserved */
} CPXHEAD;
```

Following the 512-byte CPX header the 28-byte GEMDOS header and executable follow. CPX’s do not have a ‘main()’ function. Execution begins at the first instruction of the TEXT segment. The first source file you should link should resemble the following:

```assembly
.xref _cpx_init
```

T H E  A T A R I  C O M P E N D I U M
Every CPX must have a `cpx_init()` function.

If you plan to store defaults back into the CPX using `CPX_Save()` (described later) you should add to the first source file a statement allocating as much storage as you will need at the beginning of the DATA segment. For example, the following is a complete stub for a CPX requiring 10 `LONG`s of data for permanent storage.

```assembly
.cpxstart:  
jmp _cpx_init
.end
```

XCONTROL Structures

**CPXINFO**

A pointer to a CPX’s `CPXINFO` structure must be returned by the `cpx_init()` function (‘Set Only’ CPX’s return `NULL`). The `CPXINFO` structure is filled in with pointers to user functions as follows:

```c
typedef struct
{
    WORD (*cpx_call)( GRECT *);
    VOID (*cpx_draw)( GRECT *);
    VOID (*cpx_wmove)( GRECT *);
    VOID (*cpx_timer)( WORD *);
    VOID (*cpx_key)( WORD, WORD, WORD * );
    VOID (*cpx_button)( MRETS *, WORD * );
    VOID (*cpx_m1)( MRETS *, WORD * );
    VOID (*cpx_m2)( MRETS *, WORD * );
    WORD (*cpx_hook)( WORD, WORD *, MRETS *, WORD *, WORD * );
    WORD (*cpx_close)( WORD );
} CPXINFO;
```

Form CPX’s use only `cpx_call()` and (optionally) `cpx_close()`. Event CPX’s use the remaining members. Members not being used should be set to `NULL`. 
XCPB

A pointer to the “XControl Parameter Block” is passed to the `cpx_call()` function. This pointer should be copied to a static variable on entry so that other functions may utilize its members. **XCPB** is defined as follows:

```c
typedef struct
{
    WORD    handle;
    WORD    booting;
    WORD    reserved;
    WORD    SkipRshFix;
    VOID    *reserve1;
    VOID    *reserve2;
    VOID    (*rsh_fix)( WORD, WORD, WORD, OBJECT *, TEDINFO *, char *, ICONBLK *, BITBLK *, LONG *, LONG *, VOID * );
    VOID    (*rsh_obfix)( OBJECT *, WORD );
    WORD    (*Popup)( char *items[], WORD, WORD, WORD,
                     GRECT *, GRECT * );
    VOID    (*Sl_size)( OBJECT *, WORD, WORD, WORD, WORD,
                     WORD, WORD );
    VOID    (*Sl_x)( OBJECT *, WORD, WORD, WORD, WORD, WORD, void (*)( ) );
    VOID    (*Sl_y)( OBJECT *, WORD, WORD, WORD, WORD, WORD, void (*)( ) );
    VOID    (*Sl_arrow)( OBJECT *, WORD, WORD, WORD, WORD, void (*)( ) );
    VOID    (*Sl_dragx)( OBJECT *, WORD, WORD, WORD, WORD, void (*)( ) );
    VOID    (*Sl_dragy)( OBJECT *, WORD, WORD, WORD, WORD, void (*)( ) );
    WORD    (*Xform_do)( OBJECT *, WORD, WORD * );
    GRECT   *(GetFirstRect)( GRECT * );
    GRECT   *(GetNextRect)( VOID );
    VOID    *(Set_Event_Mask)( WORD, MOBLK *, MOBLK *, LONG );
    WORD    *(XGen_Alert)( WORD );
    WORD    *(CPX_Save)( VOID *, LONG );
    VOID    *(Set_Buffer)( VOID );
    WORD    *(getcookie)( LONG, LONG * );
    WORD    Country_Code;
    VOID    *(MFSave)( WORD, MFORM * );
} XCPB;
```

Almost all of **XCPB**’s members are pointers to utility functions covered in the **XCONTROL** Function Reference at the end of this chapter. The remaining utilized members have the following meaning:

<table>
<thead>
<tr>
<th>XCPB Member</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>handle</code></td>
<td>This value contains the physical workstation handle returned by <code>graf_handle()</code> to the Control Panel for use in calling <code>v_opnvwk()</code>.</td>
</tr>
<tr>
<td><code>booting</code></td>
<td>When <strong>XCONTROL</strong> is initializing as the result of a power-on, reset, or resolution change, it loads each CPX and calls its <code>cpx_init()</code> function with <code>booting</code> set to <code>TRUE</code>. At all other times, <strong>XCONTROL</strong> sets <code>booting</code> to <code>FALSE</code>.</td>
</tr>
</tbody>
</table>
When a CPX is first called after being loaded, its SkipRshFix flag is set to FALSE. The application should then use xcpb->rsh_fix() to fix its internal resource tree. xcpb->rsh_fix() sets the CPX’s SkipRshFlag to TRUE so that the CPX can skip this step on subsequent calls.

This value indicates the country which this version of the Control Panel was compiled for as follows:

<table>
<thead>
<tr>
<th>Country Code</th>
<th>Country</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>USA</td>
</tr>
<tr>
<td>1</td>
<td>Germany</td>
</tr>
<tr>
<td>2</td>
<td>France</td>
</tr>
<tr>
<td>3</td>
<td>United Kingdom</td>
</tr>
<tr>
<td>4</td>
<td>Spain</td>
</tr>
<tr>
<td>5</td>
<td>Italy</td>
</tr>
<tr>
<td>6</td>
<td>Sweden</td>
</tr>
<tr>
<td>7</td>
<td>Swiss (French)</td>
</tr>
<tr>
<td>8</td>
<td>Swiss (German)</td>
</tr>
<tr>
<td>9</td>
<td>Turkey</td>
</tr>
<tr>
<td>10</td>
<td>Finland</td>
</tr>
<tr>
<td>11</td>
<td>Norway</td>
</tr>
<tr>
<td>12</td>
<td>Denmark</td>
</tr>
<tr>
<td>13</td>
<td>Saudi Arabia</td>
</tr>
<tr>
<td>14</td>
<td>Holland</td>
</tr>
</tbody>
</table>

**CPX Flavors**

**Boot Initialization**

Any CPX which has its _cpxhead.bootinit flag set will have its cpx_init() function called when XCONTROL initializes upon bootup. This provides a way for CPX’s to set system configuration from data the user has saved in previous sessions.

cpx_init() is always called each time the user selects your CPX from the XCONTROL CPX list prior to calling cpx_call(). If the CPX is being initialized at boot time, the xcpb->booting flag will be TRUE.

**Resident CPX’s**

CPX’s which have their _cpxhead.resident flag set will be retained in memory after being initialized at bootup. In general, this option should not be used unless absolutely necessary.

Resident CPX’s should be aware that variables stored in their DATA and BSS segments will not be reinitialized each time the CPX is called.
Set-Only CPX’s
Set-Only CPX’s are designed to initialize system configuration options each time XCONTROL initializes (during boot-ups and resolution changes) by calling the cpx_init() function. These CPX’s will not appear in the XCONTROL list of CPX’s.

Form CPX’s
Every CPX must be either a ‘Form’ or ‘Event’ CPX. Most CPX’s will be Form CPX’s.

In a Form CPX, XCONTROL handles most user-interaction and messaging by relaying messages through a callback function. XCONTROL is responsible for redraws (although the CPX does have a hook to do non-AES object redraws) and form handling. A simple ‘C’ outline for a Form CPX follows:

/* Example Form CPX Skeleton */
#include "skel.h"
#include "skel.rsh"
#include <cpxdata.h>
CPXINFO *cpx_init();
BOOLEAN cpx_call();
XCPB *xcpb;
CPXINFO cpxinfo;
CPXINFO
*cpx_init( Xcpb )  
XCPB *Xcpb;
{
  xcpb = Xcpb;
  appl_init();
  if(xcpb->booting)
  {
    /* CPX’s that do boot-time initialization do it here */

    /* Returning TRUE here tells XCONTROL to retain the header
     * for later access by the user. If CPX is Set-Only,
     * return FALSE. */

    return ( (CPXINFO *) TRUE )
  }
  else
  {
    /* If you haven’t already done so, fix resource tree.
     * DEFINE’s and variables are from an RSH file generated
     * by the Atari Resource Construction Set. */

    if(!SkipRshFix)
(*xcpb->rsh_fix)( NUM_OBS, NUM_FRSTR, NUM_FRIMG, NUM_TREE,
rs_object, rs_tedinfo, rs_strings, rs_iconblk, rs_bitblk,
rs_frstr, rs_frimg, rs_trindex, rs_imdope );

cpxinfo.cpx_call = cpx_call;
cpxinfo.cpx_draw = NULL;
cpxinfo.cpx_wmove = NULL;
cpxinfo.cpx_timer = NULL;
cpxinfo.cpx_key = NULL;
cpxinfo.cpx_button = NULL;
cpxinfo.cpx_m1 = NULL;
cpxinfo.cpx_m2 = NULL;
cpxinfo.cpx_hook = NULL;
cpxinfo.cpx_close = NULL;

/* Tell XCONTROL to send generic and keyboard
 * messages. */

return ( &cpxinfo );
}

BOOLEAN

cpx_call( rect )
GRECT *rect;
{
  /* Put MAINFORM tree in *tree for object macros */

  OBJECT *tree = (OBJECT *)rs_trindex[ MAINFORM ];
  WORD button, quit = FALSE;
  WORD msg[8];

  ObX( ROOT ) = rect->g_x;
  ObY( ROOT ) = rect->g_y;
  objc_draw( tree, ROOT, MAX_DEPTH, PTRS( rect ) );

  do
  {
    button = (*xcpb->Xform_do)( tree, 0, msg );

    /* Be sure and mask off double-clicks if you’re
     * not interested in them. */

    if( ( button & 0x8000 ) && ( button != 0xFFFF ) )
      button &= 0x7FFF;
    button &= 0x7FFF;

    switch( button )
    {
      /* Check for EXIT or TOUCHEXIT resource objects */

      case OK:
        break;
      case CANCEL:
        break;
      case -1:

Event CPX’s

CPX’s which are not possible as Form CPX’s may be designed as Event CPX’s.

Event CPX’s accomplish most of their work in several callback functions identified to the Control Panel by the CPXINFO structure and called when the appropriate message is received. An outline for a typical Event CPX follows:

/* Example Event CPX Skeleton */

#include "skel.h"
#include "skel.rsh"
#include <cpxdata.h>

CPXINFO *cpx_init();
BOOLEAN cpx_call();
void cpx_draw(), cpx_wmove(), cpx_key();

XCPB *xcpb;
CPXINFO cpxinfo;

CPXINFO *
cpx_init( Xcpb )
XCPB *Xcpb;
{
    xcpb = Xcpb;
    appl_init();
    if(xcpb->booting)
    {
        /* CPX’s that do boot-time initialization do it here */

        /* Returning TRUE here tells XCONTROL to retain the header
         * for later access by the user. If CPX is Set-Only,
         * return FALSE.
         */
return ( (CPXINFO *) TRUE )
}
else
{
/* If you haven’t already done so, fix resource tree.
* 
* DEFINE’s and variables are from RSH file generated
* by the Atari Resource Construction Set.
*/

if(!SkipRshFix)
    (*xcpb->rsh_fix)( NUM_OBS, NUM_FRSTR, NUM_FRIMG, NUM_TREE,
        rs_object, rs_tedinfo, rs_strings, rs_iconblk, rs_bitblk,
        rs_frstr, rs_frimg, rs_trindex, rs_imdope );

cpxinfo.cpx_call = cpx_call;
cpxinfo.cpx_draw = cpx_draw;
cpxinfo.cpx_wmove = cpx_wmove;
cpxinfo.cpx_timer = NULL;
cpxinfo.cpx_key = cpx_key;
cpxinfo.cpx_button = NULL;
cpxinfo.cpx_m1 = NULL;
cpxinfo.cpx_m2 = NULL;
cpxinfo.cpx_hook = NULL;
cpxinfo.cpx_close = NULL;

/* Tell XCONTROL to send generic and keyboard
* messages.
*/

(*xcpb->Set_Evnt_Mask)( MU_MESAG | MU_KEYBD, NULL, NULL, -1L );

return ( &cpxinfo );
}
}

BOOLEAN
cpx_call( rect )
GRECT *rect;
{
    /* Put MAINFORM tree in *tree for object macros */

    OBJECT *tree = (OBJECT *)rs_trindex[ MAINFORM ];

    ObX( ROOT ) = rect->g_x;
    ObY( ROOT ) = rect->g_y;

    objc_draw( tree, ROOT, MAX_DEPTH, PTRS( rect ) );

    return ( TRUE );
}

VOID
cpx_draw( rect )
GRECT *rect;
{
    OBJECT *tree = (OBJECT *)rs_trindex[ MAINFORM ];
    GRECT *xrect, rect;

    xrect = (*xcpb->GetFirstRect)( rect );
while( xrect )
{
  rect = *xrect;
  objc_draw( tree, ROOT, MAX_DEPTH, ELTS( rect ) );
  xrect = (*xcpb->GetNextRect)();
}
}

VOID
cpx_wmove( work )
GRECT *work;
{
  OBJECT *tree = (OBJECT *)rs_trindex[ MAINFORM ];
  ObX( tree ) = work->g_x;
  ObY( tree ) = work->g_y;
}

VOID
cpx_key( kstate, key, quit )
WORD kstate, key;
WORD *quit;
{
  /* Substitute case values for values you’re interested
     * in.
     */
   switch( key )
   {
     case KEY_1:
       case KEY_2:
   
}

CPX File Formats

File Naming
Several standard naming conventions for CPX executables and development files follow:

<table>
<thead>
<tr>
<th>File Name</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>*.CPX</td>
<td>Standard CPX ready for execution by the Control Panel.</td>
</tr>
<tr>
<td>*.CP</td>
<td>CPX missing the 512 byte header.</td>
</tr>
<tr>
<td>*_R.CPX</td>
<td>A resident CPX.</td>
</tr>
<tr>
<td>*_S.CPX</td>
<td>A “Set-only” CPX.</td>
</tr>
<tr>
<td>*.HDR</td>
<td>A 512 byte CPX header file.</td>
</tr>
<tr>
<td>*.CPZ</td>
<td>An inactive CPX.</td>
</tr>
<tr>
<td>*.RSH</td>
<td>An “embeddable” resource file. CPX’s can’t execute a rsrc_load() so all resource files must be in this format.</td>
</tr>
</tbody>
</table>
The CPX File Format

A CPX file can be represented graphically as follows:

```
CPX Header Record
(512 bytes)

GEMDOS Executable
Header
(28 bytes)

CPX TEXT Segment
(cpx_init() must begin at
offset 0 of this segment)

CPX DATA Segment
(any data to be saved back
into the CPX must begin at
offset 0 of this segment)

CPX Symbol Table (if any)
```

XCONTROL Function Calling Procedure

Calling Conventions

XCONTROL uses “right–left” stack-based parameter passing for all of its functions and expects that user defined callback functions are similarly “right–left” stack-based. Compilers which do not default to this method should use either the ‘cdecl’ or ‘_stdargs’ keyword depending on your compiler.

Function entry stubs must also consider the longword return code placed on the stack by the 68x00 ‘JSR’ function. ‘C’ compilers always expect this. For example, the pointer to the XCPB passed to the cpx_init() function can be stored through the following machine language statement:

```
   __cpx_init:
   move.l 4(sp),xcpb
```
Stack Space
   CPX programmers should note that all CPX operations use the default Control Panel stack space (2048 bytes) and should therefore restrict heavy usage of automatic variables and other large consumers of stack space.
The XCONTROL callback functions are user-supplied functions which are identified to the Control Panel in the CPXINFO structure returned by the cpx_init() function which is also described in this section. When creating a Form CPX, the only callback function that is utilized is cpx_call(). The remaining functions are used only when creating Event CPX’s. The XCONTROL callback functions are:

- cpx_button()
- cpx_call()
- cpx_close()
- cpx_draw()
- cpx_hook()
- cpx_init()
- cpx_key()
- cpx_m1()
- cpx_m2()
- cpx_timer()
- cpx_wmove()
cpx_button()

VOID (*cpx_button)( mrets, nclicks, event )
MRETS *mrets;
WORD nclicks;
WORD *event;

cpx_button() is called in an Event CPX when a MU_BUTTON event has occurred.

PARAMETERS
mrets points to a structure containing the mouse event which triggered the function as follows:

typedef struct
{
    WORD x;  /* X position of mouse */
    WORD y;  /* Y position of mouse */
    WORD buttons; /* Mask of buttons depressed */
    WORD kstate; /* Keyboard shift state */
} MRETS;

nclicks specifies the number of clicks processed. If this event should terminate the CPX, the function should place a 1 in the WORD pointed to by event.

BINDING

cpxinfo.cpx_button = cpx_button;
return ( &cpxinfo );

COMMENTS
This function will only be called if Set_Evnt_Mask() is called with MU_BUTTON specified as an event to wait for.

cpx_call()

BOOLEAN (*cpx_call)( work )
GRECT *work;

cpx_call() is called immediately after the cpx_init() function when the user activates the CPX.

PARAMETERS
Upon entry, the GRECT structure pointed to by work contains the current rectangular extent of the control panel window work area.

BINDING

cpxinfo.cpx_call = cpx_call;
return ( &cpxinfo );
10.20 – XCONTROL Callback Functions

**RETURN VALUE**

The `cpx_call()` function should return **TRUE** if it wants to continue processing events through the event handlers specified in the **CPXINFO** structure or **FALSE** to indicate the CPX is finished.

**COMMENTS**

When exiting the `cpx_call()` function, the CPX must deallocate any allocated memory and close any **VDI** workstations opened.

---

### cpx_close()

**VOID** (*cpx_close)(**flag**)

**BOOLEAN** **flag**;

`cpx_close()` is called in an Event CPX when a **WM_CLOSED** or **AC_CLOSE** message is received by the control panel.

**PARAMETERS**

**flag** contains **TRUE** if a **WM_CLOSED** message was received or **FALSE** if **AC_CLOSE** was received.

**BINDING**

```c
    cpxinfo.cpx_close = cpx_close;
    return ( &cpxinfo );
```

**COMMENTS**

This function will only be called if `Set_Evnt_Mask()` is called with **MU_MESAG** specified as an event to wait for.

**WM_CLOSED** messages should be treated as equivalent to ‘OK’ whereas **AC_CLOSE** messages should be treated as ‘Cancel’.

---

### cpx_draw()

**VOID** (*cpx_draw)(**clip**)

**GRECT** *clip*;

`cpx_draw()` is called when a **WM_REDRAW** message is received by the control panel in an Event CPX.

**PARAMETERS**

**clip** points to a **GRECT** structure specifying the dirtied area.

**BINDING**

```c
    cpxinfo.cpx_draw = cpx_draw;
    return ( &cpxinfo );
```

**COMMENTS**

This routine should utilize `GetFirstRect()` and `GetNextRect()` to obtain the true rectangles of the area to redraw.
This function will only be called if SetEvt_Mask() is called with MU_MESAG specified as an event to wait for.

**cpx_hook()**

```c
BOOLEAN (*cpx_hook)( event, msg, mrets, key, nclicks )
WORD event;
WORD *msg;
WORD *mrets;
WORD key, nclicks;
```

cpx_hook() is called in an Event CPX immediately after the Control Panel’s evnt_multi() function returns before the message is processed.

**PARAMETERS**

All parameters share counterparts with the evnt_multi() function. For a detailed explanation of the return values please consult the documentation for that function. event contains the event mask of one or more events that occurred. msg points to an array of eight WORDs containing the message buffer. mrets and nclicks point to the mouse event (if any) as described in cpx_button(). key points to a WORD containing the keyboard scancode of the key pressed (if any).

**BINDING**

```c
  cpxinfo.cpx_hook = cpx_hook;
  return ( &cpxinfo );
```

**RETURN VALUE**

The function should return TRUE to override default event handling or FALSE to continue processing the message.

**cpx_init()**

```c
CPXINFO (*cpx_init)( xcpb )
XCPB *xcpb;
```

cpx_init() is called upon bootup and every subsequent time the CPX is opened by the user.

**PARAMETERS**

xcpb points to an XControl Parameter Block structure as described in the beginning of this chapter.

**BINDING**

The cpx_init() function is called by JSR’ing to the first location in the CPX’s TEXT segment. ‘C’ programmers should assemble and link the following code as the first object file in the link to ensure that the correct function is properly called:
### 10.22 – XCONTROL Callback Functions

```
; Startup stub for CPX’s without save area
  .xref  _cpx_init
  .text
cpxstart:
    jmp  _cpx_init
  .end

If the CPX has default data which is to be saved back into the CPX with the `CPX_Save()` function, the following stub should be used (substitute the ‘.dc.w 1’ statement with the appropriate amount of space required to store your data):

```
; Startup stub for CPX’s with save area
  .xref  _cpx_init
  .globl  _save_vars
  .text
cpxstart:
    jmp  _cpx_init
  .data
  _save_vars:
    .dc.w  1
  .end
```

**RETURN VALUE**
The `cpx_init()` function returns a pointer to its `CPXINFO` structure to allow the Control Panel to access its other routines. If it is a ‘Set-Only’ CPX, it should return `NULL`.

**COMMENTS**
A CPX can distinguish when a CPX is booting by checking the `xcpb->booting` structure member.

It is recommended that the CPX to create a copy of `xcpb` each time `cpx_init()` is called for the other callback functions to utilize.

---

**cpx_key()**

```c
VOID (*cpx_key)( kstate, key, event )
WORD kstate;
WORD key;
WORD *event;
```

cpx_key() is called in an Event CPX when a `MU_KEYBD` event has occurred.
PARAMETERS  

$kstate$ specifies the state of the keyboard shift keys as in $\text{evnt_keybd}()$. $\text{key}$ specifies the keyboard scan code of the key struck. The $\text{WORD}$ pointed to by $\text{event}$ should be filled in with a 1 if this event should terminate the CPX.

BINDING  

\[
\text{cpxinfo.cpx_key} = \text{cpx_key};
\]

\[
\text{return ( \&cpxinfo );}
\]

COMMENTS  

This function will only be called if Set_Evnt_Mask() is called with $\text{MU_KEYBD}$ specified as an event to wait for.

cpx_m1()

VOID (*cpx_m1)($mrets, event$)
MRETS *$mrets$;
WORD $event$;

cpx_m1() is called when a $\text{MU_M1}$ event has occurred in an Event CPX.

PARAMETERS  

$mrets$ will contain a pointer to a MRETS structure as specified in $\text{cpx_button}()$ which contains the mouse state as it satisfied the condition. The $\text{WORD}$ pointed to by $\text{event}$ should be filled in with 1 if this event should terminate the CPX.

BINDING  

\[
\text{cpxinfo.cpx_m1} = \text{cpx_m1};
\]

\[
\text{return ( \&cpxinfo );}
\]

COMMENTS  

This function will only be called if Set_Evnt_Mask() is called with $\text{MU_M1}$ specified as an event to wait for.

SEE ALSO  

cpx_m2()

cpx_m2()

VOID (*cpx_m2)($mrets, event$)
MRETS *$mrets$;
WORD $event$;

cpx_m2() is called when a $\text{MU_M2}$ event has occurred in an Event CPX.

PARAMETERS  

See cpx_m1().

BINDING  

\[
\text{cpxinfo.cpx_m2} = \text{cpx_m2};
\]

\[
\text{return ( \&cpxinfo );}
\]
This function will only be called if `Set_Evnt_Mask()` is called with `MU_M2` specified as an event to wait for.

**cpx_timer()**

```c
VOID (*cpx_timer)( event )
WORD *event;
```

cpx_timer() is called when a `MU_TIMER` event has occurred in an Event CPX.

**Parameters**

The `WORD` pointed to by `event` should be filled in with 1 if this event should terminate the CPX.

**Binding**

```c
cpxinfo.cpx_timer = cpx_timer;
return ( &cpxinfo );
```

**Comments**

This function will only be called if `Set_Evnt_Mask()` is called with `MU_TIMER` specified as an event to wait for.

**cpx_wmove()**

```c
VOID (*cpx_wmove)( work )
GRECT *work;
```

cpx_wmove() is called when a `WM_MOVED` message is received by the Control Panel in an Event CPX.

**Parameters**

`work` is a pointer to a `GRECT` containing the new coordinates of the window work area.

**Binding**

```c
cpxinfo.cpx_wmove = cpx_wmove;
return ( &cpxinfo );
```

**Comments**

This function will only be called if `Set_Evnt_Mask()` is called with `MU_MESAG` specified as an event to wait for.
The **XCONTROL** utility functions are accessed via the **XCPB** (XControl Parameter Block) in the following format for users of ‘C’:

\[
    \text{ret} = (*\text{xcpb->Function})( \text{param1, param2, ... })
\]

These functions provide functions useful mostly to CPX’s as well as functions that closely resemble **AES** functions better suited for CPX’s. The **XCONTROL** Utility Functions are:

- (*xcpb->CPX_Save)()
- (*xcpb->Get_Buffer)()
- (*xcpb->getcookie)()
- (*xcpb->GetFirstRect)()
- (*xcpb->GetNextRect)()
- (*xcpb->MFsave)()
- (*xcpb->Popup)()
- (*xcpb->rsh_fix)()
- (*xcpb->rsh_obfix)()
- (*xcpb->Set_Evnt_Mask)()
- (*xcpb->Sl_arrow)()
- (*xcpb->Sl_dragx)()
- (*xcpb->Sl_dragy)()
- (*xcpb->Sl_size)()
- (*xcpb->Sl_x)()
- (*xcpb->Sl_y)()
- (*xcpb->Xform_do)()
- (*xcpb->XGen_Alert)()
(\(*\text{xcpb}->\text{CPX\_Save}\)())

\begin{verbatim}
BOOLEAN (\*\text{xcpb}->\text{CPX\_Save})(\ ptr,\ num );
VOIDP \text{ptr};
LONG \text{num};
\end{verbatim}

\text{CPX\_Save()} writes the specified data to the CPX on disk at the beginning of the CPX's DATA segment.

**PARAMETERS**  
\text{ptr} is a pointer to the data to save. \text{num} specifies the length of the data in bytes.

**BINDING**  
(\*\text{xcpb}->\text{CPX\_Save})(\ \text{ptr},\ \text{num } );

**RETURN VALUE**  
\text{CPX\_Save()} returns \text{TRUE} if the operation was successful or \text{FALSE} if an error occurred.

**COMMENTS**  
\text{CPX\_Save()} stores the specified data on disk in the original CPX file at the start of the DATA segment of the program. For this reason, enough space should be allocated to account for this data. See \text{cpx\_init()} for an example method of accomplishing this.

**SEE ALSO**  
(\*\text{xcpb}->\text{Get\_Buffer}())

(\*\text{xcpb}->\text{Get\_Buffer}())

\begin{verbatim}
VOIDP (\*\text{xcpb}->\text{Get\_Buffer})(\ \text{VOID } )
\end{verbatim}

\text{Get\_Buffer()} returns the address of a 64-byte static storage location for the calling CPX.

**BINDING**  
bufptr = (\*\text{xcpb}->\text{Get\_Buffer}());

**RETURN VALUE**  
\text{Get\_Buffer()} returns a pointer to a 64-byte static storage location which can be used by the CPX to preserve data between invocations.

**COMMENTS**  
Data stored in this area is lost upon a reboot. Permanent data should be stored using \text{CPX\_Save()}.

**SEE ALSO**  
(\*\text{xcpb}->\text{CPX\_Save}())
(*xcpb->getcookie)()

WORD (*xcpb->getcookie)(cookie, pvalue)
LONG cookie;
LONG *pvalue;

getcookie() searches the ‘cookie jar’ for a given cookie and if found returns its stored longword.

PARAMETERS cookie contains the longword cookie (usually a packed 4 character ASCII code) to search for. If found, the value of the cookie is placed in the LONG pointed to by pvalue.

BINDING err = (*xcpb->getcookie)(cookie, pvalue);

RETURN VALUE getcookie() returns TRUE if the value placed in pvalue is valid or FALSE if the cookie was not found.

COMMENTS This function is useful in locating TSR’s or other resident processes which a CPX is designed to configure.

(*xcpb->GetFirstRect)()

GRECT *(xcpb->GetFirstRect)(prect)
GRECT *prect;

GetFirstRect() returns the first member of the Control Panel’s rectangle list intersected by prect.

PARAMETERS prect points to a GRECT containing the extent of the dirtied area.

BINDING rdraw = (*xcpb->GetFirstRect)(prect);

RETURN VALUE GetFirstRect() will return a pointer to a GRECT containing the first intersecting rectangle to redraw or NULL if none of the CPX’s rectangles intersect the dirtied area.

COMMENTS Xform_do() handles resource object redraws in Form CPX’s. Other objects requiring a redraw in Form CPX’s and all objects in Event CPX’s must be redrawn with using these functions when a redraw message is generated.

SEE ALSO (*xcpb->GetNextRect)()
(*xcpb->GetNextRect)()

GRECT *(*xcpb->GetNextRect)( VOID )

GetNextRect() returns subsequent rectangles needing to be redrawn after first calling GetFirstRect().

**Binding**

rdraw = (*xcpb->GetNextRect)();

**Return Value**

GetNextRect() returns a pointer to a GRECT structure containing a subsequent rectangle needing to be redrawn.

**Comments**

When a redraw message is received, it should be handled as illustrated below (the example given is for an Event CPX but it may be applied to the WM_REDRAW message handling section of a Form CPX as well):

VOID
cpx_draw( clip )
GRECT *clip;
{
    GRECT *rdraw;
    rdraw = (*xcpb->GetFirstRect)( clip );
    while( rdraw )
    {
        /* User redraw function */
        my_redraw( rdraw );
        rdraw = (*xcpb->GetNextRect)();
    }
}

**See Also**

(*xcpb->GetFirstRect)()

(*xcpb->MFsave)()

VOID (*xcpb->MFsave)( flag, mf )
BOOLEAN flag;
MFORM *mf;

MFsave() saves the current mouse form so that a custom application mouse form is not destroyed when the CPX calls graf_mouse() or vsc_form() to change the shape of the mouse.

**Parameters**

flag specifies the action to take. If flag is MFSAVE (1), the current mouse form will be written into the MFORM structure pointed to by mf. If flag is MFRESTORE (0), the mouse form will be restored from the MFORM structure.
pointed to by mf. See vsc_form() for the definition of MFORM.

**Binding**

```
(*xcpb->MFsave)( flag, mf );
```

---

**(*xcpb->Popup)()**

```c
WORD (*xcpb->Popup)( items, num_items, default, font, button, world );
```

**PARAMETERS**

- `items` points to an array of character pointers pointing to the text of the items. Each string must be padded in front with at least 2 spaces and should be of equal length (at least as long as the longest string). `num_items` specifies the number of items to display in the popup. If `num_items` exceeds five, the popup will only show three items with two arrows to allow scrolling.

- `default` indicates the default item (the default item is displayed with a checkmark) or -1 to indicate no default item.

- `font` specifies the font size (3 = large, 5 = small) of the items in the popup.

- `button` points to a GRECT containing the rectangular extent of the button pressed to call the popup. `world` points to a GRECT containing the current extent of the CPX work area.

**Binding**

```
ret = (*xcpb->Popup)( items, num_items, default, font, button, world );
```

**RETURN VALUE**

`Popup()` returns the item selected (0 based) or -1 if no selection was made (the user clicked outside of the popup area).

**COMMENTS**

This function is unique to CPX’s and is not the same as menu_popup().

Button objects which are to be used as popups should be TOUCHEXIT objects. In addition, as a matter of style, popup buttons should be SHADOWED.
(*xcpb->rsh_fix)()

VOID (*xcpb->rsh_fix)( num_objs, num_frstr, num_frimg, num_tree, rs_object, rs_tedinfo, 
rs_strings, rs_iconblk, rs_bitblk, rs_frstr, rs_frimg, rs_trindex, rs_imdope );

WORD num_objs, num_frstr, num_frimg, num_tree;
OBJEKT *rs_object;
TEDINFO *rs_tedinfo;
char *rs_strings[];
ICONBLK *rs_iconblk;
BITBLK *rs_bitblk;
LONG *rs_frstr, *rs_frimg, *rs_trindex;
struct foobar *rs_imdope;

rsh_fix() fixes up a resource tree in memory based on an 8x16 character font.

PARAMETERS When using the Atari Resource Construction Set the parameters are generated in
the .RSH file created by the compiler.

When using other resource construction sets you should refer to their instructions
for applying their resource structure to this function or use the CPX function
rsh_obfix() on each OBJECT.

BINDING (xcpb->rsh_fix)( num_objs, num_frstr, num_frimg, num_tree, 
rs_object, rs_tedinfo, rs_strings, rs_iconblk, rs_bitblk, 
rs_frstr, rs_frimg, rs_trindex, rs_imdope );

COMMENTS rsrc_load(), rsrc_obfix(), and rsrce_rcfix() fix up a resource file based upon the
current screen character size. CPX resource data is always fixed up based upon an
8x16 character font.

Resources should be designed on a screen that supports an 8x16 ratio. When using
the Atari Resource Construction Set, the resource should be designed as a ‘Panel’
rather than a ‘Dialog’. With other resource construction applications the same
effect is achieved by turning snap off.

Resources should only be fixed up when the xcpb->SkipRshFix flag is 0. This
prevents resources from being fixed up more than once.

SEE ALSO (*xcpb->rsh_obfix)()
(*xcpb->rsh_obfix)()

VOID (*xcpb->rsh_obfix)( tree, curob )
OBJECT *tree;
WORD curob;

rsh_obfix() converts the specified object from character to pixel based coordinates based on an 8x16 character font.

PARAMETERS

* tree points to the OBJECT tree which contains the object curob to fix up.

BINDING

(*xcpb->rsh_obfix)( tree, curob );

COMMENTS

See rsh_fix().

SEE ALSO

(*xcpb->rsh_fix())

(*xcpb->Set_Evnt_Mask)()

VOID (*xcpb->Set_Evnt_Mask)( mask, m1, m2, time )
WORD mask;
MOBLK *m1;
MOBLK *m2;
LONG time;

Set_Evnt_Mask() defines which events an Event CPX will process with its callback functions.

PARAMETERS

* mask is a bit mask of events (MU_MESAG, MU_TIMER, etc...) that the CPX wishes to process as in evnt_multi(). m1 and m2 point to MOBLK structures which define mouse rectangles to wait for if the CPX wishes to wait for MU_M1 and/or MU_M2 events as in evnt_mouse(). MOBLK is defined as follows:

```c
typedef struct {
    WORD m_out; /* 0 = enter, 1 = exit */
    WORD m_x;
    WORD m_y;
    WORD m_w;
    WORD m_h;
} MOBLK;
```

* time specifies the length of time to specify for the MU_TIMER event if appropriate.
(*)xcpb->Sl_arrow()

VOID (*xcpb->Sl_arrow)( tree, base, slider, obj, inc, min, max, numvar, dir, foo )
OBJECT *tree;
WORD base, slider, obj, inc, min, max;
WORD *numvar;
WORD dir;
VOID (*foo)();

(*)xcpb->Sl_arrow is called by a CPX when the user clicks on an arrow element of an ‘active’ slider.

PARAMETERS

tree points to the object tree containing the slider elements. base is the object index of the slider ‘track’. slider is the object index of the slider ‘elevator’. obj is the index of the arrow element clicked on by the user.

inc specifies the increment amount for each slider step (+/-). min specifies the minimum value the slider can represent. max specifies the maximum value the slider can represent.

numvar points to a WORD containing the value which the slider represents and which is to be updated as the slider is moved. dir specifies the direction of the slider movement (VERTICAL (0) or HORIZONTAL (1)).

foo is a pointer to a user-defined callback function which is called once for each step of the slider to allow the user’s action to ‘actively’ update the slider. foo may be NULL if no updating is desired.

BINDING

(*xcpb->SetEvtMask)( mask, m1, m2, time );

COMMENTS

This function is only valid for Event CPX’s.

Slider paging can be accomplished with this function. To do so use a method similar to the following (this example is for vertical sliders):

graf_mkstate( &mx, &my, &dum, &dum );
objc_offset( tree, slider, &ox, &oy );
inc = ( ( my < oy ) ? ( -1 ) : ( 1 ) );
(*xcpb->Sl_arrow( tree, base, slider, base, inc, min, max, &numvar, VERTICAL, foo );

THE ATARI COMPENDIUM
(*)xcpb->Sl_dragx()() 

VOID (*xcpb->Sl_dragx)( tree, base, slider, min, max, numvar, foo )
OBJECT *tree;
WORD base, slider, min, max;
WORD *numvar;
VOID (*foo)();

Sl_dragx() is called by a CPX when a user clicks on the horizontal slider ‘elevator’ of an ‘active’ slider.

PARAMETERS

tree points to an OBJECT tree containing the slider elements. base is the object index of the slider ‘track’. slider is the object index of the slider ‘elevator’.

min specifies the minimum value the slider can represent. max specifies the maximum value the slider can represent.

numvar points to a WORD containing the value which the slider represents and which is to be updated as the slider is moved.

foo points to a user-defined routine which is called each time the slider value numvar is modified. foo may be NULL if no updating is desired.

BINDING

(*xcpb->Sl_dragx)( tree, base, slider, min, max, numvar, foo );

COMMENTS

It is appropriate to change the shape of the mouse to FLAT_HAND while the user is dragging a slider.

SEE ALSO

(*xcpb->Sl_dragy)()

(*)xcpb->Sl_dragy()() 

VOID (*xcpb->Sl_dragy)( tree, base, slider, min, max, numvar, foo )
OBJECT *tree;
WORD base, slider, min, max;
WORD *numvar;
VOID (*foo)();

Sl_dragy() is called by a CPX when a user clicks on the vertical slider ‘elevator’ of an ‘active’ slider.

PARAMETERS

See Sl_dragx().
(*xcpb->Sl_size)() – 10.35

**BINDING**

`(*xcpb->Sl_dragy)( tree, base, slider, min, max, numvar, foo );`

**COMMENTS**

It is appropriate to change the shape of the mouse to **FLAT_HAND** while the user is dragging a slider.

**SEE ALSO**

`(*xcpb->Sl_dragx)()`

(*xcpb->Sl_size)()

**VOID** (*xcpb->Sl_size)( tree, base, slider, num_items, visible, direction, min_size )

**OBJECT** *tree;**

**WORD** base, slider, num_items, visible, direction, min_size ;

SL_size() adjusts the size of the slider ‘track’ relative to the size of the slider ‘elevator’.

**PARAMETERS**

- `tree` points to the **OBJECT** tree containing the slider elements. `base` is the object index of the slider ‘track’. `slider` is the object index of the slider ‘elevator’.

- `num_items` is the total number of items represented by the slider. `visible` is the number of items actually seen by the user.

- `direction` specifies the direction of the slider as either **VERTICAL** (0) or **HORIZONTAL** (1). `min_size` represents the minimum pixel size of the adjusted slider elevator.

**BINDING**

`(*xcpb->Sl_size)( tree, base, slider, num_items, visible, direction, min_size );`

**COMMENTS**

This function does not redraw the slider.

(*xcpb->Sl_x)()

**VOID** (*xcpb->Sl_x)( tree, base, slider, value, min, max, foo )

**OBJECT** *tree;**

**WORD** base, slider, value, min, max;

**VOID** (*foo)();

SL_x() updates the position of a horizontal slider within its base.

**PARAMETERS**

- `tree` points to an **OBJECT** tree containing the slider elements. `base` is the object index of the slider ‘track’. `slider` is the object index of the slider ‘elevator’.
value is the value the slider should represent. min and max are the minimum and maximum values the slider can represent respectively.

If foo is not NULL, it points to a user-function which is called to redraw the slider.

**BINDING**

(*xcpb->Sl_x)( tree, base, slider, value, min, max, foo );

**SEE ALSO**

(*xcpb->Sl_y)()

## (*xcpb->Sl_y)()

VOID (*xcpb->Sl_y)( tree, base, slider, value, min, max, foo )

OBJECT *tree;

WORD base, slider, value, min, max;

VOID (*foo)();

Sl_y() updates the position of a vertical slider within its base.

**PARAMETERS**

See Sl_x().

**BINDING**

(*xcpb->Sl_y)( tree, base, slider, value, min, max, foo );

**SEE ALSO**

(*xcpb->Sl_x)()

## (*xcpb->Xform_do)()

WORD (*xcpb->Xform_do)( tree, editobj, msg )

OBJECT *tree;

WORD editobj;

WORD *msg;

Xform_do() is a specialized version of form_do() designed to handle a CPX object tree and window messages concurrently.

**PARAMETERS**

*tree should point to an OBJECT tree containing a form with the root object being 256x176. editobj specifies the editable text object to initially display the text cursor at (or 0 if no editable object exists on the form).

*msg should point to an 8 WORD array used by the function to store special messages returned by evnt_multi().
**Binding**

\[
\text{ret = (*xcpb->Xform_do)( tree, editobj, msg );}
\]

**Return Value**

`Xform_do()` returns the positive object number of the EXIT or TOUCHEXIT object selected. The high bit of this value indicates if the object was double-clicked and should therefore be masked off if unused. If `Xform_do()` returns a -1, then a message should be processed as contained in `msg`. The structure of messages are the same as in `evnt_multi()`. Possible messages are:

- WM_REDRAW
- AC_CLOSE
- WM_CLOSE
- CT_KEY

`CT_KEY` (53) is a special XCONTROL message indicating that a key was pressed. The scancode of the key pressed is contained in `msg[3]`. Only special keyboard keys such as HELP, F1–F10, UNDO, ALT-X, etc. will be returned as the standard alphabetic keys are processed in editable fields.

**Comments**

The `Xform_do()` function automatically handles and redraws of the given OBJECT tree. Any other items needing to be redrawn should be handled at the appropriate window redraw message.

`WM_CLOSED` messages should always be treated as ‘OK’ while `AC_CLOSE` messages should be treated as ‘Cancel’.

---

\*[xcpb->XGen_Alert]() – 10.37

**boolean** (*xcpb->XGen_Alert)( *id*)

```c
WORD id;

XGen_Alert() displays a specialized alert centered in the Control Panel’s work area.

**Parameters**

`id` specifies the alert to display as follows:

<table>
<thead>
<tr>
<th>Name</th>
<th>id</th>
<th>Alert</th>
</tr>
</thead>
<tbody>
<tr>
<td>SAVE_DEFAULTS</td>
<td>0</td>
<td>Save Defaults?</td>
</tr>
<tr>
<td>MEM_ERR</td>
<td>1</td>
<td>Memory Allocation Error</td>
</tr>
<tr>
<td>FILE_ERR</td>
<td>2</td>
<td>File I/O Error</td>
</tr>
<tr>
<td>FILE_NOT_FOUND</td>
<td>3</td>
<td>File Not Found Error</td>
</tr>
</tbody>
</table>

**Binding**

\[
\text{ret = (*xcpb->XGen_Alert)( id );}
\]

**The Atari Compendium**
**RETURN VALUE**  
XGen Alert() returns **TRUE** if ‘OK’ was selected or **FALSE** if ‘Cancel’ was selected. Alerts 1-3 always returns **TRUE**.
GEM USER INTERFACE GUIDELINES
Overview

Maintaining consistent elements of style in a user-interface is an important aspect of programming which should not be overlooked. An extremely powerful application will have its usefulness compromised by an interface that is unlike the majority of other applications a user will be exposed to.

In an effort to create a more standardized method of application programming, this reference will diagram many interface elements that every Atari programmer should use, regardless of whether you are applying them to existing parts of GEM or programmer-defined elements.

In a case where you provide an enhanced interface element that departs from these specifications, you should at least allow the user to disable the option in a ‘Settings...’ dialog.

The Basics

All GEM applications should contain a menu bar providing access to program features. Desk accessories should appear in a window.

‘Dialogware’ and ‘Alertware’ applications are strongly discouraged. Each performs user interaction exclusively in one or more dialogs or alerts respectively. This makes it impossible for the user to take advantage of other programs or desk accessories while in use.

Document-oriented applications that are launched with one or more valid documents specified on the command line should launch those documents into their own windows, otherwise the application should initialize in one of two other ways:

- Open an empty document window with the default parameters labeled “Untitled.”
- Present a dialog allowing three choices. “New” opens a blank document (as above), “Open” presents a file selector used to select a document to open, “Cancel” removes the dialog and leaves the user with the menu bar to make other selections.

Windows

A window is a viewport through which all or part of an application’s document may be viewed. Windows are modeless forms of input. This means that they do not restrict the user from switching to another window or executing a command.

Normal document windows should have a title bar and should be moveable (these characteristics are set with the wind_create() function – see Chapter 6: AES ). The following illustration shows a window with all window components identified:
Here are some other basic rules to use when creating windows:

- Windows should almost always have the **MOVE** characteristic set.
- If it is possible that the contents of the information displayed in the window might overflow, provide sliders (horizontal and/or vertical) as appropriate. The sliders should be updated as necessary to ensure that they are proportional in size and position to the amount of information viewable in the window versus the size of the entire document.
- Generally, all document windows will include all window elements (with the possible exception of the information line). Only exclude an element if its use would be inappropriate in the current context.
Window Messages

An application’s use of windows depends on either the `evnt_mesag()` or `evnt_multi()` functions of the AES. These functions return messages which in turn must be responded to by the application for any changes to occur. The following list illustrates all messages that a window may receive along with an appropriate action(s) that should be taken.

<table>
<thead>
<tr>
<th>Message</th>
<th>Action</th>
</tr>
</thead>
<tbody>
<tr>
<td>WM_REDRAW</td>
<td>Redraw the rectangular portion of the window which was dirtied (as specified in the message). Always use <code>wind_get()</code> with <code>WF_FIRSTXYWH</code> and <code>WF_NEXXTXYWH</code> to walk the rectangle list and enable clipping to the appropriate regions. If the window had a SMALLER gadget, check prior to drawing whether you are drawing the actual window contents or an iconified representation. If the window has an attached toolbar that requires special redrawing, use <code>wind_get()</code> with <code>WF_FTOOLBAR</code> and <code>WF_NTOOLBAR</code> as parameters to walk the rectangle list and enable clipping to the returned regions. In some situations you may want to redraw the entire window upon each WM_REDRAW call. You must still walk the rectangle list as specified above.</td>
</tr>
<tr>
<td>WM_TOPPED</td>
<td>Call <code>wind_set()</code> with a parameter of <code>WF_TOP</code> to actually top the window. Do not redraw the window. Your application will receive WM_REDRAW messages for portions of the window uncovered by the call. Also, set the mouse form as desired.</td>
</tr>
<tr>
<td>WM_SIZED</td>
<td>Call <code>wind_set()</code> with a parameter of <code>WF_CURRXYWH</code> to actually change the current size of the window. Update slider positions as necessary to reflect the new size of the window. Applications will automatically receive a redraw message if any portion of the window was uncovered. If you need to redraw the entire window each time the window size changes, send your own application a WM_REDRAW message with <code>appl_write()</code> to cause a redraw.</td>
</tr>
<tr>
<td>WMMOVED</td>
<td>Call <code>wind_set()</code> with a parameter of <code>WF_CURRXYWH</code> to actually change the current size of the window. This message and the message WM_SIZED are usually handled by common code.</td>
</tr>
<tr>
<td>Message</td>
<td>Description</td>
</tr>
<tr>
<td>--------------</td>
<td>-----------------------------------------------------------------------------</td>
</tr>
<tr>
<td>WM_ARROWED</td>
<td>Scroll the contents of the document window as necessary and redraw the window (using the rectangle list). When an arrow indicator is clicked, scroll the window by one 'line' (a small increment in a non-text oriented application). When the exposed area of the slider bar is clicked, scroll the contents of the document window by one 'page' (current viewable portion of the document) minus one 'line'.</td>
</tr>
<tr>
<td>WM_VSLID</td>
<td>Scroll the contents of the document window in proportion with the new position of the slider elevator.</td>
</tr>
<tr>
<td>WM_HSLID</td>
<td>Scroll the contents of the document window in proportion with the new position of the slider elevator.</td>
</tr>
<tr>
<td>WM_FULLED</td>
<td>Restore the size of the window using wind_get() with a parameter of WF_PREVXYWH. Update slider bars as necessary.</td>
</tr>
<tr>
<td>WM_CLOSED</td>
<td>Close the window. If the window context required a positive or negative answer from the user ('Yes/No' or 'OK/Cancel'), assume positive. If the window contains a document which has been altered since the last time it was saved to disk, it is appropriate to ask the user if the document should be saved before proceeding.</td>
</tr>
<tr>
<td>WM_BOTTOMED</td>
<td>Call wind_set() with a parameter of WF_BOTTOM to send the window to the bottom of the window stack.</td>
</tr>
<tr>
<td>WM_ICONIFY</td>
<td>See below.</td>
</tr>
<tr>
<td>WM_UNICONIFY</td>
<td>See below.</td>
</tr>
<tr>
<td>WM_ALLICONIFY</td>
<td>See below.</td>
</tr>
<tr>
<td>WM_TOOLBAR</td>
<td>Respond as necessary to the toolbar event.</td>
</tr>
<tr>
<td>WM_ONTOP</td>
<td>Set the mouse form appropriately for your application.</td>
</tr>
<tr>
<td>WM_UNTOPPED</td>
<td>No action is mandated by this message.</td>
</tr>
</tbody>
</table>
Clipping Rectangles

In every instance where text or graphics are rendered in a window, you should walk the rectangle list in order to ensure that the screen is properly updated. This includes all instances when the contents of the window are updated as a response to a user command (as opposed to a WM_REDRAW message) or dynamic interaction (i.e. selection or animation).

Window Titles

The title bar of a window should accurately reflect its basic contents. If a window contains a document the title bar should contain the filename of the document or ‘Untitled’ if it is a new document that has not been saved yet. If the window does not contain a document, the title bar should serve to clearly explain the purpose of the menu. For example, if you were to implement a find and replace dialog in a window, the window should be titled “Find & Replace.”

In some cases you may wish to provide an option (though a menu or keystroke) which allows the user to open a duplicate copy of the document in another window. This allows the user to select separate views in each open window yet changes in one window are reflected in others. In this case, suffix the document name with a colon and the window number such as “FILENAME.DOC:1”. The numbering should only be present when more than one document window actually exists.

Iconified Windows

AES versions 4.1 and above support the SMALLER gadget for window iconification. The basic rules for iconification follow:

<table>
<thead>
<tr>
<th>Action</th>
<th>Is a ‘program group’ iconified window already open?</th>
<th>Response</th>
</tr>
</thead>
<tbody>
<tr>
<td>User wishes to iconify a single window.</td>
<td>No</td>
<td>Iconify the single window.</td>
</tr>
<tr>
<td>User wishes to iconify a single window.</td>
<td>Yes</td>
<td>Close the window the user wishes to iconify and add it to those represented by the ‘program group’ window.</td>
</tr>
<tr>
<td>User wishes to iconify all windows.</td>
<td>No</td>
<td>Create a new, iconified window as a ‘program group’ and close all other windows.</td>
</tr>
<tr>
<td>User wishes to iconify all windows.</td>
<td>Yes</td>
<td>Add all open windows to those represented by the ‘program group’ window and close all other windows.</td>
</tr>
<tr>
<td>User wishes to uniconify a single window.</td>
<td>N/A</td>
<td>Uniconify the window.</td>
</tr>
<tr>
<td>User wishes to uniconify a ‘program group’ window.</td>
<td>Yes</td>
<td>Close the iconified window and open all of the windows in the ‘program group’.</td>
</tr>
</tbody>
</table>

Here are some other hints that are helpful when dealing with iconification:

- Due to the smaller size of the window title line, it may be desirable to adjust the title text when a window is iconified.
• Draw an icon which represents the contents of the window when drawing a single iconified window. When drawing a ‘program group’ iconified window, draw an icon which represents the application.

• Use `graf_growbox()` and `graf_shrinkbox()` to graphically show the user the iconification/uniconification process.

Window Information Line

When appropriate, the addition of the INFO component of a window should serve to provide additional information about the objects visible in the window. This information should change to provide the most useful information. A vector graphics editor might display the document size, statistics, and zoom factor normally, but provide information on the number and extent of selected objects when at least one object is selected.

Window Colors

AES versions 3.0 and above allow the color of each window component to be modified. An application should never modify the global settings. Allow the user to use the Window Colors CPX to choose global colors of his/her choice.

If your application wants to draw a visual distinction between windows by displaying them in different colors, provide a dialog where the user may choose color preferences or (at least) enable/disable this option.

Dialog Boxes

A dialog box is the modal counterpart to a window. When a dialog box is displayed, all of the user’s input is exclusively directed towards it until the user releases control by satisfying the needs of the dialog. Here are some basic rules regarding dialog boxes:

• Prior to drawing a dialog and calling `form_do()`, call the AES function `wind_update(BEG_UPDATE)`. Do not release control with `END_UPDATE` until the dialog box is removed and input with it is finished.

• If a dialog box controls a physical attribute (such as text face or fill type), provide a ‘Sample’ area where changes are automatically displayed prior to exiting the dialog.

• Dialogs that position themselves automatically at the center of the active window or mouse location are convenient to some users, annoying to others. When providing this feature, allow it to be disabled.

Button Positioning

Most dialogs consist of several resource objects that can be edited or changed by the user and several exit buttons which terminate the dialog (or cause a supplementary action). Dialogs which supply information should have an ‘OK’ button and a ‘Help’ button if additional information is available. Dialogs which manipulate settings should have an ‘OK’ button to accept changes, a ‘Cancel’ button to revert to the state prior to entering the dialog, and an ‘Help’ button if help is to be provided.
Buttons should always appear in the order ‘OK’, ‘Cancel’, ...other buttons..., ‘Help’ when working left to right or top to bottom. ‘OK’ should be in all capitals. All other buttons should be capitalized. When other wording is appropriate (such as ‘Yes/No’) the positive answer should always precede the negative answer.

All dialogs should have a default exit button which exits the dialog. In most cases this will be the positive ‘OK’ or ‘Yes’ response. In a case where an action is irreversible and data will be changed (for example, formatting a disk), it is appropriate for the negative response to be made default rather than the positive one.

Exit buttons should be placed in a dialog so that they are either centered at the bottom of the dialog or listed from top to bottom starting at the upper-righthand corner of a dialog as pictured in the following diagrams:

![Dialog with Horizontal Buttons]

![Dialog with Vertical Buttons]

When using the ‘top-down’ style, buttons with complementary meanings may be grouped by inserting one space between groups. The dialog pictured above shows an example of a dialog with an ‘OK’, ‘Cancel’, and ‘Help’ button correctly positioned.

**Unfolding Dialogs**

In some cases a dialog may contain features for both the common and advanced user. In this case it is recommended that an ‘unfolding’ dialog be presented.

An unfolding dialog contains a button such as ‘Options >>’ or ‘More >>’ which, when pressed, expands the dialog to reveal additional features. When this happens the ‘Options >>’ button...
becomes ‘<< Options’ (or ‘More >>’) becomes ‘<< Less’ which, when pressed, will return the dialog box to its original state.

**User-Defined Controls.**

When adding custom objects to dialog boxes using G_PROGDEF objects or other means, it is important to keep the interface with these objects consistent with an already existing object. For instance, a custom text control should respond to keystrokes in the same manner as the G_FTEXT object. If a custom object departs from these standards, its implementation should be capable of being disabled.

**Alerts**

Alerts are special dialog boxes which provide information and/or a limited choice of options to the user. Alerts are often used to present an error condition to the user or to inform them of a choice. Some basic rules regarding alert boxes follow:

- In general apply rules regarding button text (such as capitalization, the default object, etc.) to alerts.
- Whenever possible, provide the user with more than one option in an alert box. Alerts with only one button are frustrating and should only be used when only one possible course of action exists.
- Never provide an ‘OK’ button and a ‘Cancel’ button when either button will lead to the same action/inaction.
- Avoid using the word ‘error’ or any other text which might blame the user.
- If an error has occurred, suggest a remedy (possibly using a dialog box for data reentry).
- Use ‘Cannot’ instead of ‘Can’t’ or ‘Can not’.
- If an error alert might occurring during multi-tasking while another process has focus, make the first line of the alert text the program name followed by a colon.
- A message such as “Not enough memory to load file TEST.DOC.” is much better than “Insufficient memory.”
- Minor warnings to a user might become increasingly apparent by having the response to the first incorrect action be the system bell and the second occurrence being a dialog box politely guiding the user along.
- Message text should be left-aligned.
- If message text is too long to fit into the 5 line/30 character per line limit, consider downsizing the message for clarity, or if necessary, place the alert in a form. Never use consecutive alerts.
- Alerts should be capitalized by standard grammatical rules and should be punctuated with a period or question mark (not an exclamation mark).
Alerts boxes may be displayed with one of three icons (or no icon at all). The following lists examples of when to use a specific icon:

<table>
<thead>
<tr>
<th>Icon</th>
<th>Uses</th>
</tr>
</thead>
<tbody>
<tr>
<td>None</td>
<td>Program credits, reminders, general help.</td>
</tr>
<tr>
<td><img src="image" alt="Alert Icon" /></td>
<td>Error conditions, conditions requiring immediate action.</td>
</tr>
<tr>
<td><img src="image" alt="Question Mark Icon" /></td>
<td>Inquiries, most confirmations.</td>
</tr>
<tr>
<td><img src="image" alt="Stop Icon" /></td>
<td>Potentially program-fatal errors, confirmation of an irreversible action.</td>
</tr>
<tr>
<td><img src="image" alt="Information Icon" /></td>
<td>Informational alerts. These usually have only an 'OK' button. Alerts with more than one choice might be better suited for the question mark icon.</td>
</tr>
<tr>
<td><img src="image" alt="Folder Icon" /></td>
<td>General disk errors and requests.</td>
</tr>
</tbody>
</table>

### The File Selector

Several important style guidelines are important to follow when using the system calls `fsel_input()` or `fsel_exinput()` to provide the common system file selector to the user. If your application provides a custom file selector unique to your application, always allow the user the choice of using the system file selector as opposed to your own. In general, it is better to use the internal selector rather than provide a customized one. The user may install a third-party file selector replacement if they want the extra features that custom file selectors usually provide. This provides more user-interface consistency throughout the system.

If you commonly use a third-party replacement file selector on the system you test applications on, always test your application with the replacement file selector disabled. Several third-party file selectors handle screen redraws and pathname parsing differently than the internal file selector does.
When your application needs to display the file selector, always ensure that the pathname that is going to be passed to the file selector call is valid. If the pathname becomes invalid, revert to a system default path such as that of your applications own. It is also courteous to the user to store the last used path in a global buffer so that each time the file selector is accessed the user doesn’t have to change directories again.

If your application requires that its files be loaded and saved with a specific file extension, append that file mask to the end of the pathname so that the user’s choices are restricted. If during a save operation the user chooses to override your default extension, either allow it or prompt the user as to their true intention.

When the file selector call returns, if the filename field is blank, treat it as a ‘Cancel’. If a filename was entered but it contains no file extension, append your default file extension (if appropriate) to it.

**Progress Indicators**

When an application begins a task that may require a substantial amount of time to complete, it is normally appropriate to change the mouse to a **BUSY_BEE** form to indicate to the user a long action is taking place.

If the screen display does not reflect the actual task in real time, it is helpful to display a progress bar (sometimes referred to as a thermometer) indicator on screen to remind the user that an task is indeed taking place and that the computer has not entered a locked state. In this case, you may leave the mouse form in the **ARROW** shape so that the user may perform other functions in a multitasking environment.

It is helpful to place a progress bar for potentially long operations into a window so that other applications or desk accessories may be accessed. When possible, the exact length of the operation might be stated like “Time Left: xx:xx”.

The progress bar should move as closely as possible to a true proportional representation of time (i.e. avoid circumstances where it might take ten seconds to move from 25% to 50% but only a second to move from 50% to 100%).

An example progress bar showing a task in progress is shown below:
Toolboxes

Toolboxes are groups of buttons (usually G_IMAGE or G_ICON) which either select between editing modes (often in graphic editors or DTP applications) or choose object properties. A toolbox may be contained in its own window or appear ‘attached’ in the document window aligned with the upper-left corner of the work area. A toolbox in its own window should have its ‘un-toppable’ characteristic set under MultiTOS (see wind_set()) to prevent the user from having to click twice to select a button.

Buttons on these specialized dialog/window combinations fall into three categories, exclusive buttons (such as a pointer tool and rectangle tool), non-exclusive buttons (such as zoom on/off), and style buttons (such as fill style and line style).

Buttons should reflect their state by appearing either inverted or depressed. The currently selected exclusive button as well as any selected non-exclusive button retains this state until a new object is chosen or it is deselected. Style buttons are only selected until the user has completed the operation. When available, toolbox buttons should appear in color using a G_CICON. An example of a toolbox window follows:
Toolbars (sometimes referred to as ‘Ribbons’) are single-strip toolboxes placed at the top of the document work area which contain buttons or combo boxes which are usually used to alter properties of the document. An example of a control bar embedded in a window follows:

Newer versions of the AES provide built-in support for toolbars, though they can be implemented in applications running in an OS that does not support the new calls.
Menus

The Menu Bar

Each application in the system should initialize a menu bar as soon as it is called. The menu bar consists of several titles which when pointed to by the mouse cause a list of individual menu items to be displayed.

The leftmost menu title (commonly referred to as the ‘Desk’ menu) should be the application name\(^1\). An example of the first menu title/items are shown below:

```
<table>
<thead>
<tr>
<th>PrgName</th>
<th>File</th>
</tr>
</thead>
<tbody>
<tr>
<td>About PrgName...</td>
<td></td>
</tr>
</tbody>
</table>
```

The first item in the menu should be “About \textit{PRGNAME}...”. \textit{PRGNAME} should be substituted with the name of the application. The lines below are reserved for desk accessories and applications (when running under \textit{MultiTOS}).

An application should call \texttt{menu_register()} (under \textit{MultiTOS}) to change its entry in the menu from the filename to the program title.

The second and third menu titles should be “File” and “Edit” as appropriate (though the inclusion of both of these menus is highly recommended). Application defined menus should be placed after these. If a “Help” menu is available it should be the rightmost title. A “Window” menu should be placed rightmost second only to “Help” if it exists. An example title bar follows:

```
| PrgName | File | Edit | Options | Window | Help |
```

Menu entries should be grouped by function under appropriate titles and subgrouped by placing separator bars between them (disabled dashes).

Menu entries which end in an ellipsis should lead to a dialog box. Those without ellipsis should carry out an action with no further user interaction.

\(^1\)This menu title used to be labeled “Desk” or contain the fuji logo. With the advent of \textit{MultiTOS}, however, placing the application name here makes it possible for the user to easily determine the application which has the input focus.
The File Menu

The “File” menu should consist of the following items (presented in order):

- New
- Open...
- Recall (optional – has cascading menu attached with most-recently used file list)
- Save
- Save as...
- Save all (optional)
- Any other document closing commands as required.

Separator

- Import (if applicable)
- Export (if applicable)
- Any other file operations as required2.

Separator

- Page Setup... (if applicable)
- Print (if applicable)
- Any other printing commands as required.

Separator

- Quit

Following is an example “File” menu:

<table>
<thead>
<tr>
<th>File</th>
</tr>
</thead>
<tbody>
<tr>
<td>New</td>
</tr>
<tr>
<td>Open</td>
</tr>
<tr>
<td>Close</td>
</tr>
<tr>
<td>Save</td>
</tr>
<tr>
<td>Save as...</td>
</tr>
<tr>
<td>Save all</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Import</th>
</tr>
</thead>
<tbody>
<tr>
<td>Export</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Page Setup...</th>
</tr>
</thead>
<tbody>
<tr>
<td>Print</td>
</tr>
</tbody>
</table>

| Quit | ^Q |

2 This does not refer to operations such as ‘Delete File’ or ‘Rename File’. These commands should not be supported in applications because they are available from the Desktop running under MultiTOS or from disk utility CPX’s and accessories.
The Edit Menu

The next menu, “Edit”, usually contains the following items:

- Undo (if supported)
- Redo (if supported\(^3\))
- Cut
- Copy
- Paste
- Delete
- Select All (optional)
- Find... (optional)
- Replace... (optional)
- Find Next (optional)
- Any other editing/searching commands.

An example “Edit” menu follows:

```
<table>
<thead>
<tr>
<th>Undo</th>
<th>Undo</th>
</tr>
</thead>
<tbody>
<tr>
<td>Cut</td>
<td>^X</td>
</tr>
<tr>
<td>Copy</td>
<td>^C</td>
</tr>
<tr>
<td>Paste</td>
<td>^Y</td>
</tr>
<tr>
<td>Delete</td>
<td>Del</td>
</tr>
<tr>
<td>Select All</td>
<td>^A</td>
</tr>
<tr>
<td>Find</td>
<td>^F</td>
</tr>
<tr>
<td>Replace</td>
<td>^G</td>
</tr>
<tr>
<td>Find Next</td>
<td>^H</td>
</tr>
</tbody>
</table>
```

Dual-State Menu Items

Menu selections can be designed to represent toggles. There are two methods of accomplishing this as follows:

- Apply a checkmark to the item to indicate an enabled state.
- Alter the text. For example, when “Hide Toolbar” is clicked, change the text to “Show Toolbar”.

\(^3\)‘Redo’ is used when multiple levels of ‘Undo’ are to be provided.
In addition, some menu item groups may provide a choice between more than two options as shown in the following example:

<table>
<thead>
<tr>
<th>Style</th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>Font...</td>
<td>F4</td>
</tr>
<tr>
<td>Normal</td>
<td></td>
</tr>
<tr>
<td>Bold</td>
<td>^B</td>
</tr>
<tr>
<td>Italic</td>
<td>^I</td>
</tr>
<tr>
<td>Underline</td>
<td>^U</td>
</tr>
<tr>
<td>Shadowed</td>
<td></td>
</tr>
</tbody>
</table>

Again, checkmarks can be used to indicate the selection.

Here are some other general pointers about using menus:

- Menu items such as “Preferences...” or “Save Preferences” belong in the “Options” menu.
- Menu items for text styles (like bold, italic) can be made G_USERDEF objects and made to reflect their actual state.
- If you add a “Window” menu, items such as “New Window” which opens a new window for the current document, “Arrange All”, “Tile All”, “Cascade All”, which positions windows can optionally be included. Followed by a separator, a generic item “Window” can be attached to a cascading menu which contains an updated list of all document windows so that the user can use the menu bar to ‘top’ a window.
- If you add a “Help” menu, different options can provide different levels of help such as “Contents” or “Index”. Don’t list help items for each possible dialog box or mode, instead provide context sensitive help that is activated through a “Help” button or by pressing the HELP key.

**Popup Menus**

Popup menus are menus which can appear anywhere on screen at the request of the user. A common use of popup menus is for object-specific options which are called upon when an object is right-clicked on with the mouse.

Popup menus can also be placed in dialog boxes as shown below. Dialog objects which lead to popup menus should be TOUCHEXIT and SHADOWED. If text describing the popup appears at the left of the button, it should be inverted when the popup is displayed and until it is closed.

When a popup menu contains a list of exclusive options, the option currently selected should be properly identified to the menu_popup() command so that it is aligned with the object in addition to having a checkmark. Popups with no selected option should always start at the first selection.
Popup menus may contain objects other than text (like fill styles or bitmaps) but will be unable to scroll.

**Drop-Down List Boxes**

Drop-down list boxes are handled in the same manner as popup menus with the following exceptions:

An ‘equivalence’ character (ASCII 240) in a **BOXCHAR** object should be displayed immediately to the right of the box leading to the drop-down list and should also be **TOUCHEXIT** and **SHADOWED**. A click on this object is the same as clicking on the main object.

No checkmark should be displayed next to the current selection.

The **TOUCHEXIT** box leading to a drop-down list may be editable, if appropriate, to allow the user to add items to those currently in the list.

The following illustrations show examples of both a ‘closed’ (prior to being selected) and ‘open’ (during selection) drop-down list:

- **Drop-Down List Box (closed)**

- **Drop-Down List Box (open)**
Hierarchical Menus

Hierarchical menus (or sub-menus) are menus attached to either a main menu item or a popup menu item. These menus can be nested several levels deep but it is recommended that this feature not be used because your menu bar, in general, should never be this complex. An example of a hierarchical menu follows:

Keyboard Equivalents

Some users prefer to do their program interaction via the mouse while others prefer the keyboard. Those users who prefer keyboard interaction are often frustrated by a lack of consistency among programs concerning keyboard equivalents.

The following keyboard equivalents are universal among many platforms (including Atari) and should be enabled in all cases where a counterpart option exists in an application. Other keyboard equivalents may be assigned as long as they do not conflict with one of those already predefined. The use of the ALTERNATE key as a modifier in a keyboard equivalent is discouraged because international users use the ALTERNATE key to access special keyboard characters.

Menus

Menu keyboard equivalents should be notated next the menu item and flush right (excepting one space) with the menu. The CONTROL key should be notated by the caret, the ALTERNATE key should be notated by the window closer character, and the SHIFT key should be notated by the up arrow character. Function keys are notated “Fnn” and other keys are notated as, for example, “Del”, “Bksp”, “Help”, etc.

Menu items with a sub-menu attachment should not have a keyboard equivalent. An example menu with keyboard equivalents shown correctly follows:
Following is a list of defined keyboard equivalents:

<table>
<thead>
<tr>
<th>Key Equivalent</th>
<th>Operation</th>
</tr>
</thead>
<tbody>
<tr>
<td>CTRL-N</td>
<td>New</td>
</tr>
<tr>
<td>CTRL-O</td>
<td>Open</td>
</tr>
<tr>
<td>CTRL-W</td>
<td>Close</td>
</tr>
<tr>
<td>CTRL-S</td>
<td>Save as...</td>
</tr>
<tr>
<td>CTRL-SHIFT-S</td>
<td>Save</td>
</tr>
<tr>
<td>CTRL-P</td>
<td>Print</td>
</tr>
<tr>
<td>CTRL-SHIFT-P</td>
<td>Page Setup</td>
</tr>
<tr>
<td>CTRL-Q</td>
<td>Quit</td>
</tr>
<tr>
<td>CTRL-X</td>
<td>Cut</td>
</tr>
<tr>
<td>CTRL-C</td>
<td>Copy</td>
</tr>
<tr>
<td>CTRL-V</td>
<td>Paste</td>
</tr>
<tr>
<td>CTRL-A</td>
<td>Select all</td>
</tr>
<tr>
<td>CTRL-F</td>
<td>Find</td>
</tr>
<tr>
<td>CTRL-R</td>
<td>Replace</td>
</tr>
<tr>
<td>HELP</td>
<td>Access help</td>
</tr>
<tr>
<td>SHIFT-HELP</td>
<td>Engage context sensitive help. Pointer should change to arrow/question mark and help should be provided for any object clicked on.</td>
</tr>
<tr>
<td>UNDO</td>
<td>Undo last operation</td>
</tr>
</tbody>
</table>

### Windows

When working with text-oriented applications, the following list of keyboard equivalents apply. Keep in mind that CTRL is generally a character-based modifier while SHIFT is line-based.

<table>
<thead>
<tr>
<th>Key Equivalent</th>
<th>Operation</th>
</tr>
</thead>
<tbody>
<tr>
<td>CTRL-B</td>
<td>Bold</td>
</tr>
<tr>
<td>CTRL-I</td>
<td>Italic</td>
</tr>
</tbody>
</table>
When working with object-oriented applications, the following keyboard equivalents are suggested:

<table>
<thead>
<tr>
<th>Key Equivalent</th>
<th>Operation</th>
</tr>
</thead>
<tbody>
<tr>
<td>ARROW</td>
<td>Deselect current object(s), select previous/next object.</td>
</tr>
<tr>
<td>BACKSPACE</td>
<td>Delete selected object.</td>
</tr>
<tr>
<td>DELETE</td>
<td>Delete selected object.</td>
</tr>
<tr>
<td>TAB</td>
<td>Deselect current object, select next object.</td>
</tr>
</tbody>
</table>

**Disjoint/Group Selection**

When in the context of a text-editing application, **SHIFT**-clicking on a point should select the text from the cursor position to the point clicked or add that region to a current selection (if one exists). In an object-oriented application, **SHIFT**-clicking should allow the user to select and deselect multiple objects.

**Device Independence**

Programming for compatibility on the Atari is a simple task. Here are some basic tips:

- A GEM program should use the **VDI** for all graphical/screen output. Never use **GEMDOS**, **BIOS**, or **XBIOS** functions to output to the screen or manipulate the palette.

- Don’t make assumptions about the type of display based on any call such as **Getrez()**, **EsetShift()**, or **Vsetmode()**. Only look at the values returned by the **VDI v_opnvwk()** call.

- For printing, always support **GDOS**. It is the only way to ensure that a user has a printer driver and fonts for the attached printer and that output is consistent among different printers. As with the screen, never make assumptions about the printer based on criteria like driver name, etc.
Never write directly to hardware unless it’s the documented way to accomplish a task. This is an almost sure sign that your program will break in future hardware releases.

Avoid using interrupt vectors. If you must use them, use `Setexc()`.

**Globalization**

One of the most effective ways a software marketer can increase his product’s sales is by ensuring its usability in foreign countries. Programmers can make their software more portable through the following methods:

- Store all language-dependent strings in the application’s resource file. Porting to other languages may then be accomplished by the modification of the resource file only.
- When creating resource files, allow at least 50% more space than that is required for English text. The English language tends to require fewer characters than most others.
- Use the `__IDT` and `__AKP` cookie to globalize references to dates, times, and currencies. If your application does not have a resource file, you may also use the `__AKP` cookie to select among language specific strings embedded within your code. When the `__AKP` cookie is not present you can check for language information embedded in the program header.

**Colors**

An application’s proper use of color can greatly enhance its effectiveness. Likewise, improper use of color can thoroughly confuse a user. Below are some basic rules about the use of color:

- Never alter the first 16 colors in modes with 256 colors or more. Only change system colors in other cases when absolutely necessary. These are system colors which should be controlled exclusively by the user.
- When providing a custom 3D effect to complement the OS under TOS 4.0 and above, use `objc_sysvar()` to interrogate color settings to allow your objects to match.
- Make dialogs `FL3DBAK` objects to allow the user’s selected dialog color to come through.
- Don’t use colors to decorate, use them to emphasize or draw attention to an important screen element. Use colors to display choices relating to color or when a user expects it in the document.
- When using color as a choice indicator, use green as a positive, red as a negative.
Sound

As with color, the proper use of sound can help or hinder an application program. The system bell should be used as a polite reminder to the user when an operation is being attempted that is beyond the capabilities of the application (ex: scrolling past the last line in a document). It is also useful to alert the user to the end of a long operation (during which the user might have stepped away).

In general, applications should restrict their use of sounds to the system bell. Beyond that, applications can support sounds through the use of the accessory “System Audio Manager” (supplied with the Falcon030) or have their custom sounds provided they may be enabled selectively by the user.

Application Software

Application software programmers writing for the Atari line of computers should follow the following suggestions:

- Provide an installation program on the distribution floppy called ‘INSTALL.PRG’. See below for details.
- Use the ‘_IDT’ cookie to determine the proper method of displaying dates and times. Use the ‘_AKP’ cookie to determine the country’s currency character.
- Provide help in as many places as possible. Provide context-sensitive help if possible.
- Your application file, its resource file(s), and any ‘readme’ files should be together in one directory. Any other application data files should be kept in a child directory of the application directory.

Installation Software

Every disk distributed for end-user use should have an installation program called ‘INSTALL.PRG’ on the root directory of the floppy or CD-ROM diskette. Even disks containing only data files should be installable in this manner. Basic guidelines for installation programs follow:

- The installation program should allow the user to specify a location for the files to be installed and create a new directory for them if necessary.
- The installation program may (if desired by the user) add icons for the application itself and data files to the DESKICON.RSC or DESKCICN.RSC file as appropriate. If the application
Entertainment software written for Atari computers should follow these minimum standards.

• Allow the user to install your software on the hard drive using an ‘INSTALL.PRG’.

• Don’t force the user to change resolutions prior to running your software.

• The path to your application should not contain data files, place those in a folder.

• Allow the user to return to the desktop in the same resolution he left.

• If possible, allow the game to be run in a window.

• Use device-independent graphics paired with the VDI call \texttt{vr\_trnfm()} to translate your graphics upon loading to be compatible with the installed video shifter.

• Support the enhanced analog joystick rather than CX-40 style controls on machines which have the ports to support them (like the STe and Falcon030). Use the CX-40 controls if four-player play is desired.
— APPENDIX A —

FUNCTIONS BY OPCODE
# GEMDOS Functions by Opcode

<table>
<thead>
<tr>
<th>Dec</th>
<th>Hex</th>
<th>Function</th>
<th>Summary</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0x00</td>
<td>PtermØ()</td>
<td>Exit process with a return code of 0.</td>
<td>2.122</td>
</tr>
<tr>
<td>1</td>
<td>0x01</td>
<td>Cconin()</td>
<td>Fetch a character from the console device and echo it.</td>
<td>2.41</td>
</tr>
<tr>
<td>2</td>
<td>0x02</td>
<td>Cconout()</td>
<td>Output a character to the console device processing any special keys.</td>
<td>2.43</td>
</tr>
<tr>
<td>3</td>
<td>0x03</td>
<td>Cauxin()</td>
<td>Fetch character from the auxiliary device.</td>
<td>2.39</td>
</tr>
<tr>
<td>4</td>
<td>0x04</td>
<td>Cauxout()</td>
<td>Output a character to the auxiliary device.</td>
<td>2.41</td>
</tr>
<tr>
<td>5</td>
<td>0x05</td>
<td>Cprnout()</td>
<td>Output a character to the printer device.</td>
<td>2.47</td>
</tr>
<tr>
<td>6</td>
<td>0x06</td>
<td>Crawio()</td>
<td>Perform input and output on the console device.</td>
<td>2.49</td>
</tr>
<tr>
<td>7</td>
<td>0x07</td>
<td>Crawcin()</td>
<td>Output a character to the console device.</td>
<td>2.48</td>
</tr>
<tr>
<td>8</td>
<td>0x08</td>
<td>Cnecin()</td>
<td>Fetch a character from the console device.</td>
<td>2.46</td>
</tr>
<tr>
<td>9</td>
<td>0x09</td>
<td>Cconws()</td>
<td>Write a string to the console device.</td>
<td>2.45</td>
</tr>
<tr>
<td>10</td>
<td>0x0A</td>
<td>Cconrs()</td>
<td>Read a string from the console device.</td>
<td>2.44</td>
</tr>
<tr>
<td>11</td>
<td>0x0B</td>
<td>Cconis()</td>
<td>Determine if a character is waiting to be received from the console device.</td>
<td>2.42</td>
</tr>
<tr>
<td>14</td>
<td>0x0E</td>
<td>Dsetdrv()</td>
<td>Set the default drive.</td>
<td>2.62</td>
</tr>
<tr>
<td>16</td>
<td>0x10</td>
<td>Cconos()</td>
<td>Determine if a character may be sent to the console device.</td>
<td>2.43</td>
</tr>
<tr>
<td>17</td>
<td>0x11</td>
<td>Cprnos()</td>
<td>Determine if a character may be sent to the printer device.</td>
<td>2.46</td>
</tr>
<tr>
<td>18</td>
<td>0x12</td>
<td>Cauxis()</td>
<td>Determine if a character is waiting to be received from the auxiliary device.</td>
<td>2.39</td>
</tr>
<tr>
<td>19</td>
<td>0x13</td>
<td>Cauxos()</td>
<td>Determine if a character may be sent to the auxiliary device.</td>
<td>2.40</td>
</tr>
<tr>
<td>20</td>
<td>0x14</td>
<td>Maddalt()</td>
<td>Notify GEMDOS of additional memory.</td>
<td>2.97</td>
</tr>
<tr>
<td>25</td>
<td>0x19</td>
<td>Dgetdta()</td>
<td>Return the default drive.</td>
<td>2.56</td>
</tr>
<tr>
<td>26</td>
<td>0x1A</td>
<td>Fsetdta()</td>
<td>Set the address of the DTA.</td>
<td>2.91</td>
</tr>
<tr>
<td>32</td>
<td>0x20</td>
<td>Super()</td>
<td>Modify user/supervisor status.</td>
<td>2.128</td>
</tr>
<tr>
<td>42</td>
<td>0x2A</td>
<td>Tgetdate()</td>
<td>Get the current date.</td>
<td>2.132</td>
</tr>
<tr>
<td>43</td>
<td>0x2B</td>
<td>Tsetdate()</td>
<td>Set the current date.</td>
<td>2.133</td>
</tr>
<tr>
<td>44</td>
<td>0x2C</td>
<td>Tgettime()</td>
<td>Get the current time.</td>
<td>2.132</td>
</tr>
<tr>
<td>45</td>
<td>0x2D</td>
<td>Tsettime()</td>
<td>Set the current time.</td>
<td>2.133</td>
</tr>
<tr>
<td>47</td>
<td>0x2F</td>
<td>Fgetdta()</td>
<td>Return a pointer to the DTA.</td>
<td>2.79</td>
</tr>
<tr>
<td>48</td>
<td>0x30</td>
<td>Sversion()</td>
<td>Obtain the current GEMDOS version.</td>
<td>2.129</td>
</tr>
<tr>
<td>49</td>
<td>0x31</td>
<td>Ptermres()</td>
<td>Exit process leaving some data intact.</td>
<td>2.123</td>
</tr>
<tr>
<td>54</td>
<td>0x36</td>
<td>Dfree()</td>
<td>Determine the free space on a drive.</td>
<td>2.54</td>
</tr>
<tr>
<td>57</td>
<td>0x39</td>
<td>Dcreate()</td>
<td>Create a directory.</td>
<td>2.53</td>
</tr>
<tr>
<td>58</td>
<td>0x3A</td>
<td>Ddelete()</td>
<td>Delete a directory.</td>
<td>2.54</td>
</tr>
<tr>
<td>59</td>
<td>0x3B</td>
<td>Dsetpath()</td>
<td>Set the default path.</td>
<td>2.63</td>
</tr>
<tr>
<td>60</td>
<td>0x3C</td>
<td>Fcreate()</td>
<td>Create a file.</td>
<td>2.74</td>
</tr>
<tr>
<td>61</td>
<td>0x3D</td>
<td>Fopen()</td>
<td>Open a file.</td>
<td>2.84</td>
</tr>
<tr>
<td>62</td>
<td>0x3E</td>
<td>Fclose()</td>
<td>Close a file.</td>
<td>2.66</td>
</tr>
<tr>
<td>63</td>
<td>0x3F</td>
<td>Fread()</td>
<td>Read binary data from a file.</td>
<td>2.87</td>
</tr>
<tr>
<td>64</td>
<td>0x40</td>
<td>Fwrite()</td>
<td>Write binary data to a file.</td>
<td>2.95</td>
</tr>
<tr>
<td>65</td>
<td>0x41</td>
<td>Fdelete()</td>
<td>Delete a file.</td>
<td>2.76</td>
</tr>
<tr>
<td>66</td>
<td>0x42</td>
<td>Fseek()</td>
<td>Move a file pointer.</td>
<td>2.89</td>
</tr>
<tr>
<td>67</td>
<td>0x43</td>
<td>Fattrib()</td>
<td>Get or set the attributes of a file.</td>
<td>2.64</td>
</tr>
<tr>
<td>68</td>
<td>0x44</td>
<td>Mxalloc()</td>
<td>Allocate memory with preference.</td>
<td>2.100</td>
</tr>
<tr>
<td>69</td>
<td>0x45</td>
<td>Fdup()</td>
<td>Duplicate a file handle.</td>
<td>2.76</td>
</tr>
<tr>
<td>Dec</td>
<td>Hex</td>
<td>Function</td>
<td>Summary</td>
<td>Page</td>
</tr>
<tr>
<td>-----</td>
<td>------</td>
<td>--------------</td>
<td>---------------------------------------------------</td>
<td>------</td>
</tr>
<tr>
<td>70</td>
<td>0x46</td>
<td>Fforce()</td>
<td>Redirect one handle to another.</td>
<td>2.77</td>
</tr>
<tr>
<td>71</td>
<td>0x47</td>
<td>Dgetpath()</td>
<td>Return the default path.</td>
<td>2.57</td>
</tr>
<tr>
<td>72</td>
<td>0x48</td>
<td>Malloc()</td>
<td>Allocate memory.</td>
<td>2.98</td>
</tr>
<tr>
<td>73</td>
<td>0x49</td>
<td>Mfree()</td>
<td>Free allocated memory.</td>
<td>2.99</td>
</tr>
<tr>
<td>74</td>
<td>0x4A</td>
<td>Mshrink()</td>
<td>Shrink or expand a block of memory.</td>
<td>2.99</td>
</tr>
<tr>
<td>75</td>
<td>0x4B</td>
<td>Pexec()</td>
<td>Execute another process.</td>
<td>2.103</td>
</tr>
<tr>
<td>76</td>
<td>0x4C</td>
<td>Pterm()</td>
<td>Exit process with the specified return code.</td>
<td>2.121</td>
</tr>
<tr>
<td>78</td>
<td>0x4E</td>
<td>Fsfirst()</td>
<td>Find a file with the specified mask.</td>
<td>2.92</td>
</tr>
<tr>
<td>79</td>
<td>0x4F</td>
<td>Fsnext()</td>
<td>Find subsequent files with the specified mask.</td>
<td>2.93</td>
</tr>
<tr>
<td>86</td>
<td>0x56</td>
<td>Frename()</td>
<td>Rename a file or directory.</td>
<td>2.89</td>
</tr>
<tr>
<td>87</td>
<td>0x57</td>
<td>Fdatetime()</td>
<td>Get or set the time/date flags of a file.</td>
<td>2.75</td>
</tr>
<tr>
<td>92</td>
<td>0x5C</td>
<td>Flock()</td>
<td>Set or remove a file lock.</td>
<td>2.82</td>
</tr>
<tr>
<td>255</td>
<td>0xFF</td>
<td>Syield()</td>
<td>Surrender the remaining portion of the processes</td>
<td>2.130</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>timeslice.</td>
<td></td>
</tr>
<tr>
<td>256</td>
<td>0x100</td>
<td>Fpipe()</td>
<td>Establish a communication pipeline between</td>
<td>2.86</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>processes.</td>
<td></td>
</tr>
<tr>
<td>260</td>
<td>0x104</td>
<td>Fcntl()</td>
<td>Perform a file-system specific file operation.</td>
<td>2.67</td>
</tr>
<tr>
<td>261</td>
<td>0x105</td>
<td>Finstat()</td>
<td>Determine the input status of a file.</td>
<td>2.80</td>
</tr>
<tr>
<td>262</td>
<td>0x106</td>
<td>Foutstat()</td>
<td>Determine the output status of a file.</td>
<td>2.85</td>
</tr>
<tr>
<td>263</td>
<td>0x107</td>
<td>Fgetchar()</td>
<td>Get a character from a file.</td>
<td>2.79</td>
</tr>
<tr>
<td>264</td>
<td>0x108</td>
<td>Fputchar()</td>
<td>Output a character to a file.</td>
<td>2.86</td>
</tr>
<tr>
<td>265</td>
<td>0x109</td>
<td>Pwait()</td>
<td>Determine the exit code of a stopped or terminated</td>
<td>2.125</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>child process.</td>
<td></td>
</tr>
<tr>
<td>266</td>
<td>0x10A</td>
<td>Pnice()</td>
<td>Alter the process priority of the calling</td>
<td>2.111</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>process.</td>
<td></td>
</tr>
<tr>
<td>267</td>
<td>0x10B</td>
<td>Pgtpid()</td>
<td>Obtain the process ID of the calling process.</td>
<td>2.107</td>
</tr>
<tr>
<td>268</td>
<td>0x10C</td>
<td>Pgtppid()</td>
<td>Obtain the process ID of the processes’ parent.</td>
<td>2.108</td>
</tr>
<tr>
<td>269</td>
<td>0x10D</td>
<td>Pgtpgrp()</td>
<td>Obtain the process group ID of the calling</td>
<td>2.107</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>process.</td>
<td></td>
</tr>
<tr>
<td>270</td>
<td>0x10E</td>
<td>Psetpgp()</td>
<td>Set the process group ID for the calling process.</td>
<td>2.115</td>
</tr>
<tr>
<td>271</td>
<td>0x10F</td>
<td>Pgetuid()</td>
<td>Obtain the user ID of the calling process.</td>
<td>2.108</td>
</tr>
<tr>
<td>272</td>
<td>0x110</td>
<td>Psetuid()</td>
<td>Set the user ID for the calling process.</td>
<td>2.116</td>
</tr>
<tr>
<td>273</td>
<td>0x111</td>
<td>Pkill()</td>
<td>Send a signal to one or more processes.</td>
<td>2.109</td>
</tr>
<tr>
<td>274</td>
<td>0x112</td>
<td>Psig()</td>
<td>Determine the action to take when a signal is</td>
<td>2.118</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>received.</td>
<td></td>
</tr>
<tr>
<td>275</td>
<td>0x113</td>
<td>Pvfork()</td>
<td>Create a duplicate of the current process which</td>
<td>2.124</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>shares address and data space with its parent.</td>
<td></td>
</tr>
<tr>
<td>276</td>
<td>0x114</td>
<td>Pgetgid()</td>
<td>Obtain the group ID of the calling process.</td>
<td>2.107</td>
</tr>
<tr>
<td>277</td>
<td>0x115</td>
<td>Psetgid()</td>
<td>Set the group ID of the calling process.</td>
<td>2.114</td>
</tr>
<tr>
<td>278</td>
<td>0x116</td>
<td>Psigblock()</td>
<td>Block selected signals from delivery.</td>
<td>2.118</td>
</tr>
<tr>
<td>279</td>
<td>0x117</td>
<td>Psigsetmask()</td>
<td>Specifies which signals should be blocked and which</td>
<td>2.121</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>should be received.</td>
<td></td>
</tr>
<tr>
<td>280</td>
<td>0x118</td>
<td>Pusrval()</td>
<td>Get or set the user-defined value associated with</td>
<td>2.124</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>a process.</td>
<td></td>
</tr>
<tr>
<td>281</td>
<td>0x119</td>
<td>Pdomain()</td>
<td>Get or set the processes execution domain.</td>
<td>2.102</td>
</tr>
<tr>
<td>282</td>
<td>0x11A</td>
<td>Psigreturn()</td>
<td>Clean up from a signal handler.</td>
<td>2.120</td>
</tr>
<tr>
<td>283</td>
<td>0x11B</td>
<td>Pfork()</td>
<td>Create a copy of the current process.</td>
<td>2.105</td>
</tr>
<tr>
<td>284</td>
<td>0x11C</td>
<td>Pwait3()</td>
<td>Determine the exit code of stopped or terminated</td>
<td>2.126</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>child processes.</td>
<td></td>
</tr>
<tr>
<td>285</td>
<td>0x11D</td>
<td>Fselect()</td>
<td>Enumerate file descriptors which are ready for</td>
<td>2.90</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>reading/writing.</td>
<td></td>
</tr>
<tr>
<td>286</td>
<td>0x11E</td>
<td>Prusage()</td>
<td>Return resource usage information on the calling</td>
<td>2.112</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>process.</td>
<td></td>
</tr>
<tr>
<td>287</td>
<td>0x11F</td>
<td>Psetlimit()</td>
<td>Read or modify resource usage limits for a process.</td>
<td>2.114</td>
</tr>
<tr>
<td>288</td>
<td>0x120</td>
<td>Talarm()</td>
<td>Read or set an alarm for the current process.</td>
<td>2.131</td>
</tr>
<tr>
<td>Dec</td>
<td>Hex</td>
<td>Function</td>
<td>Summary</td>
<td>Page</td>
</tr>
<tr>
<td>-----</td>
<td>-----</td>
<td>----------------</td>
<td>------------------------------------------------------------------------------------------------------------------------------------------</td>
<td>------</td>
</tr>
<tr>
<td>289</td>
<td>0x121</td>
<td>Pause()</td>
<td>Suspend the process until a signal is received.</td>
<td>2.101</td>
</tr>
<tr>
<td>290</td>
<td>0x122</td>
<td>Sysconf()</td>
<td>Return information regarding current capabilities and limitations of processes running under MiNT.</td>
<td>2.130</td>
</tr>
<tr>
<td>291</td>
<td>0x123</td>
<td>Psigpending()</td>
<td>Determines which signals have been sent but not yet received to the calling process.</td>
<td>2.120</td>
</tr>
<tr>
<td>292</td>
<td>0x124</td>
<td>Dpathconf()</td>
<td>Return information regarding limitations and capabilities of a file system.</td>
<td>2.59</td>
</tr>
<tr>
<td>293</td>
<td>0x125</td>
<td>Pmsg()</td>
<td>Send or receive a message.</td>
<td>2.109</td>
</tr>
<tr>
<td>294</td>
<td>0x126</td>
<td>Fmidipipe()</td>
<td>Change the file handles which refer to MIDI input and output.</td>
<td>2.83</td>
</tr>
<tr>
<td>295</td>
<td>0x127</td>
<td>Prenice()</td>
<td>Alter the process priority of the specified process.</td>
<td>2.111</td>
</tr>
<tr>
<td>296</td>
<td>0x128</td>
<td>Dopendir()</td>
<td>Open a directory.</td>
<td>2.58</td>
</tr>
<tr>
<td>297</td>
<td>0x129</td>
<td>Dreaddir()</td>
<td>Read a directory entry.</td>
<td>2.61</td>
</tr>
<tr>
<td>298</td>
<td>0x12A</td>
<td>Drewinddir()</td>
<td>Reset the directory pointer.</td>
<td>2.62</td>
</tr>
<tr>
<td>299</td>
<td>0x12B</td>
<td>Dclosedir()</td>
<td>Close a directory.</td>
<td>2.50</td>
</tr>
<tr>
<td>300</td>
<td>0x12C</td>
<td>Fxattr()</td>
<td>Return extended attribute information for a file.</td>
<td>2.95</td>
</tr>
<tr>
<td>301</td>
<td>0x12D</td>
<td>Flink()</td>
<td>Create a file link.</td>
<td>2.81</td>
</tr>
<tr>
<td>302</td>
<td>0x12E</td>
<td>Fsymlink()</td>
<td>Establish a symbolic link to a file.</td>
<td>2.94</td>
</tr>
<tr>
<td>303</td>
<td>0x12F</td>
<td>Freadlink()</td>
<td>Determine the actual file to which a link refers.</td>
<td>2.88</td>
</tr>
<tr>
<td>304</td>
<td>0x130</td>
<td>Dcntl()</td>
<td>Perform a file-system specific device operation.</td>
<td>2.50</td>
</tr>
<tr>
<td>305</td>
<td>0x131</td>
<td>Fchown()</td>
<td>Modify the ownership of a file.</td>
<td>2.66</td>
</tr>
<tr>
<td>306</td>
<td>0x132</td>
<td>Fchmod()</td>
<td>Modify the access permission flags of a file.</td>
<td>2.65</td>
</tr>
<tr>
<td>307</td>
<td>0x133</td>
<td>Pumask()</td>
<td>Determines the minimum file and/or directory creation access permission masks.</td>
<td>2.123</td>
</tr>
<tr>
<td>308</td>
<td>0x134</td>
<td>Psemaphore()</td>
<td>Create a semaphore.</td>
<td>2.113</td>
</tr>
<tr>
<td>309</td>
<td>0x135</td>
<td>Dlock()</td>
<td>Lock or unlock a BIOS disk device.</td>
<td>2.57</td>
</tr>
<tr>
<td>310</td>
<td>0x136</td>
<td>Psigpause()</td>
<td>Suspends the process until a specified signal (or signals) is received.</td>
<td>2.119</td>
</tr>
<tr>
<td>311</td>
<td>0x137</td>
<td>Psigaction()</td>
<td>Changes the way a signal is handled.</td>
<td>2.116</td>
</tr>
<tr>
<td>312</td>
<td>0x138</td>
<td>Pgeteuid()</td>
<td>Returns the effective user ID of the caller.</td>
<td>2.106</td>
</tr>
<tr>
<td>313</td>
<td>0x139</td>
<td>Pgetegid()</td>
<td>Returns the effective group ID of the caller.</td>
<td>2.106</td>
</tr>
<tr>
<td>314</td>
<td>0x13A</td>
<td>Pwaitpid()</td>
<td>Attempts to determine the exit code of a particular process.</td>
<td>2.127</td>
</tr>
<tr>
<td>315</td>
<td>0x13B</td>
<td>Dgetcwd()</td>
<td>Returns the current GEMDOS working directory for the process on the specified drive.</td>
<td>2.56</td>
</tr>
<tr>
<td>316</td>
<td>0x13C</td>
<td>Salert()</td>
<td>Sends an alert to the alert pipe ‘U:\PIPE\ALERT’.</td>
<td>2.128</td>
</tr>
</tbody>
</table>
## BIOS Functions by Opcode

<table>
<thead>
<tr>
<th>Dec</th>
<th>Hex</th>
<th>Function</th>
<th>Summary</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0x00</td>
<td>Getmpb()</td>
<td>Return the address of the MPB (Memory Parameter Block) structure.</td>
<td>3.31</td>
</tr>
<tr>
<td>1</td>
<td>0x01</td>
<td>Bconstat()</td>
<td>Determine if a character is waiting from a device.</td>
<td>3.28</td>
</tr>
<tr>
<td>2</td>
<td>0x02</td>
<td>Bconin()</td>
<td>Input a character from a device.</td>
<td>3.27</td>
</tr>
<tr>
<td>3</td>
<td>0x03</td>
<td>Bconout()</td>
<td>Output a character from a device.</td>
<td>3.28</td>
</tr>
<tr>
<td>4</td>
<td>0x04</td>
<td>Rwabs()</td>
<td>Read/write sectors to a device.</td>
<td>3.34</td>
</tr>
<tr>
<td>5</td>
<td>0x05</td>
<td>Setexc()</td>
<td>Set or read a system exception vector.</td>
<td>3.35</td>
</tr>
<tr>
<td>6</td>
<td>0x06</td>
<td>Tickcal()</td>
<td>Return the current system timer calibration.</td>
<td>3.36</td>
</tr>
<tr>
<td>7</td>
<td>0x07</td>
<td>Getbpbi()</td>
<td>Return the address of the BPB (BIOS Parameter Block).</td>
<td>3.30</td>
</tr>
<tr>
<td>8</td>
<td>0x08</td>
<td>Bcostat()</td>
<td>Determine if a device is ready to receive a character.</td>
<td>3.29</td>
</tr>
<tr>
<td>9</td>
<td>0x09</td>
<td>Mediach()</td>
<td>Determine if a drive’s media has been changed.</td>
<td>3.33</td>
</tr>
<tr>
<td>10</td>
<td>0x0A</td>
<td>Drvmap()</td>
<td>Return a bitmap of mounted drives.</td>
<td>3.30</td>
</tr>
<tr>
<td>11</td>
<td>0x0B</td>
<td>Kbshift()</td>
<td>Return the state of the keyboard shift keys.</td>
<td>3.32</td>
</tr>
<tr>
<td>Dec</td>
<td>Hex</td>
<td>Function</td>
<td>Summary</td>
<td>Page</td>
</tr>
<tr>
<td>-----</td>
<td>------</td>
<td>---------------------</td>
<td>---------------------------------------------------</td>
<td>------</td>
</tr>
<tr>
<td>0</td>
<td>0x00</td>
<td>Initmouse()</td>
<td>Initialize the mouse handler.</td>
<td>4.73</td>
</tr>
<tr>
<td>1</td>
<td>0x01</td>
<td>Ssbrk()</td>
<td>Reserve memory at the top of RAM.</td>
<td>4.102</td>
</tr>
<tr>
<td>2</td>
<td>0x02</td>
<td>Physbase()</td>
<td>Return the address of the physical screen.</td>
<td>4.85</td>
</tr>
<tr>
<td>3</td>
<td>0x03</td>
<td>Logbase()</td>
<td>Return the address of the logical screen.</td>
<td>4.80</td>
</tr>
<tr>
<td>4</td>
<td>0x04</td>
<td>Getrez()</td>
<td>Return the current screen resolution code.</td>
<td>4.68</td>
</tr>
<tr>
<td>5</td>
<td>0x05</td>
<td>Setscreen() and VsetScreen()</td>
<td>Set the current screen address and mode.</td>
<td>4.97</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td>4.108</td>
</tr>
<tr>
<td>6</td>
<td>0x06</td>
<td>Setpalette()</td>
<td>Set entries in the ST compatible palette.</td>
<td>4.95</td>
</tr>
<tr>
<td>7</td>
<td>0x07</td>
<td>Setcolor()</td>
<td>Set an entry in the ST compatible palette.</td>
<td>4.93</td>
</tr>
<tr>
<td>8</td>
<td>0x08</td>
<td>Floprd()</td>
<td>Read a sector from a floppy disk.</td>
<td>4.66</td>
</tr>
<tr>
<td>9</td>
<td>0x09</td>
<td>Flopwr()</td>
<td>Write a sector to a floppy disk.</td>
<td>4.67</td>
</tr>
<tr>
<td>10</td>
<td>0x0A</td>
<td>Flopfmt()</td>
<td>Format a sector on a floppy disk.</td>
<td>4.63</td>
</tr>
<tr>
<td>11</td>
<td>0x0B</td>
<td>Dbsmsg()</td>
<td>Send a debugging message to the resident debugger.</td>
<td>4.28</td>
</tr>
<tr>
<td>12</td>
<td>0x0C</td>
<td>Midiw()</td>
<td>Write a string to the MIDI port.</td>
<td>4.82</td>
</tr>
<tr>
<td>13</td>
<td>0x0D</td>
<td>Mpint()</td>
<td>Define an MFP interrupt.</td>
<td>4.81</td>
</tr>
<tr>
<td>14</td>
<td>0x0E</td>
<td>Iorec()</td>
<td>Return the address of the system IOREC structure.</td>
<td>4.75</td>
</tr>
<tr>
<td>15</td>
<td>0x0F</td>
<td>Rsconf()</td>
<td>Configure the currently mapped RS-232 port.</td>
<td>4.89</td>
</tr>
<tr>
<td>16</td>
<td>0x10</td>
<td>Keytbl()</td>
<td>Return the addresses of the current key mapping tables.</td>
<td>4.78</td>
</tr>
<tr>
<td>17</td>
<td>0x11</td>
<td>Random()</td>
<td>Return a random number.</td>
<td>4.89</td>
</tr>
<tr>
<td>18</td>
<td>0x12</td>
<td>Protobt()</td>
<td>Prototype a floppy boot sector.</td>
<td>4.86</td>
</tr>
<tr>
<td>19</td>
<td>0x13</td>
<td>Flopver()</td>
<td>Verify a sector on a floppy disk.</td>
<td>4.66</td>
</tr>
<tr>
<td>20</td>
<td>0x14</td>
<td>Scrdump()</td>
<td>Execute the built-in screen dump code.</td>
<td>4.91</td>
</tr>
<tr>
<td>21</td>
<td>0x15</td>
<td>Cursconf()</td>
<td>Configure the TOS cursor.</td>
<td>4.27</td>
</tr>
<tr>
<td>22</td>
<td>0x16</td>
<td>Settime()</td>
<td>Set the time of day and current date.</td>
<td>4.98</td>
</tr>
<tr>
<td>23</td>
<td>0x17</td>
<td>Gettime()</td>
<td>Get the time of day and current date.</td>
<td>4.69</td>
</tr>
<tr>
<td>24</td>
<td>0x18</td>
<td>Bioskeys()</td>
<td>Reset the keyboard mapping tables to default.</td>
<td>4.24</td>
</tr>
<tr>
<td>25</td>
<td>0x19</td>
<td>Ikbwds()</td>
<td>Write a string to the intelligent keyboard controller.</td>
<td>4.72</td>
</tr>
<tr>
<td>26</td>
<td>0x1A</td>
<td>Jdisint()</td>
<td>Disable an MFP interrupt.</td>
<td>4.76</td>
</tr>
<tr>
<td>27</td>
<td>0x1B</td>
<td>Jenabint()</td>
<td>Enable an MFP interrupt.</td>
<td>4.76</td>
</tr>
<tr>
<td>28</td>
<td>0x1C</td>
<td>Giaccess()</td>
<td>Modify or set a register on the PSG.</td>
<td>4.70</td>
</tr>
<tr>
<td>29</td>
<td>0x1D</td>
<td>Offgibit()</td>
<td>Toggle bits of the PSG Port A off.</td>
<td>4.84</td>
</tr>
<tr>
<td>30</td>
<td>0x1E</td>
<td>On gibit()</td>
<td>Toggle bits of the PSG Port A on.</td>
<td>4.84</td>
</tr>
<tr>
<td>31</td>
<td>0x1F</td>
<td>Xtimer()</td>
<td>Set an interrupt on the 68901.</td>
<td>4.113</td>
</tr>
<tr>
<td>32</td>
<td>0x20</td>
<td>Dosound()</td>
<td>Start an interrupt driven sound routine.</td>
<td>4.33</td>
</tr>
<tr>
<td>33</td>
<td>0x21</td>
<td>Setprt()</td>
<td>Set or read the printer configuration bits.</td>
<td>4.96</td>
</tr>
<tr>
<td>34</td>
<td>0x22</td>
<td>Kbdvibse()</td>
<td>Return the address of the current IKBD interrupt table.</td>
<td>4.77</td>
</tr>
<tr>
<td>35</td>
<td>0x23</td>
<td>Kbrate()</td>
<td>Set or read the keyboard repeat rate.</td>
<td>4.78</td>
</tr>
<tr>
<td>36</td>
<td>0x24</td>
<td>Prtblk()</td>
<td>Print a block of memory using the built-in screen dump routines.</td>
<td>4.87</td>
</tr>
<tr>
<td>37</td>
<td>0x25</td>
<td>Vsync()</td>
<td>Hold the process until the next vertical blank.</td>
<td>4.110</td>
</tr>
<tr>
<td>38</td>
<td>0x26</td>
<td>Supexec()</td>
<td>Execute a routine in supervisor mode.</td>
<td>4.103</td>
</tr>
<tr>
<td>39</td>
<td>0x27</td>
<td>Puntaes()</td>
<td>Discard the AES.</td>
<td>4.88</td>
</tr>
<tr>
<td>Dec</td>
<td>Hex</td>
<td>Function</td>
<td>Summary</td>
<td>Page</td>
</tr>
<tr>
<td>-----</td>
<td>------</td>
<td>----------------</td>
<td>--------------------------------------------------------</td>
<td>------</td>
</tr>
<tr>
<td>41</td>
<td>0x29</td>
<td>Floprate()</td>
<td>Set the floppy drive seek rates.</td>
<td>4.65</td>
</tr>
<tr>
<td>42</td>
<td>0x2A</td>
<td>DMAread()</td>
<td>Read sectors from a DMA/SCSI device.</td>
<td>4.31</td>
</tr>
<tr>
<td>43</td>
<td>0x2B</td>
<td>DMAwrite()</td>
<td>Write sectors to a DMA/SCSI device.</td>
<td>4.32</td>
</tr>
<tr>
<td>44</td>
<td>0x2C</td>
<td>Bconmap()</td>
<td>Modify the BIOS device mapping table.</td>
<td>4.23</td>
</tr>
<tr>
<td>46</td>
<td>0x2E</td>
<td>NVaccess()</td>
<td>Access non-volatile RAM.</td>
<td>4.83</td>
</tr>
<tr>
<td>48</td>
<td>0x30</td>
<td>Metainit()</td>
<td>Initialize MetaDOS.</td>
<td>4.80</td>
</tr>
<tr>
<td>64</td>
<td>0x40</td>
<td>Blitmode()</td>
<td>Get or set the state of the BLITTER chip.</td>
<td>4.25</td>
</tr>
<tr>
<td>80</td>
<td>0x50</td>
<td>EsetShift()</td>
<td>Set the TT030 shift mode registers.</td>
<td>4.61</td>
</tr>
<tr>
<td>81</td>
<td>0x51</td>
<td>EgetShift()</td>
<td>Get the TT030 shift mode registers.</td>
<td>4.57</td>
</tr>
<tr>
<td>82</td>
<td>0x52</td>
<td>EsetBank()</td>
<td>Set the current TT030 color bank.</td>
<td>4.58</td>
</tr>
<tr>
<td>83</td>
<td>0x53</td>
<td>EsetColor()</td>
<td>Get or set a color in the TT030 palette.</td>
<td>4.59</td>
</tr>
<tr>
<td>84</td>
<td>0x54</td>
<td>EsetPalette()</td>
<td>Set the TT030 palette.</td>
<td>4.60</td>
</tr>
<tr>
<td>85</td>
<td>0x55</td>
<td>EgetPalette()</td>
<td>Get the TT030 palette.</td>
<td>4.56</td>
</tr>
<tr>
<td>86</td>
<td>0x56</td>
<td>EsetGray()</td>
<td>Set the TT030 gray mode register.</td>
<td>4.60</td>
</tr>
<tr>
<td>87</td>
<td>0x57</td>
<td>EsetSmeare()</td>
<td>Set the TT030 smear mode register.</td>
<td>4.62</td>
</tr>
<tr>
<td>88</td>
<td>0x58</td>
<td>VsetMode()</td>
<td>Set the Falcon030 video mode.</td>
<td>4.107</td>
</tr>
<tr>
<td>89</td>
<td>0x59</td>
<td>VgetMonitor()</td>
<td>Identify the kind of monitor attached to the Falcon030.</td>
<td>4.104</td>
</tr>
<tr>
<td>90</td>
<td>0x5A</td>
<td>VsetSync()</td>
<td>Set the Falcon030 sync mode.</td>
<td>4.109</td>
</tr>
<tr>
<td>91</td>
<td>0x5B</td>
<td>VgetSize()</td>
<td>Get the size of screen memory in bytes.</td>
<td>4.105</td>
</tr>
<tr>
<td>92</td>
<td>0x5C</td>
<td>VsetMask()</td>
<td>Set the mask assigned to each true color plotted.</td>
<td>4.106</td>
</tr>
<tr>
<td>93</td>
<td>0x5D</td>
<td>VsetRGB()</td>
<td>Set the Falcon030 palette using RGB data.</td>
<td>4.108</td>
</tr>
<tr>
<td>94</td>
<td>0x5E</td>
<td>VgetRGB()</td>
<td>Get the Falcon030 palette using RGB data.</td>
<td>4.104</td>
</tr>
<tr>
<td>96</td>
<td>0x60</td>
<td>Dsp_DoBlock()</td>
<td>Transfer byte wise packed data to/from the DSP.</td>
<td>4.38</td>
</tr>
<tr>
<td>97</td>
<td>0x61</td>
<td>Dsp_BlkHandshake()</td>
<td>Handshakes byte wise packed data to/from the DSP.</td>
<td>4.35</td>
</tr>
<tr>
<td>98</td>
<td>0x62</td>
<td>Dsp_BlkUnpacked()</td>
<td>Transfers data stored in a longword array to/from the DSP.</td>
<td>4.36</td>
</tr>
<tr>
<td>99</td>
<td>0x63</td>
<td>Dsp_InStream()</td>
<td>Transfers data to the DSP via an interrupt handler.</td>
<td>4.45</td>
</tr>
<tr>
<td>100</td>
<td>0x64</td>
<td>Dsp_OutStream()</td>
<td>Transfers data from the DSP via an interrupt handler.</td>
<td>4.51</td>
</tr>
<tr>
<td>101</td>
<td>0x65</td>
<td>Dsp_IOStream()</td>
<td>Transfers data to/from the DSP via concurrent interrupt handlers.</td>
<td>4.46</td>
</tr>
<tr>
<td>102</td>
<td>0x66</td>
<td>Dsp_RemoveInterrupts()</td>
<td>Disable the generation of DSP interrupts.</td>
<td>4.51</td>
</tr>
<tr>
<td>103</td>
<td>0x67</td>
<td>Dsp_GetWordSize()</td>
<td>Get the current size of a DSP word.</td>
<td>4.41</td>
</tr>
<tr>
<td>104</td>
<td>0x68</td>
<td>Dsp_Lock()</td>
<td>Lock the DSP system.</td>
<td>4.48</td>
</tr>
<tr>
<td>105</td>
<td>0x69</td>
<td>Dsp_Unlock()</td>
<td>Unlock the DSP system.</td>
<td>4.55</td>
</tr>
<tr>
<td>106</td>
<td>0x6A</td>
<td>Dsp_Available()</td>
<td>Determines the amount of free X and Y memory available in the DSP.</td>
<td>4.34</td>
</tr>
<tr>
<td>107</td>
<td>0x6B</td>
<td>Dsp_Reserve()</td>
<td>Reserves a portion of DSP memory for a user program.</td>
<td>4.53</td>
</tr>
<tr>
<td>108</td>
<td>0x6C</td>
<td>Dsp_LoadProg()</td>
<td>Loads a `.LOD' file from disk, transmits it to the DSP, and executes it.</td>
<td>4.47</td>
</tr>
<tr>
<td>109</td>
<td>0x6D</td>
<td>Dsp_ExecProg()</td>
<td>Transfers a DSP program in memory to the DSP and executes it.</td>
<td>4.39</td>
</tr>
<tr>
<td>110</td>
<td>0x6E</td>
<td>Dsp_ExecBoot()</td>
<td>Resets the DSP and loads a new bootstrap program into the first 512 words of DSP memory.</td>
<td>4.39</td>
</tr>
<tr>
<td>Dec</td>
<td>Hex</td>
<td>Function</td>
<td>Summary</td>
<td>Page</td>
</tr>
<tr>
<td>-----</td>
<td>------</td>
<td>---------------------------</td>
<td></td>
<td></td>
</tr>
<tr>
<td>111</td>
<td>0x6F</td>
<td><strong>Dsp_LodToBinary()</strong></td>
<td>Converts a '.LOD' file to binary format.</td>
<td>4.49</td>
</tr>
<tr>
<td>112</td>
<td>0x70</td>
<td><strong>Dsp_TriggerHC()</strong></td>
<td>Causes a host command set aside for DSP programs to execute.</td>
<td>4.55</td>
</tr>
<tr>
<td>113</td>
<td>0x71</td>
<td><strong>Dsp_RequestUniqueAbility()</strong></td>
<td>Requests a unique DSP ability identifier.</td>
<td>4.52</td>
</tr>
<tr>
<td>114</td>
<td>0x72</td>
<td><strong>Dsp_GetProgAbility()</strong></td>
<td>Returns the ability code for the program residing in DSP memory.</td>
<td>4.40</td>
</tr>
<tr>
<td>115</td>
<td>0x73</td>
<td><strong>Dsp_FlushSubroutines()</strong></td>
<td>Removes all DSP subroutines from memory.</td>
<td>4.40</td>
</tr>
<tr>
<td>116</td>
<td>0x74</td>
<td><strong>Dsp_LoadSubroutine()</strong></td>
<td>Loads a DSP subroutine into memory.</td>
<td>4.48</td>
</tr>
<tr>
<td>117</td>
<td>0x75</td>
<td><strong>Dsp_InqSubrAbility()</strong></td>
<td>Determines if a subroutine with the specified ability code is currently loaded into the DSP.</td>
<td>4.44</td>
</tr>
<tr>
<td>118</td>
<td>0x76</td>
<td><strong>Dsp_RunSubroutine()</strong></td>
<td>Begins execution of the specified subroutine.</td>
<td>4.53</td>
</tr>
<tr>
<td>119</td>
<td>0x77</td>
<td><strong>Dsp_Hf0()</strong></td>
<td>Reads/writes bit #3 of the HSR.</td>
<td>4.41</td>
</tr>
<tr>
<td>120</td>
<td>0x78</td>
<td><strong>Dsp_Hf1()</strong></td>
<td>Reads/writes bit #4 of the HSR.</td>
<td>4.42</td>
</tr>
<tr>
<td>121</td>
<td>0x79</td>
<td><strong>Dsp_Hf2()</strong></td>
<td>Reads bit #5 of the HSR.</td>
<td>4.43</td>
</tr>
<tr>
<td>122</td>
<td>0x7A</td>
<td><strong>Dsp_Hf3()</strong></td>
<td>Reads bit #6 of the HSR.</td>
<td>4.43</td>
</tr>
<tr>
<td>123</td>
<td>0x7B</td>
<td><strong>Dsp_BlkWords()</strong></td>
<td>Transfers an array of WORDs to/from the DSP.</td>
<td>4.37</td>
</tr>
<tr>
<td>124</td>
<td>0x7C</td>
<td><strong>Dsp_BlkBytes()</strong></td>
<td>Transfers an array of bytes to/from the DSP.</td>
<td>4.34</td>
</tr>
<tr>
<td>125</td>
<td>0x7D</td>
<td><strong>Dsp_Hstat()</strong></td>
<td>Returns the value of the DSP’s ICR register.</td>
<td>4.44</td>
</tr>
<tr>
<td>126</td>
<td>0x7E</td>
<td><strong>Dsp_SetVectors()</strong></td>
<td>Defines interrupt handlers to be called when DSP data is ready to be sent or received.</td>
<td>4.54</td>
</tr>
<tr>
<td>127</td>
<td>0x7F</td>
<td><strong>Dsp_RunBlocks()</strong></td>
<td>Transmits multiple blocks to/from the DSP.</td>
<td>4.50</td>
</tr>
<tr>
<td>128</td>
<td>0x80</td>
<td><strong>Locksnd()</strong></td>
<td>Lock the sound system.</td>
<td>4.79</td>
</tr>
<tr>
<td>129</td>
<td>0x81</td>
<td><strong>Unlocksnd()</strong></td>
<td>Unlock the sound system.</td>
<td>4.103</td>
</tr>
<tr>
<td>130</td>
<td>0x82</td>
<td><strong>Soundcmd()</strong></td>
<td>Execute a sound system specific function.</td>
<td>4.100</td>
</tr>
<tr>
<td>131</td>
<td>0x83</td>
<td><strong>Setbuffer()</strong></td>
<td>Set the record and playback buffers.</td>
<td>4.92</td>
</tr>
<tr>
<td>132</td>
<td>0x84</td>
<td><strong>Setmode()</strong></td>
<td>Set the playback/record mode.</td>
<td>4.94</td>
</tr>
<tr>
<td>133</td>
<td>0x85</td>
<td><strong>Settracks()</strong></td>
<td>Set the playback/record tracks.</td>
<td>4.99</td>
</tr>
<tr>
<td>134</td>
<td>0x86</td>
<td><strong>Setmontracks()</strong></td>
<td>Set the track to be output over the speaker/headphone.</td>
<td>4.95</td>
</tr>
<tr>
<td>135</td>
<td>0x87</td>
<td><strong>Setinterrupt()</strong></td>
<td>Set the sound system interrupts.</td>
<td>4.93</td>
</tr>
<tr>
<td>136</td>
<td>0x88</td>
<td><strong>Buffop()</strong></td>
<td>Enable or disable playback/recording.</td>
<td>4.25</td>
</tr>
<tr>
<td>137</td>
<td>0x89</td>
<td><strong>Dsptristate()</strong></td>
<td>Connect or disconnect the DSP from the connection matrix.</td>
<td>4.56</td>
</tr>
<tr>
<td>138</td>
<td>0x8A</td>
<td><strong>Gpio()</strong></td>
<td>Read or write data over the general purpose pins on the DSP port.</td>
<td>4.72</td>
</tr>
<tr>
<td>139</td>
<td>0x8B</td>
<td><strong>Devconnect()</strong></td>
<td>Connect devices in the connection matrix.</td>
<td>4.29</td>
</tr>
<tr>
<td>140</td>
<td>0x8C</td>
<td><strong>Sndstatus()</strong></td>
<td>Obtain the status of the sound system.</td>
<td>4.99</td>
</tr>
<tr>
<td>141</td>
<td>0x8D</td>
<td><strong>Buffptr()</strong></td>
<td>Return the current position of the record or playback buffer pointers.</td>
<td>4.26</td>
</tr>
<tr>
<td>165</td>
<td>0xA5</td>
<td><strong>WavePlay()</strong></td>
<td>Playback a DMA sample.</td>
<td>4.110</td>
</tr>
</tbody>
</table>
# AES Functions by Opcode

<table>
<thead>
<tr>
<th>Dec</th>
<th>Hex</th>
<th>Function</th>
<th>Summary</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>10</td>
<td>0x0A</td>
<td>appl_init()</td>
<td>Initializes a GEM application.</td>
<td>6.53</td>
</tr>
<tr>
<td>11</td>
<td>0x0B</td>
<td>appl_read()</td>
<td>Reads data from the message pipe.</td>
<td>6.54</td>
</tr>
<tr>
<td>12</td>
<td>0x0C</td>
<td>appl_write()</td>
<td>Writes data to the message pipe.</td>
<td>6.58</td>
</tr>
<tr>
<td>13</td>
<td>0x0D</td>
<td>appl_find()</td>
<td>Locates a system process.</td>
<td>6.47</td>
</tr>
<tr>
<td>14</td>
<td>0x0E</td>
<td>appl_tplay()</td>
<td>Plays back recorded events.</td>
<td>6.56</td>
</tr>
<tr>
<td>15</td>
<td>0x0F</td>
<td>appl_trecord()</td>
<td>Records keyboard and mouse events.</td>
<td>6.57</td>
</tr>
<tr>
<td>18</td>
<td>0x12</td>
<td>appl_search()</td>
<td>Enumerates system processes.</td>
<td>6.55</td>
</tr>
<tr>
<td>19</td>
<td>0x13</td>
<td>appl_exit()</td>
<td>Prepares a GEM application for termination.</td>
<td>6.47</td>
</tr>
<tr>
<td>20</td>
<td>0x14</td>
<td>evnt_keybd()</td>
<td>Waits for a keyboard event.</td>
<td>6.63</td>
</tr>
<tr>
<td>21</td>
<td>0x15</td>
<td>evnt_button()</td>
<td>Waits for a mouse button event.</td>
<td>6.61</td>
</tr>
<tr>
<td>22</td>
<td>0x16</td>
<td>evnt_mouse()</td>
<td>Waits for a mouse rectangle event.</td>
<td>6.70</td>
</tr>
<tr>
<td>23</td>
<td>0x17</td>
<td>evnt_mesag()</td>
<td>Waits for an application message.</td>
<td>6.64</td>
</tr>
<tr>
<td>24</td>
<td>0x18</td>
<td>evnt_timer()</td>
<td>Waits for a timer event.</td>
<td>6.73</td>
</tr>
<tr>
<td>25</td>
<td>0x19</td>
<td>evnt_multi()</td>
<td>Waits for multiple events.</td>
<td>6.71</td>
</tr>
<tr>
<td>26</td>
<td>0x1A</td>
<td>evnt_dclick()</td>
<td>Sets the mouse double-click rate.</td>
<td>6.62</td>
</tr>
<tr>
<td>30</td>
<td>0x1E</td>
<td>menu_bar()</td>
<td>Displays/removes a menu bar.</td>
<td>6.105</td>
</tr>
<tr>
<td>31</td>
<td>0x1F</td>
<td>menu_icheck()</td>
<td>Places a checkmark beside a menu item.</td>
<td>6.106</td>
</tr>
<tr>
<td>32</td>
<td>0x20</td>
<td>menu_ienable()</td>
<td>Enables/disables a menu item.</td>
<td>6.106</td>
</tr>
<tr>
<td>33</td>
<td>0x21</td>
<td>menu_tnormal()</td>
<td>Selects/deselects a menu item or title.</td>
<td>6.111</td>
</tr>
<tr>
<td>34</td>
<td>0x22</td>
<td>menu_text()</td>
<td>Changes menu item/title text.</td>
<td>6.111</td>
</tr>
<tr>
<td>35</td>
<td>0x23</td>
<td>menu_register()</td>
<td>Registers applications in the menu bar.</td>
<td>6.109</td>
</tr>
<tr>
<td>36</td>
<td>0x24</td>
<td>menu_popup()</td>
<td>Manages a floating popup menu.</td>
<td>6.108</td>
</tr>
<tr>
<td>37</td>
<td>0x25</td>
<td>menu_attach()</td>
<td>Attaches a sub-menu to a menu item.</td>
<td>6.103</td>
</tr>
<tr>
<td>38</td>
<td>0x26</td>
<td>menu_istart()</td>
<td>Defines the initial selection of a sub-menu.</td>
<td>6.107</td>
</tr>
<tr>
<td>39</td>
<td>0x27</td>
<td>menu_settings()</td>
<td>Modifies popup menu settings.</td>
<td>6.110</td>
</tr>
<tr>
<td>40</td>
<td>0x28</td>
<td>objc_add()</td>
<td>Adds an object to an object tree.</td>
<td>6.115</td>
</tr>
<tr>
<td>41</td>
<td>0x29</td>
<td>objc_delete()</td>
<td>Deletes an object from an object tree.</td>
<td>6.116</td>
</tr>
<tr>
<td>42</td>
<td>0x2A</td>
<td>objc_draw()</td>
<td>Draws an object tree.</td>
<td>6.117</td>
</tr>
<tr>
<td>43</td>
<td>0x2B</td>
<td>objc_find()</td>
<td>Locates an object based on screen coordinates.</td>
<td>6.119</td>
</tr>
<tr>
<td>44</td>
<td>0x2C</td>
<td>objc_offset()</td>
<td>Determines the offset of child objects in an object tree.</td>
<td>6.120</td>
</tr>
<tr>
<td>45</td>
<td>0x2D</td>
<td>objc_order()</td>
<td>Reorders objects within an object tree.</td>
<td>6.121</td>
</tr>
<tr>
<td>46</td>
<td>0x2E</td>
<td>objc_edit()</td>
<td>Manipulates an editable object.</td>
<td>6.118</td>
</tr>
<tr>
<td>47</td>
<td>0x2F</td>
<td>objc_change()</td>
<td>Changes the state of an object.</td>
<td>6.115</td>
</tr>
<tr>
<td>48</td>
<td>0x30</td>
<td>objc_sysvar()</td>
<td>Reads/modifies the system defaults for 3D effects.</td>
<td>6.121</td>
</tr>
<tr>
<td>50</td>
<td>0x32</td>
<td>form_do()</td>
<td>Manages a user-defined form.</td>
<td>6.81</td>
</tr>
<tr>
<td>51</td>
<td>0x33</td>
<td>form_dial()</td>
<td>Reserves/releases screen space for forms.</td>
<td>6.80</td>
</tr>
<tr>
<td>52</td>
<td>0x34</td>
<td>form_alert()</td>
<td>Manages a generic alert.</td>
<td>6.77</td>
</tr>
<tr>
<td>53</td>
<td>0x35</td>
<td>form_error()</td>
<td>Manages a generic error alert.</td>
<td>6.82</td>
</tr>
<tr>
<td>54</td>
<td>0x36</td>
<td>form_center()</td>
<td>Centers an object tree on screen.</td>
<td>6.79</td>
</tr>
<tr>
<td>55</td>
<td>0x37</td>
<td>form_keybd()</td>
<td>Provides a system-level editable field handler.</td>
<td>6.83</td>
</tr>
<tr>
<td>56</td>
<td>0x38</td>
<td>form_button()</td>
<td>Provides a system-level button handler.</td>
<td>6.78</td>
</tr>
<tr>
<td>70</td>
<td>0x46</td>
<td>graf_rubberbox()</td>
<td>Controls the shrinking/enlarging of a box outline.</td>
<td>6.97</td>
</tr>
<tr>
<td>71</td>
<td>0x47</td>
<td>graf_dragbox()</td>
<td>Controls the moving of a box outline.</td>
<td>6.91</td>
</tr>
<tr>
<td>72</td>
<td>0x48</td>
<td>graf_movebox()</td>
<td>Draws a moving box.</td>
<td>6.96</td>
</tr>
<tr>
<td>73</td>
<td>0x49</td>
<td>graf_growbox()</td>
<td>Draws an expanding box.</td>
<td>6.92</td>
</tr>
<tr>
<td>Dec</td>
<td>Hex</td>
<td>Function</td>
<td>Summary</td>
<td>Page</td>
</tr>
<tr>
<td>-----</td>
<td>-------</td>
<td>---------------------</td>
<td>----------------------------------------</td>
<td>------</td>
</tr>
<tr>
<td>74</td>
<td>0x50</td>
<td>graf_shrinkbox()</td>
<td>Draws a shrinking box.</td>
<td></td>
</tr>
<tr>
<td>75</td>
<td>0x51</td>
<td>graf_watchbox()</td>
<td>Selects/draws an object depending on the position of the mouse.</td>
<td></td>
</tr>
<tr>
<td>76</td>
<td>0x52</td>
<td>graf_slidebox()</td>
<td>Controls a slider outline.</td>
<td></td>
</tr>
<tr>
<td>77</td>
<td>0x53</td>
<td>graf_handle()</td>
<td>Obtains AES workstation attributes.</td>
<td></td>
</tr>
<tr>
<td>78</td>
<td>0x54</td>
<td>graf_mouse()</td>
<td>Defines the mouse form.</td>
<td></td>
</tr>
<tr>
<td>79</td>
<td>0x55</td>
<td>graf_mkstate()</td>
<td>Provides information about the mouse state.</td>
<td></td>
</tr>
<tr>
<td>80</td>
<td>0x56</td>
<td>scrp_read()</td>
<td>Determines the system scrap directory.</td>
<td></td>
</tr>
<tr>
<td>81</td>
<td>0x57</td>
<td>scrp_write()</td>
<td>Sets the system scrap directory.</td>
<td></td>
</tr>
<tr>
<td>90</td>
<td>0x58</td>
<td>fsel_input()</td>
<td>Manages the file selector.</td>
<td></td>
</tr>
<tr>
<td>91</td>
<td>0x59</td>
<td>fsel_exinput()</td>
<td>Manages the extended file selector.</td>
<td></td>
</tr>
<tr>
<td>100</td>
<td>0x64</td>
<td>wind_create()</td>
<td>Creates a window.</td>
<td></td>
</tr>
<tr>
<td>101</td>
<td>0x65</td>
<td>wind_open()</td>
<td>Opens a window.</td>
<td></td>
</tr>
<tr>
<td>102</td>
<td>0x66</td>
<td>wind_close()</td>
<td>Closes a window.</td>
<td></td>
</tr>
<tr>
<td>103</td>
<td>0x67</td>
<td>wind_delete()</td>
<td>Deletes a window.</td>
<td></td>
</tr>
<tr>
<td>104</td>
<td>0x68</td>
<td>wind_get()</td>
<td>Returns window attributes.</td>
<td></td>
</tr>
<tr>
<td>105</td>
<td>0x69</td>
<td>wind_set()</td>
<td>Sets a window attribute.</td>
<td></td>
</tr>
<tr>
<td>106</td>
<td>0x6A</td>
<td>wind_find()</td>
<td>Determines the window at given pixel coordinates.</td>
<td></td>
</tr>
<tr>
<td>107</td>
<td>0x6B</td>
<td>wind_update()</td>
<td>Manages the window update semaphore.</td>
<td></td>
</tr>
<tr>
<td>108</td>
<td>0x6C</td>
<td>wind_calc()</td>
<td>Calculates window extents.</td>
<td></td>
</tr>
<tr>
<td>109</td>
<td>0x6D</td>
<td>wind_new()</td>
<td>Removes all windows.</td>
<td></td>
</tr>
<tr>
<td>110</td>
<td>0x6E</td>
<td>rsrc_load()</td>
<td>Loads a disk-based resource file.</td>
<td></td>
</tr>
<tr>
<td>111</td>
<td>0x6F</td>
<td>rsrc_free()</td>
<td>Releases a resource file from memory.</td>
<td></td>
</tr>
<tr>
<td>112</td>
<td>0x70</td>
<td>rsrc_gaddr()</td>
<td>Calculates the address of a resource element.</td>
<td></td>
</tr>
<tr>
<td>113</td>
<td>0x71</td>
<td>rsrc_saddr()</td>
<td>Sets the address of a resource element.</td>
<td></td>
</tr>
<tr>
<td>114</td>
<td>0x72</td>
<td>rsrc_obfix()</td>
<td>Changes the coordinates of an object from character-based to pixel-based.</td>
<td></td>
</tr>
<tr>
<td>115</td>
<td>0x73</td>
<td>rsrc_rcfix()</td>
<td>Changes the coordinates of a resource file from character-based to pixel-based.</td>
<td></td>
</tr>
<tr>
<td>120</td>
<td>0x78</td>
<td>shel_read()</td>
<td>Determine’s the processes parent and command tail.</td>
<td></td>
</tr>
<tr>
<td>121</td>
<td>0x79</td>
<td>shel_write()</td>
<td>Manages process loading and control.</td>
<td></td>
</tr>
<tr>
<td>122</td>
<td>0x7A</td>
<td>shel_get()</td>
<td>Copies data from the system’s shell buffer.</td>
<td></td>
</tr>
<tr>
<td>123</td>
<td>0x7B</td>
<td>shel_put()</td>
<td>Stores data in the system’s shell buffer.</td>
<td></td>
</tr>
<tr>
<td>124</td>
<td>0x7C</td>
<td>shel_find()</td>
<td>Searches the AES’s path for a file.</td>
<td></td>
</tr>
<tr>
<td>125</td>
<td>0x7D</td>
<td>shel_envrm()</td>
<td>Searches the system environment string.</td>
<td></td>
</tr>
<tr>
<td>130</td>
<td>0x82</td>
<td>appl_getinfo()</td>
<td>Returns information about the AES.</td>
<td></td>
</tr>
</tbody>
</table>

---

THE ATARI COMPENDIUM

6.48

6.88

6.89

6.90

6.91

6.92

6.93

6.94

6.95

6.96

6.97

6.98

6.99

6.100

6.101

6.102

6.103

6.104

6.105

6.106

6.107

6.108

6.109

6.110

6.111

6.112

6.113

6.114

6.115

6.116

6.117

6.118

6.119

6.120

6.121

6.122

6.123

6.124

6.125

6.126

6.127

6.128

6.129

6.130

6.131

6.132

6.133

6.134

6.135

6.136

6.137

6.138

6.139

6.140

6.141

6.142

6.143

6.144

6.145

6.146

6.147

6.148

6.149

6.150

6.151

6.152

6.153

6.154

6.155

6.156

6.157

6.158

6.159

6.160
# VDI Functions by Opcode

<table>
<thead>
<tr>
<th>Opcode, Subopcode(s) (if required)</th>
<th>Function</th>
<th>Summary</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>N/A</td>
<td><code>vq_gdos()</code></td>
<td>Test for presence of <strong>GDOS</strong>.</td>
<td>7.92</td>
</tr>
<tr>
<td>-1, 6</td>
<td><code>v_set_app_buff()</code></td>
<td>Reserve bezier workspace.</td>
<td>7.77</td>
</tr>
<tr>
<td>1</td>
<td><code>v_opnwk()</code></td>
<td>Open physical workstation.</td>
<td>7.66</td>
</tr>
<tr>
<td>2</td>
<td><code>v_clswk()</code></td>
<td>Close a physical workstation.</td>
<td>7.35</td>
</tr>
<tr>
<td>3</td>
<td><code>v_clrwk()</code></td>
<td>Close a physical workstation.</td>
<td>7.34</td>
</tr>
<tr>
<td>4</td>
<td><code>v_updwk()</code></td>
<td>Update workstation.</td>
<td>7.78</td>
</tr>
<tr>
<td>5, 1</td>
<td><code>vq_chcells()</code></td>
<td>Return alpha screen size.</td>
<td>7.87</td>
</tr>
<tr>
<td>5, 2</td>
<td><code>v_exit_cur()</code></td>
<td>Exit text mode.</td>
<td>7.46</td>
</tr>
<tr>
<td>5, 3</td>
<td><code>v_enter_cur()</code></td>
<td>Enter text mode.</td>
<td>7.45</td>
</tr>
<tr>
<td>5, 4</td>
<td><code>v_curup()</code></td>
<td>Move text cursor up one row.</td>
<td>7.40</td>
</tr>
<tr>
<td>5, 5</td>
<td><code>v_curdown()</code></td>
<td>Move text cursor down one row.</td>
<td>7.37</td>
</tr>
<tr>
<td>5, 6</td>
<td><code>v_curright()</code></td>
<td>Move text cursor right one row.</td>
<td>7.38</td>
</tr>
<tr>
<td>5, 7</td>
<td><code>v_curleft()</code></td>
<td>Move text cursor up one row.</td>
<td>7.38</td>
</tr>
<tr>
<td>5, 8</td>
<td><code>v_curhome()</code></td>
<td>Home text cursor.</td>
<td>7.37</td>
</tr>
<tr>
<td>5, 9</td>
<td><code>v_eeos()</code></td>
<td>Erase to end of screen.</td>
<td>7.42</td>
</tr>
<tr>
<td>5, 10</td>
<td><code>v_eeol()</code></td>
<td>Erase to end of line.</td>
<td>7.41</td>
</tr>
<tr>
<td>5, 11</td>
<td><code>vs_curaddress()</code></td>
<td>Position text cursor.</td>
<td>7.126</td>
</tr>
<tr>
<td>5, 12</td>
<td><code>v_curtext()</code></td>
<td>Output text (alpha mode).</td>
<td>7.39</td>
</tr>
<tr>
<td>5, 13</td>
<td><code>v_rvon()</code></td>
<td>Reverse text on (alpha mode).</td>
<td>7.75</td>
</tr>
<tr>
<td>5, 14</td>
<td><code>v_rvoff()</code></td>
<td>Reverse text off (alpha mode).</td>
<td>7.75</td>
</tr>
<tr>
<td>5, 15</td>
<td><code>vq_curaddress()</code></td>
<td>Inquire text cursor location.</td>
<td>7.89</td>
</tr>
<tr>
<td>5, 16</td>
<td><code>vq_tabstatus()</code></td>
<td>Get availability of tablet.</td>
<td>7.95</td>
</tr>
<tr>
<td>5, 17</td>
<td><code>v_hardcopy()</code></td>
<td>Output screen to printer.</td>
<td>7.57</td>
</tr>
<tr>
<td>5, 18</td>
<td><code>v_dspcur()</code></td>
<td>Display text cursor.</td>
<td>7.40</td>
</tr>
<tr>
<td>5, 19</td>
<td><code>v_rmcur()</code></td>
<td>Remove text cursor.</td>
<td>7.74</td>
</tr>
<tr>
<td>5, 20</td>
<td><code>v_form_adv()</code></td>
<td>Advance printer page.</td>
<td>7.48</td>
</tr>
<tr>
<td>5, 21</td>
<td><code>v_output_window()</code></td>
<td>Output window of page to printer.</td>
<td>7.68</td>
</tr>
<tr>
<td>5, 22</td>
<td><code>v_clear Disp_list()</code></td>
<td>Clear display list.</td>
<td>7.34</td>
</tr>
<tr>
<td>5, 23</td>
<td><code>v_bit_image()</code></td>
<td>Render bit-image file.</td>
<td>7.31</td>
</tr>
<tr>
<td>5, 24</td>
<td><code>vq_scan()</code></td>
<td>Return printer scan heights.</td>
<td>7.94</td>
</tr>
<tr>
<td>5, 25</td>
<td><code>v_alpha_text()</code></td>
<td>Output printer text (alpha mode).</td>
<td>7.23</td>
</tr>
<tr>
<td>5, 60</td>
<td><code>vs_palette()</code></td>
<td>Set color palette.</td>
<td>7.127</td>
</tr>
<tr>
<td>5, 81</td>
<td><code>vt_resolution()</code></td>
<td>Set tablet resolution.</td>
<td>7.165</td>
</tr>
<tr>
<td>5, 82</td>
<td><code>vt_axis()</code></td>
<td>Set tablet axis resolution.</td>
<td>7.164</td>
</tr>
<tr>
<td>5, 83</td>
<td><code>vt_origin()</code></td>
<td>Set tablet origin.</td>
<td>7.164</td>
</tr>
<tr>
<td>5, 84</td>
<td><code>vq_tdimensions()</code></td>
<td>Return tablet X and Y dimensions.</td>
<td>7.96</td>
</tr>
<tr>
<td>5, 85</td>
<td><code>vt_alignment()</code></td>
<td>Set tablet alignment.</td>
<td>7.163</td>
</tr>
<tr>
<td>5, 91</td>
<td><code>vqp_films()</code></td>
<td>Return camera film types.</td>
<td>7.101</td>
</tr>
<tr>
<td>5, 92</td>
<td><code>vqp_state()</code></td>
<td>Return camera driver state.</td>
<td>7.101</td>
</tr>
<tr>
<td>5, 93</td>
<td><code>vsp_state()</code></td>
<td>Set camera driver state.</td>
<td>7.145</td>
</tr>
<tr>
<td>5, 94</td>
<td><code>vsp_save()</code></td>
<td>Save camera driver state.</td>
<td>7.145</td>
</tr>
<tr>
<td>5, 95</td>
<td><code>vsp_message()</code></td>
<td>Supress camera screen messages.</td>
<td>7.144</td>
</tr>
<tr>
<td>5, 96</td>
<td><code>vqp_error()</code></td>
<td>Return camera error status.</td>
<td>7.100</td>
</tr>
<tr>
<td>5, 98</td>
<td><code>v_meta_extents()</code></td>
<td>Specify metafile bounding box.</td>
<td>7.60</td>
</tr>
<tr>
<td>Opcode, Subopcode(s) (if required)</td>
<td>Function</td>
<td>Summary</td>
<td>Page</td>
</tr>
<tr>
<td>-----------------------------------</td>
<td>------------------------</td>
<td>--------------------------------------</td>
<td>-------</td>
</tr>
<tr>
<td>5, 99†</td>
<td>v_write_meta()</td>
<td>Write metafile item.</td>
<td>7.79</td>
</tr>
<tr>
<td>5, 99, 0†</td>
<td>vm_pagesize()</td>
<td>Set metafile page size.</td>
<td>7.85</td>
</tr>
<tr>
<td>5, 99, 1†</td>
<td>vm_coords()</td>
<td>Set metafile coordinate system.</td>
<td>7.83</td>
</tr>
<tr>
<td>5, 99, 32, 1†</td>
<td>v_bez_qual()</td>
<td>Set bezier quality.</td>
<td>7.30</td>
</tr>
<tr>
<td>5, 100</td>
<td>vm_filename()</td>
<td>Set metafile filename.</td>
<td>7.84</td>
</tr>
<tr>
<td>5, 102</td>
<td>v_fontinit()</td>
<td>Select a new system font.</td>
<td>7.48</td>
</tr>
<tr>
<td>5, 2000</td>
<td>v_pgcount()</td>
<td>Specify laser printer copies.</td>
<td>7.69</td>
</tr>
<tr>
<td>6</td>
<td>vpline()</td>
<td>Draw a polyline.</td>
<td>7.71</td>
</tr>
<tr>
<td>6, 13</td>
<td>v_bez()</td>
<td>Draw a bezier curve.</td>
<td>7.26</td>
</tr>
<tr>
<td>7</td>
<td>v_pmarker()</td>
<td>Draw polymarkers.</td>
<td>7.72</td>
</tr>
<tr>
<td>8</td>
<td>v_text()</td>
<td>Output graphic text.</td>
<td>7.56</td>
</tr>
<tr>
<td>9</td>
<td>v_fillarea()</td>
<td>Draw a filled polygon.</td>
<td>7.46</td>
</tr>
<tr>
<td>9, 13</td>
<td>v_bez_fill()</td>
<td>Draw a filled bezier curve.</td>
<td>7.27</td>
</tr>
<tr>
<td>10</td>
<td>v_cellarray()</td>
<td>Draw a cell array.</td>
<td>7.32</td>
</tr>
<tr>
<td>11, 1</td>
<td>v_bar()</td>
<td>Draw a rectangle.</td>
<td>7.25</td>
</tr>
<tr>
<td>11, 2</td>
<td>v_arc()</td>
<td>Draw an arc.</td>
<td>7.24</td>
</tr>
<tr>
<td>11, 3</td>
<td>v_pieslice()</td>
<td>Draw a pieslice.</td>
<td>7.70</td>
</tr>
<tr>
<td>11, 4</td>
<td>v_circle()</td>
<td>Draw a circle.</td>
<td>7.33</td>
</tr>
<tr>
<td>11, 5</td>
<td>v_ellipse()</td>
<td>Draw an ellipse.</td>
<td>7.43</td>
</tr>
<tr>
<td>11, 6</td>
<td>v_ellarc()</td>
<td>Draw an elliptical arc.</td>
<td>7.42</td>
</tr>
<tr>
<td>11, 7</td>
<td>v_ellpie()</td>
<td>Draw an elliptical pie segment.</td>
<td>7.44</td>
</tr>
<tr>
<td>11, 8</td>
<td>v_rbox()</td>
<td>Draw a rounded-rectangle.</td>
<td>7.72</td>
</tr>
<tr>
<td>11, 9</td>
<td>v_rfbox()</td>
<td>Draw a filled rounded-rectangle.</td>
<td>7.73</td>
</tr>
<tr>
<td>11, 10</td>
<td>v_justified()</td>
<td>Output justified text.</td>
<td>7.58</td>
</tr>
<tr>
<td>11, 13†</td>
<td>v_bez_off()</td>
<td>Disable bezier drawing.</td>
<td>7.28</td>
</tr>
<tr>
<td>11, 13†</td>
<td>v_bez_on()</td>
<td>Enable bezier drawing.</td>
<td>7.29</td>
</tr>
<tr>
<td>12</td>
<td>vst_height()</td>
<td>Set graphic text height (in pixels).</td>
<td>7.153</td>
</tr>
<tr>
<td>13</td>
<td>vst_rotation()</td>
<td>Set graphic text rotation.</td>
<td>7.156</td>
</tr>
<tr>
<td>14</td>
<td>vs_color()</td>
<td>Set color palette index.</td>
<td>7.126</td>
</tr>
<tr>
<td>15</td>
<td>vsl_type()</td>
<td>Set line type.</td>
<td>7.135</td>
</tr>
<tr>
<td>16</td>
<td>vsl_width()</td>
<td>Set line width.</td>
<td>7.137</td>
</tr>
<tr>
<td>17</td>
<td>vsl_color()</td>
<td>Set line color.</td>
<td>7.134</td>
</tr>
<tr>
<td>18</td>
<td>vsm_type()</td>
<td>Set marker type.</td>
<td>7.142</td>
</tr>
<tr>
<td>19</td>
<td>vsm_height()</td>
<td>Set marker height.</td>
<td>7.139</td>
</tr>
<tr>
<td>20</td>
<td>vsm_color()</td>
<td>Set marker color.</td>
<td>7.138</td>
</tr>
<tr>
<td>21</td>
<td>vst_font()</td>
<td>Set graphic text font.</td>
<td>7.152</td>
</tr>
<tr>
<td>22</td>
<td>vst_color()</td>
<td>Set graphic text color.</td>
<td>7.150</td>
</tr>
<tr>
<td>23</td>
<td>vsf_interior()</td>
<td>Set fill interior type.</td>
<td>7.129</td>
</tr>
<tr>
<td>24</td>
<td>vsf_style()</td>
<td>Set fill style type.</td>
<td>7.131</td>
</tr>
<tr>
<td>25</td>
<td>vsf_color()</td>
<td>Set fill color.</td>
<td>7.129</td>
</tr>
<tr>
<td>26</td>
<td>vq_color()</td>
<td>Inquire palette index.</td>
<td>7.88</td>
</tr>
<tr>
<td>27</td>
<td>vq_cellarray()</td>
<td>Inquire cell array.</td>
<td>7.86</td>
</tr>
<tr>
<td>28†</td>
<td>vrq_locator()</td>
<td>Poll for mouse/keyboard input.</td>
<td>7.121</td>
</tr>
<tr>
<td>28†</td>
<td>vsm_locator()</td>
<td>Sample mouse/keyboard input.</td>
<td>7.121</td>
</tr>
<tr>
<td>29†</td>
<td>vrq_valuator()</td>
<td>Poll for ‘valuator’ input.</td>
<td>7.123</td>
</tr>
<tr>
<td>29†</td>
<td>vsm_valuator()</td>
<td>Sample ‘valuator’ input.</td>
<td>7.143</td>
</tr>
<tr>
<td>30†</td>
<td>vrq_choice()</td>
<td>Poll for ‘choice’ input.</td>
<td>7.121</td>
</tr>
<tr>
<td>30†</td>
<td>vsm_choice()</td>
<td>Sample input from ‘choice’ device.</td>
<td>7.138</td>
</tr>
<tr>
<td>Opcode, Subopcode(s) (if required)</td>
<td>Function</td>
<td>Summary</td>
<td>Page</td>
</tr>
<tr>
<td>------------------------------------</td>
<td>---------------------</td>
<td>---------------------------------------------------</td>
<td>-------</td>
</tr>
<tr>
<td>31†</td>
<td>vrq_string()</td>
<td>Poll for keyboard string input.</td>
<td>7.122</td>
</tr>
<tr>
<td>31†</td>
<td>vsm_string()</td>
<td>Sample keyboard string input.</td>
<td>7.141</td>
</tr>
<tr>
<td>32</td>
<td>vswr_mode()</td>
<td>Set writing mode.</td>
<td>7.162</td>
</tr>
<tr>
<td>33</td>
<td>vsin_mode()</td>
<td>Set input mode.</td>
<td>7.133</td>
</tr>
<tr>
<td>35</td>
<td>vql_attributes()</td>
<td>Return line attributes.</td>
<td>7.98</td>
</tr>
<tr>
<td>36</td>
<td>vqm_attributes()</td>
<td>Return marker attributes.</td>
<td>7.99</td>
</tr>
<tr>
<td>37</td>
<td>vqf_attributes()</td>
<td>Return fill area attributes.</td>
<td>7.96</td>
</tr>
<tr>
<td>38</td>
<td>vqt_attributes()</td>
<td>Return text attributes.</td>
<td>7.104</td>
</tr>
<tr>
<td>39</td>
<td>vst_alignment()</td>
<td>Set graphic text alignment.</td>
<td>7.146</td>
</tr>
<tr>
<td>100</td>
<td>v_opnvwk()</td>
<td>Open virtual workstation.</td>
<td>7.61</td>
</tr>
<tr>
<td>101</td>
<td>v_clsvwk()</td>
<td>Close a virtual workstation.</td>
<td>7.35</td>
</tr>
<tr>
<td>102</td>
<td>vq_extnd()</td>
<td>Inquire workstation attributes.</td>
<td>7.89</td>
</tr>
<tr>
<td>103</td>
<td>v_contourfill()</td>
<td>Fill an irregularly shaped region.</td>
<td>7.36</td>
</tr>
<tr>
<td>104</td>
<td>vsf_perimeter()</td>
<td>Set fill perimeter visibility.</td>
<td>7.130</td>
</tr>
<tr>
<td>105</td>
<td>v_get_pixel()</td>
<td>Read screen pixel value.</td>
<td>7.55</td>
</tr>
<tr>
<td>106</td>
<td>vst_effects()</td>
<td>Set graphic text effects.</td>
<td>7.150</td>
</tr>
<tr>
<td>107</td>
<td>vst_point()</td>
<td>Set graphic text height (by point).</td>
<td>7.155</td>
</tr>
<tr>
<td>108</td>
<td>vsl_ends()</td>
<td>Set line end style.</td>
<td>7.134</td>
</tr>
<tr>
<td>109</td>
<td>vro_cpyfm()</td>
<td>Copy raster (opaque mode).</td>
<td>7.119</td>
</tr>
<tr>
<td>110</td>
<td>vr_trnfm()</td>
<td>Transform raster form.</td>
<td>7.117</td>
</tr>
<tr>
<td>111</td>
<td>vsc_form()</td>
<td>Set mouse form.</td>
<td>7.128</td>
</tr>
<tr>
<td>112</td>
<td>vsf_updat()</td>
<td>Set user defined fill pattern.</td>
<td>7.132</td>
</tr>
<tr>
<td>113</td>
<td>vsl_udsty()</td>
<td>Set user-defined line style.</td>
<td>7.136</td>
</tr>
<tr>
<td>114</td>
<td>vr_rectf()</td>
<td>Output filled rectangle.</td>
<td>7.117</td>
</tr>
<tr>
<td>115</td>
<td>vqin_mode()</td>
<td>Return input mode for device.</td>
<td>7.97</td>
</tr>
<tr>
<td>116</td>
<td>vqt_extent()</td>
<td>Return graphic text extent.</td>
<td>7.107</td>
</tr>
<tr>
<td>117</td>
<td>vqt_width()</td>
<td>Return graphic character width.</td>
<td>7.115</td>
</tr>
<tr>
<td>118</td>
<td>vex_timm()</td>
<td>Install timer tick routine.</td>
<td>7.83</td>
</tr>
<tr>
<td>119</td>
<td>vst_load_fonts()</td>
<td>Load fonts from disk.</td>
<td>7.154</td>
</tr>
<tr>
<td>120</td>
<td>vst_unload_fonts()</td>
<td>Unload fonts.</td>
<td>7.160</td>
</tr>
<tr>
<td>121</td>
<td>vrt_cpyfm()</td>
<td>Copy raster (transparent mode).</td>
<td>7.124</td>
</tr>
<tr>
<td>122</td>
<td>v_show_c()</td>
<td>Show mouse cursor.</td>
<td>7.77</td>
</tr>
<tr>
<td>123</td>
<td>v_hide_c()</td>
<td>Hide mouse cursor.</td>
<td>7.57</td>
</tr>
<tr>
<td>124</td>
<td>vq_mouse()</td>
<td>Get mouse position and state.</td>
<td>7.93</td>
</tr>
<tr>
<td>125</td>
<td>vex_butv()</td>
<td>Install mouse button routine.</td>
<td>7.80</td>
</tr>
<tr>
<td>126</td>
<td>vex_motv()</td>
<td>Install mouse movement routine.</td>
<td>7.82</td>
</tr>
<tr>
<td>127</td>
<td>vex_curv()</td>
<td>Install mouse rendering routine.</td>
<td>7.81</td>
</tr>
<tr>
<td>128</td>
<td>vq_key_s()</td>
<td>Get shift key status.</td>
<td>7.93</td>
</tr>
<tr>
<td>129</td>
<td>vs_clip()</td>
<td>Set clipping rectangle.</td>
<td>7.125</td>
</tr>
<tr>
<td>130</td>
<td>vqt_name()</td>
<td>Return font name and index.</td>
<td>7.113</td>
</tr>
<tr>
<td>131</td>
<td>vqt_fontinfo()</td>
<td>Return font size information.</td>
<td>7.111</td>
</tr>
<tr>
<td>232</td>
<td>vqt_fontheader()</td>
<td>Copy the Speedo font header into a user defined buffer.</td>
<td>7.110</td>
</tr>
<tr>
<td>234</td>
<td>vqt_trackkern()</td>
<td>Inquire about current track kerning.</td>
<td>7.114</td>
</tr>
<tr>
<td>235</td>
<td>vqt_pairkern()</td>
<td>Inquire about current pair kerning.</td>
<td>7.115</td>
</tr>
<tr>
<td>236</td>
<td>vst_charmap()</td>
<td>Set ASCII/Speedo index interpretation mode.</td>
<td>7.149</td>
</tr>
<tr>
<td>237</td>
<td>vst_kern()</td>
<td>Set kerning modes.</td>
<td>7.154</td>
</tr>
<tr>
<td>239†</td>
<td>v_getbitmap_info()</td>
<td>Return Speedo font bitmap extents.</td>
<td>7.53</td>
</tr>
<tr>
<td>240†</td>
<td>vqt_f_extent()</td>
<td>Return outline text extent.</td>
<td>7.108</td>
</tr>
<tr>
<td>Opcode, Subopcode(s) (if required)</td>
<td>Function</td>
<td>Summary</td>
<td>Page</td>
</tr>
<tr>
<td>----------------------------------</td>
<td>----------</td>
<td>---------</td>
<td>------</td>
</tr>
<tr>
<td>240†</td>
<td>vqt_f_extent16()</td>
<td>Return 16-bit outline text extent.</td>
<td>7.109</td>
</tr>
<tr>
<td>241†</td>
<td>v_ftext()</td>
<td>Output outlined text.</td>
<td>7.49</td>
</tr>
<tr>
<td>241†</td>
<td>v_ftext16()</td>
<td>Output 16-bit outlined text.</td>
<td>7.50</td>
</tr>
<tr>
<td>241†</td>
<td>v_ftext_offset()</td>
<td>Output outlined text with individual character offsets.</td>
<td>7.51</td>
</tr>
<tr>
<td>241†</td>
<td>v_ftext_offset16()</td>
<td>Output 16-bit outlined text with individual character offsets.</td>
<td>7.52</td>
</tr>
<tr>
<td>242</td>
<td>v_killoutline()</td>
<td>Free character outline (no longer used with SpeedoGDOS).</td>
<td>7.59</td>
</tr>
<tr>
<td>243</td>
<td>v_getoutline()</td>
<td>Return character outline.</td>
<td>7.54</td>
</tr>
<tr>
<td>244</td>
<td>vst_scratch()</td>
<td>Set outline scratch buffer.</td>
<td>7.157</td>
</tr>
<tr>
<td>245</td>
<td>vst_error()</td>
<td>Set GDOS error reporting mode.</td>
<td>7.151</td>
</tr>
<tr>
<td>246†</td>
<td>vst_arbpt32()</td>
<td>Set outline text point size to a fix31 value.</td>
<td>7.148</td>
</tr>
<tr>
<td>247</td>
<td>vqt_advance()</td>
<td>Return character advance vector.</td>
<td>7.102</td>
</tr>
<tr>
<td>247</td>
<td>vqt_advance32()</td>
<td>Return character advance vector as a fix31 value.</td>
<td>7.103</td>
</tr>
<tr>
<td>248</td>
<td>vqt_devinfo()</td>
<td>Return device information.</td>
<td>7.106</td>
</tr>
<tr>
<td>249</td>
<td>v_savecache()</td>
<td>Save bitmap cache to disk.</td>
<td>7.76</td>
</tr>
<tr>
<td>250</td>
<td>v_loadcache()</td>
<td>Load bitmap cache from disk.</td>
<td>7.59</td>
</tr>
<tr>
<td>251</td>
<td>v_flushcache()</td>
<td>Flush outline font cache.</td>
<td>7.47</td>
</tr>
<tr>
<td>252†</td>
<td>vst_setsize()</td>
<td>Set outline text proportion.</td>
<td>7.158</td>
</tr>
<tr>
<td>252†</td>
<td>vst_setsize32()</td>
<td>Set outline text proportion to a fix31 value.</td>
<td>7.159</td>
</tr>
<tr>
<td>253</td>
<td>vst_skew()</td>
<td>Set outline text skew factor.</td>
<td>7.160</td>
</tr>
<tr>
<td>254</td>
<td>vqt_get_table()</td>
<td>Return character mappings.</td>
<td>7.112</td>
</tr>
<tr>
<td>255</td>
<td>vqt_cachesize()</td>
<td>Return bitmap cache size</td>
<td>7.105</td>
</tr>
</tbody>
</table>

† These functions share an opcode and sub-opcode.
Usage

The information in this appendix provides a useful reference to the memory locations of the Atari computer series. While most documented locations have stayed backwardly compatible, some have changed in meaning. Software programmers directly accessing these locations should carefully consider the possibility that a location may move or not even exist in a newer version of the OS. For this reason many OS functions exist to manipulate system variables, vectors, interrupts, and devices. These should always be used, if possible, as an alternative to directly accessing hardware registers, vectors, interrupts, and variables.

WARNING!

In addition to those considerations mentioned above, directly accessing hardware registers can cause damage to hardware if not done correctly. In particular, improper use of the Falcon030 video registers could damage an attached monitor. Likewise, use of the floppy and hard drive registers can cause data loss and drive damage. For these reasons, it is strongly recommended that you avoid using hardware registers when possible, and when otherwise unavoidable, they should be used with extreme care.

Memory Map Conventions

For each Atari computer that a specific hardware location is valid for, the appropriate box will be shaded. Following is a key to several abbreviations and concepts used in this guide:

<table>
<thead>
<tr>
<th>Abbreviation</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>BYTE</td>
<td>Occupies one byte (8 bits).</td>
</tr>
<tr>
<td>WORD</td>
<td>Occupies one WORD (16 bits).</td>
</tr>
<tr>
<td>LONG</td>
<td>Occupies one longword (32 bits).</td>
</tr>
<tr>
<td>OW</td>
<td>Occupies the odd WORD of a LONG.</td>
</tr>
<tr>
<td>EW</td>
<td>Occupies the even WORD of a LONG.</td>
</tr>
<tr>
<td>OB</td>
<td>Occupies the odd BYTE of a WORD.</td>
</tr>
<tr>
<td>EB</td>
<td>Occupies the even BYTE of the WORD.</td>
</tr>
<tr>
<td>ROM</td>
<td>Location is Read-Only Memory</td>
</tr>
<tr>
<td>RAM</td>
<td>Location is Read-Write Memory</td>
</tr>
<tr>
<td>I/O</td>
<td>Location is hardware-mapped</td>
</tr>
<tr>
<td>VME</td>
<td>Location addresses VME address space</td>
</tr>
<tr>
<td>N/A</td>
<td>Not applicable</td>
</tr>
<tr>
<td>RO</td>
<td>Read-only location</td>
</tr>
<tr>
<td>WO</td>
<td>Write-only location</td>
</tr>
<tr>
<td>RW</td>
<td>Read-write location</td>
</tr>
<tr>
<td>RSVD</td>
<td>Reserved</td>
</tr>
<tr>
<td>Unassigned</td>
<td>Either not assigned or undocumented (hardware developers should always consult Atari before mapping a third-party device to a hardware location).</td>
</tr>
</tbody>
</table>
### System Boot Variables

<table>
<thead>
<tr>
<th>Location(s)</th>
<th>Size</th>
<th>Type</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>0x00000000</td>
<td>LONG</td>
<td>ROM</td>
<td>Reset: Supervisor Stack Pointer</td>
</tr>
<tr>
<td>0x00000004</td>
<td>LONG</td>
<td>ROM</td>
<td>Reset: Program Counter</td>
</tr>
</tbody>
</table>

### 68x00 Exception Vectors

<table>
<thead>
<tr>
<th>Location(s)</th>
<th>Size</th>
<th>Type</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>0x00000008</td>
<td>LONG</td>
<td>RAM</td>
<td>Bus Error Vector</td>
</tr>
<tr>
<td>0x0000000C</td>
<td>LONG</td>
<td>RAM</td>
<td>Address Error Vector</td>
</tr>
<tr>
<td>0x00000010</td>
<td>LONG</td>
<td>RAM</td>
<td>Illegal Instruction Error Vector</td>
</tr>
<tr>
<td>0x00000014</td>
<td>LONG</td>
<td>RAM</td>
<td>Divide by 0 Error Vector</td>
</tr>
<tr>
<td>0x00000018</td>
<td>LONG</td>
<td>RAM</td>
<td>CHK Instruction Exception Vector</td>
</tr>
<tr>
<td>0x0000001C</td>
<td>LONG</td>
<td>RAM</td>
<td>TRAPV, FTRAPcc, TRAPcc, cpTRAPcc Instruction Exception Vector</td>
</tr>
<tr>
<td>0x00000020</td>
<td>LONG</td>
<td>RAM</td>
<td>Privilege Violation Exception Vector</td>
</tr>
<tr>
<td>0x00000024</td>
<td>LONG</td>
<td>RAM</td>
<td>Trace Exception Vector</td>
</tr>
<tr>
<td>0x00000028</td>
<td>LONG</td>
<td>RAM</td>
<td>Line-A Exception Vector</td>
</tr>
<tr>
<td>0x0000002C</td>
<td>LONG</td>
<td>RAM</td>
<td>Line-F Exception Vector</td>
</tr>
<tr>
<td>0x00000030</td>
<td>LONG</td>
<td>RAM</td>
<td>Reserved by Motorola</td>
</tr>
<tr>
<td>0x00000034</td>
<td>LONG</td>
<td>RAM</td>
<td>Coprocessor Protocol Violation Vector</td>
</tr>
<tr>
<td>0x00000038</td>
<td>LONG</td>
<td>RAM</td>
<td>Format Error Vector</td>
</tr>
<tr>
<td>0x0000003C</td>
<td>LONG</td>
<td>RAM</td>
<td>Uninitialized Interrupt Vector</td>
</tr>
<tr>
<td>0x00000040 - 0x0000005C</td>
<td>LONG</td>
<td>RAM</td>
<td>Reserved by Motorola</td>
</tr>
<tr>
<td>0x00000060</td>
<td>LONG</td>
<td>RAM</td>
<td>Spurious Interrupt Vector (taken when an interrupt occurs during Bus Error handling)</td>
</tr>
</tbody>
</table>

### Auto-Vector Interrupts

<table>
<thead>
<tr>
<th>Location(s)</th>
<th>Size</th>
<th>Type</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>0x00000064</td>
<td>LONG</td>
<td>RAM</td>
<td>Level 1 Auto-Vector Interrupt (used if Hblank is enabled)</td>
</tr>
<tr>
<td>0x00000068</td>
<td>LONG</td>
<td>RAM</td>
<td>Level 2 Auto-Vector Interrupt (Hblank)</td>
</tr>
<tr>
<td>0x0000006C</td>
<td>LONG</td>
<td>RAM</td>
<td>Level 3 Auto-Vector Interrupt (Normal processor interrupt level)</td>
</tr>
<tr>
<td>0x00000070</td>
<td>LONG</td>
<td>RAM</td>
<td>Level 4 Auto-Vector Interrupt (Vblank)</td>
</tr>
<tr>
<td>0x00000074</td>
<td>LONG</td>
<td>RAM</td>
<td>Level 5 Auto-Vector Interrupt (currently unused)</td>
</tr>
<tr>
<td>0x00000078</td>
<td>LONG</td>
<td>RAM</td>
<td>Level 6 Auto-Vector Interrupt (MFP Interrupts)</td>
</tr>
<tr>
<td>0x0000007C</td>
<td>LONG</td>
<td>RAM</td>
<td>Level 7 Auto-Vector Interrupt (Non-maskable)</td>
</tr>
</tbody>
</table>

### TRAP Exception Vectors

<table>
<thead>
<tr>
<th>Location(s)</th>
<th>Size</th>
<th>Type</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>0x00000080</td>
<td>LONG</td>
<td>RAM</td>
<td>TRAP #0 Handler (Currently Unused)</td>
</tr>
<tr>
<td>0x00000084</td>
<td>LONG</td>
<td>RAM</td>
<td>TRAP #1 Handler (GEMDOS)</td>
</tr>
<tr>
<td>0x00000088</td>
<td>LONG</td>
<td>RAM</td>
<td>TRAP #2 Handler (AES and VDI)</td>
</tr>
<tr>
<td>Location(s)</td>
<td>Size</td>
<td>Type</td>
<td>Meaning</td>
</tr>
<tr>
<td>-------------</td>
<td>------</td>
<td>------</td>
<td>---------</td>
</tr>
<tr>
<td>0x0000008C</td>
<td>LONG</td>
<td>RAM</td>
<td>TRAP #3 Handler (Currently Unused)</td>
</tr>
<tr>
<td>0x00000090</td>
<td>LONG</td>
<td>RAM</td>
<td>TRAP #4 Handler (Currently Unused)</td>
</tr>
<tr>
<td>0x00000094</td>
<td>LONG</td>
<td>RAM</td>
<td>TRAP #5 Handler (Currently Unused)</td>
</tr>
<tr>
<td>0x00000098</td>
<td>LONG</td>
<td>RAM</td>
<td>TRAP #6 Handler (Currently Unused)</td>
</tr>
<tr>
<td>0x0000009C</td>
<td>LONG</td>
<td>RAM</td>
<td>TRAP #7 Handler (Currently Unused)</td>
</tr>
<tr>
<td>0x000000A0</td>
<td>LONG</td>
<td>RAM</td>
<td>TRAP #8 Handler (Currently Unused)</td>
</tr>
<tr>
<td>0x000000A4</td>
<td>LONG</td>
<td>RAM</td>
<td>TRAP #9 Handler (Currently Unused)</td>
</tr>
<tr>
<td>0x000000A8</td>
<td>LONG</td>
<td>RAM</td>
<td>TRAP #10 Handler (Currently Unused)</td>
</tr>
<tr>
<td>0x000000AC</td>
<td>LONG</td>
<td>RAM</td>
<td>TRAP #11 Handler (Currently Unused)</td>
</tr>
<tr>
<td>0x000000B0</td>
<td>LONG</td>
<td>RAM</td>
<td>TRAP #12 Handler (Currently Unused)</td>
</tr>
<tr>
<td>0x000000B4</td>
<td>LONG</td>
<td>RAM</td>
<td>TRAP #13 Handler (BIOS)</td>
</tr>
<tr>
<td>0x000000B8</td>
<td>LONG</td>
<td>RAM</td>
<td>TRAP #14 Handler (XBIOS)</td>
</tr>
<tr>
<td>0x000000BC</td>
<td>LONG</td>
<td>RAM</td>
<td>TRAP #15 Handler (Currently Unused)</td>
</tr>
</tbody>
</table>

### 68881 Co-processor Exception Vectors

<table>
<thead>
<tr>
<th>Location(s)</th>
<th>Size</th>
<th>Type</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>0x000000C0</td>
<td>LONG</td>
<td>RAM</td>
<td>FPCP Branch or Set on Unordered Condition Vector</td>
</tr>
<tr>
<td>0x000000C4</td>
<td>LONG</td>
<td>RAM</td>
<td>FPCP Inexact Result Vector</td>
</tr>
<tr>
<td>0x000000C8</td>
<td>LONG</td>
<td>RAM</td>
<td>FPCP Floating-Point Divide by Zero Vector</td>
</tr>
<tr>
<td>0x000000CC</td>
<td>LONG</td>
<td>RAM</td>
<td>FPCP Underflow Vector</td>
</tr>
<tr>
<td>0x000000D0</td>
<td>LONG</td>
<td>RAM</td>
<td>FPCP Operand Error Vector</td>
</tr>
<tr>
<td>0x000000D4</td>
<td>LONG</td>
<td>RAM</td>
<td>FPCP Overflow Vector</td>
</tr>
<tr>
<td>0x000000D8</td>
<td>LONG</td>
<td>RAM</td>
<td>FPCP Signaling NAN Vector</td>
</tr>
<tr>
<td>0x000000DC</td>
<td>LONG</td>
<td>RAM</td>
<td>Unassigned</td>
</tr>
</tbody>
</table>

### 68851 MMU Exception Vectors

<table>
<thead>
<tr>
<th>Location(s)</th>
<th>Size</th>
<th>Type</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>0x000000E0</td>
<td>LONG</td>
<td>RAM</td>
<td>MMU Configuration Error Vector</td>
</tr>
<tr>
<td>0x000000E4</td>
<td>LONG</td>
<td>RAM</td>
<td>MMU Illegal Operation Vector</td>
</tr>
<tr>
<td>0x000000E8</td>
<td>LONG</td>
<td>RAM</td>
<td>MMU Access Violation Vector</td>
</tr>
<tr>
<td>0x000000EC – 0x000000FC</td>
<td>LONG</td>
<td>RAM</td>
<td>Reserved by Motorola</td>
</tr>
</tbody>
</table>

### Multi-Function Peripheral Port Vectors

<table>
<thead>
<tr>
<th>Location(s)</th>
<th>Size</th>
<th>Type</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>0x00000100</td>
<td>LONG</td>
<td>RAM</td>
<td>MFP #0: Parallel-Port Interrupt Vector</td>
</tr>
<tr>
<td>0x00000104</td>
<td>LONG</td>
<td>RAM</td>
<td>MFP #1: RS-232 Carrier Detect Vector (On a Falcon030, this MFP interrupt is connected to the parallel port ‘Acknowledge’ signal, not the RS-232 port.)</td>
</tr>
<tr>
<td>0x00000108</td>
<td>LONG</td>
<td>RAM</td>
<td>MFP #2: RS-232 Clear to Send Vector</td>
</tr>
<tr>
<td>0x0000010C</td>
<td>LONG</td>
<td>RAM</td>
<td>MFP #3: BLITTER Operation Complete (when hardware BLITTER is present)</td>
</tr>
<tr>
<td>Location(s)</td>
<td>Size</td>
<td>Type</td>
<td>Meaning</td>
</tr>
<tr>
<td>-------------</td>
<td>------</td>
<td>------</td>
<td>---------</td>
</tr>
<tr>
<td>0x00000110</td>
<td>LONG</td>
<td>RAM</td>
<td>Timer D: RS-232 Baud Rate Generator</td>
</tr>
<tr>
<td>0x00000114</td>
<td>LONG</td>
<td>RAM</td>
<td>Timer C: 200 Hz System Clock</td>
</tr>
<tr>
<td>0x00000118</td>
<td>LONG</td>
<td>RAM</td>
<td>MFP #4: Keyboard/MIDI (6850 processor)</td>
</tr>
<tr>
<td>0x0000011C</td>
<td>LONG</td>
<td>RAM</td>
<td>MFP #5: Floppy/Hard Disk Controller</td>
</tr>
<tr>
<td>0x00000120</td>
<td>LONG</td>
<td>RAM</td>
<td>Timer B: Horizontal Blank Counter</td>
</tr>
<tr>
<td>0x00000124</td>
<td>LONG</td>
<td>RAM</td>
<td>RS-232 Transmit Error Interrupt</td>
</tr>
<tr>
<td>0x00000128</td>
<td>LONG</td>
<td>RAM</td>
<td>RS-232 Transmit Buffer Error Interrupt</td>
</tr>
<tr>
<td>0x0000012C</td>
<td>LONG</td>
<td>RAM</td>
<td>RS-232 Receive Error Interrupt</td>
</tr>
<tr>
<td>0x00000130</td>
<td>LONG</td>
<td>RAM</td>
<td>RS-232 Receive Buffer Full Interrupt</td>
</tr>
<tr>
<td>0x00000134</td>
<td>LONG</td>
<td>RAM</td>
<td>Timer A: DMA Sound Complete</td>
</tr>
<tr>
<td>0x00000138</td>
<td>LONG</td>
<td>RAM</td>
<td>MFP #6: RS-232 Ring Indicator (On a Falcon030, this is the only Serial port vector that remains part of the MFP. All other Serial port functions have been transferred to the SCC.)</td>
</tr>
<tr>
<td>0x0000013C</td>
<td>LONG</td>
<td>RAM</td>
<td>MFP #7: Monochrome Monitor Detect</td>
</tr>
<tr>
<td>0x00000140</td>
<td>LONG</td>
<td>RAM</td>
<td>MFP #0: General Purpose I/O Pin</td>
</tr>
<tr>
<td>0x00000144</td>
<td>LONG</td>
<td>RAM</td>
<td>MFP #1: General Purpose I/O Pin</td>
</tr>
<tr>
<td>0x00000148</td>
<td>LONG</td>
<td>RAM</td>
<td>MFP #2: SCC DMAC Interrupt</td>
</tr>
<tr>
<td>0x0000014C</td>
<td>LONG</td>
<td>RAM</td>
<td>MFP #3: RS-232 Ring Indicator</td>
</tr>
<tr>
<td>0x00000150</td>
<td>LONG</td>
<td>RAM</td>
<td>Timer D: RS-232 Baud Rate Generator</td>
</tr>
<tr>
<td>0x00000154</td>
<td>LONG</td>
<td>RAM</td>
<td>Timer C: SCC TRxCB</td>
</tr>
<tr>
<td>0x00000158</td>
<td>LONG</td>
<td>RAM</td>
<td>MFP #4: Reserved</td>
</tr>
<tr>
<td>0x0000015C</td>
<td>LONG</td>
<td>RAM</td>
<td>MFP #5: SCSI DMAC Interrupt</td>
</tr>
<tr>
<td>0x00000160</td>
<td>LONG</td>
<td>RAM</td>
<td>Timer B: Unassigned</td>
</tr>
<tr>
<td>0x00000164</td>
<td>LONG</td>
<td>RAM</td>
<td>RS-232 Transmit Error Interrupt</td>
</tr>
<tr>
<td>0x00000168</td>
<td>LONG</td>
<td>RAM</td>
<td>RS-232 Transmit Buffer Error Interrupt</td>
</tr>
<tr>
<td>0x0000016C</td>
<td>LONG</td>
<td>RAM</td>
<td>RS-232 Receive Error Interrupt</td>
</tr>
<tr>
<td>0x00000170</td>
<td>LONG</td>
<td>RAM</td>
<td>RS-232 Receive Buffer Full Interrupt</td>
</tr>
<tr>
<td>0x00000174</td>
<td>LONG</td>
<td>RAM</td>
<td>Timer A: Reserved</td>
</tr>
<tr>
<td>0x00000178</td>
<td>LONG</td>
<td>RAM</td>
<td>MFP #6: RTC IRQ</td>
</tr>
<tr>
<td>0x0000017C</td>
<td>LONG</td>
<td>RAM</td>
<td>MFP #7: SCSI Controller IRQ</td>
</tr>
<tr>
<td>0x00000180</td>
<td>LONG</td>
<td>RAM</td>
<td>SCC Port B Transmit Buffer Empty Vector</td>
</tr>
<tr>
<td>0x00000184</td>
<td>LONG</td>
<td>RAM</td>
<td>Unused</td>
</tr>
<tr>
<td>0x00000188</td>
<td>LONG</td>
<td>RAM</td>
<td>SCC Port B External Status Change Vector</td>
</tr>
<tr>
<td>0x0000018C</td>
<td>LONG</td>
<td>RAM</td>
<td>Unused</td>
</tr>
</tbody>
</table>

**Multi-Function Peripheral Port Vectors (TT)**

**Zilog 85C30 (SCC) Interrupt Vectors**
<table>
<thead>
<tr>
<th>Location(s)</th>
<th>Size</th>
<th>Type</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>0x00000190</td>
<td>LONG</td>
<td>RAM</td>
<td>SCC Port B Receive Character Available Vector</td>
</tr>
<tr>
<td>0x00000194</td>
<td>LONG</td>
<td>RAM</td>
<td>Unused</td>
</tr>
<tr>
<td>0x00000198</td>
<td>LONG</td>
<td>RAM</td>
<td>SCC Port B Special Receive Condition Vector</td>
</tr>
<tr>
<td>0x0000019C</td>
<td>LONG</td>
<td>RAM</td>
<td>Unused</td>
</tr>
<tr>
<td>0x000001A0</td>
<td>LONG</td>
<td>RAM</td>
<td>SCC Port A Transmit Buffer Empty Vector</td>
</tr>
<tr>
<td>0x000001A4</td>
<td>LONG</td>
<td>RAM</td>
<td>Unused</td>
</tr>
<tr>
<td>0x000001A8</td>
<td>LONG</td>
<td>RAM</td>
<td>SCC Port A External Status Change Vector</td>
</tr>
<tr>
<td>0x000001AC</td>
<td>LONG</td>
<td>RAM</td>
<td>Unused</td>
</tr>
<tr>
<td>0x000001B0</td>
<td>LONG</td>
<td>RAM</td>
<td>SCC Port A Receive Character Available Vector</td>
</tr>
<tr>
<td>0x000001B4</td>
<td>LONG</td>
<td>RAM</td>
<td>Unused</td>
</tr>
<tr>
<td>0x000001B8</td>
<td>LONG</td>
<td>RAM</td>
<td>SCC Port A Special Receive Condition Vector</td>
</tr>
<tr>
<td>0x000001BC</td>
<td>LONG</td>
<td>RAM</td>
<td>Unused</td>
</tr>
<tr>
<td>0x000001C0 – 0x0000037F</td>
<td>N/A</td>
<td>RAM</td>
<td>Undefined</td>
</tr>
<tr>
<td>0x00000380</td>
<td>LONG</td>
<td>RAM</td>
<td>proc_lives: If, after a system failure, the operating system is able to save the processor state in the following variables, this value will be 0x12345678.</td>
</tr>
<tr>
<td>0x00000384</td>
<td>LONG</td>
<td>RAM</td>
<td>proc_dregs: The contents of registers D0 through D7 are stored here.</td>
</tr>
<tr>
<td>0x000003A4</td>
<td>LONG</td>
<td>RAM</td>
<td>proc_arests: The contents of registers A0 through A7 are stored here.</td>
</tr>
<tr>
<td>0x000003C4</td>
<td>LONG</td>
<td>RAM</td>
<td>proc_pc: The first byte of this longword indicates the exception number that occurred.</td>
</tr>
<tr>
<td>0x000003C8</td>
<td>LONG</td>
<td>RAM</td>
<td>proc_usp: The user stack pointer (USP) is saved here.</td>
</tr>
<tr>
<td>0x000003CC – 0x000003EA</td>
<td>WORD</td>
<td>RAM</td>
<td>proc_stk: The top 16 WORDs of the supervisor stack are saved here.</td>
</tr>
<tr>
<td>0x000003EC – 0x000003FF</td>
<td>N/A</td>
<td>RAM</td>
<td>Unassigned</td>
</tr>
<tr>
<td>0x00000400</td>
<td>LONG</td>
<td>RAM</td>
<td>etv_timer: System Timer Handoff Vector (see GEMDOS)</td>
</tr>
<tr>
<td>0x00000404</td>
<td>LONG</td>
<td>RAM</td>
<td>etv_critic: Critical Error Handoff Vector (see GEMDOS)</td>
</tr>
<tr>
<td>0x00000408</td>
<td>LONG</td>
<td>RAM</td>
<td>etv_term: Process Termination Handler (see GEMDOS)</td>
</tr>
<tr>
<td>0x0000040C – 0x0000041C</td>
<td>LONG</td>
<td>RAM</td>
<td>Reserved for future vectors.</td>
</tr>
</tbody>
</table>
### System Variables

<table>
<thead>
<tr>
<th>Location(s)</th>
<th>Size</th>
<th>Type</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>0x00000420</td>
<td>LONG</td>
<td>RAM</td>
<td>memvalid: If this variable is equal to $752019F3 and the value at memval2 ($43A) is also correct, then the last coldstart was successful and memcntlr ($424) is valid. As of TOS 1.02 memval3 ($51A) must also be correct.</td>
</tr>
<tr>
<td>0x00000424</td>
<td>WORD</td>
<td>RAM</td>
<td>memcntlr: Bits 11–8 of this WORD contains the memory controller state.</td>
</tr>
<tr>
<td>0x00000426</td>
<td>LONG</td>
<td>RAM</td>
<td>resvalid: If this location contains the magic number $31415926 then the system will jump through resvector (below) on a system reset.</td>
</tr>
<tr>
<td>0x0000042A</td>
<td>LONG</td>
<td>RAM</td>
<td>resvector: If the magic number in resvalid is set properly, this vector will be jumped through on a system reset with the return address placed in A6.</td>
</tr>
<tr>
<td>0x0000042E</td>
<td>LONG</td>
<td>RAM</td>
<td>phystop: Physical top of ST compatible RAM.</td>
</tr>
<tr>
<td>0x00000432</td>
<td>LONG</td>
<td>RAM</td>
<td>_membot: This value points to the lowest memory location available for the system heap. This value is used to initialize GEMDOS free memory.</td>
</tr>
<tr>
<td>0x00000436</td>
<td>LONG</td>
<td>RAM</td>
<td>_memtop: This value points to the highest memory location available for the system heap. This value is used to initialize GEMDOS free memory.</td>
</tr>
<tr>
<td>0x0000043A</td>
<td>LONG</td>
<td>RAM</td>
<td>memval2: This value will equal $237698AA if coldstart was successful. See memvalid ($420).</td>
</tr>
<tr>
<td>0x0000043E</td>
<td>WORD</td>
<td>RAM</td>
<td>flock: This variable should be set to non-zero prior to accessing the DMA registers to prevent the system or other processes from attempting DMA concurrently.</td>
</tr>
<tr>
<td>0x00000440</td>
<td>WORD</td>
<td>RAM</td>
<td>seekrate: This variable sets the floppy drive seek rate for both floppy drives as follows:</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td><strong>Value</strong></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>0</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>1</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>2</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>3</td>
</tr>
<tr>
<td>0x00000442</td>
<td>WORD</td>
<td>RAM</td>
<td>_timr_ms: This value indicates the time between system timer ticks in milliseconds. Current machines have the value of 20 (0x14) equating to 50 timer updates per second. This value is returned by the BIOS function Tickcal() and is placed on the stack prior to jumping through the timer handoff vector ($400).</td>
</tr>
<tr>
<td>Location(s)</td>
<td>Size</td>
<td>Type</td>
<td>Meaning</td>
</tr>
<tr>
<td>-------------</td>
<td>------</td>
<td>------</td>
<td>---------</td>
</tr>
<tr>
<td>0x00000444</td>
<td>WORD</td>
<td>RAM</td>
<td>_fverify: When non-zero, all floppy writes are verified, otherwise, no verification is done.</td>
</tr>
<tr>
<td>0x00000446</td>
<td>WORD</td>
<td>RAM</td>
<td>_bootdev: This value represents the device from which the system was booted (0 = A:, 1 = B:, etc.)</td>
</tr>
<tr>
<td>0x00000448</td>
<td>WORD</td>
<td>RAM</td>
<td>_palmode: A value of 0 indicates that NTSC video is being used, otherwise, PAL is being used.</td>
</tr>
<tr>
<td>0x0000044A</td>
<td>WORD</td>
<td>RAM</td>
<td>_defshftmd: This value indicates the default video shifter mode.</td>
</tr>
<tr>
<td>0x0000044C</td>
<td>WORD</td>
<td>RAM</td>
<td>_sshftmd: This value is a copy of the hardware register at 0x00FF8260 which indicates the current ST shifter mode.</td>
</tr>
<tr>
<td>0x0000044E</td>
<td>LONG</td>
<td>RAM</td>
<td>_v_bas_ad: This indicates the starting address of the logical screen. Prior to TOS 1.06, this address needed to be aligned on a 256 byte boundary. As of TOS 1.06, it may be WORD aligned.</td>
</tr>
<tr>
<td>0x00000452</td>
<td>WORD</td>
<td>RAM</td>
<td>_vblsem: A value of 0 here disables all vertical blank processing while a value of 1 enables it.</td>
</tr>
<tr>
<td>0x00000454</td>
<td>WORD</td>
<td>RAM</td>
<td>_nvbls: This value indicates the number of slots in the deferred vertical blank handler list. If all table slots are full and your application needs to install a handler, it may allocate a new, larger list, update this value and the pointer below.</td>
</tr>
<tr>
<td>0x00000456</td>
<td>LONG</td>
<td>RAM</td>
<td>_vblqueue: This is a pointer to a list of pointers to the deferred vertical blank handlers. Each pointer in the list pointed to by this variable which contains a value other than 0 is 'JSR'ed' through at each vertical blank. This occurs 50 times per second on PAL color monitors, 60 times per second on NTSC color monitors and 70 times per second on all monochrome monitors.</td>
</tr>
<tr>
<td>0x0000045A</td>
<td>LONG</td>
<td>RAM</td>
<td>_colorptr: If this value is non-zero then at the next vertical blank, the 16 color registers pointed to by this value will be loaded into the hardware registers.</td>
</tr>
<tr>
<td>0x0000045E</td>
<td>LONG</td>
<td>RAM</td>
<td>_screenpt: If this value is non-zero then at the next vertical blank, the value stored here will be loaded into the hardware register which points to the base of the physical screen.</td>
</tr>
<tr>
<td>0x00000462</td>
<td>LONG</td>
<td>RAM</td>
<td>_vbclock: This value indicates the number of vertical blanks that have been processed since the last reset.</td>
</tr>
</tbody>
</table>
### B.10 – Memory Map

<table>
<thead>
<tr>
<th>Location(s)</th>
<th>Size</th>
<th>Type</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>0x00000466</td>
<td>LONG</td>
<td>RAM</td>
<td><code>.frlock</code>: This value indicates the number of vertical blanks regardless of whether they were processed or not (blocked by <code>vblsem</code>).</td>
</tr>
<tr>
<td>0x0000046A</td>
<td>LONG</td>
<td>RAM</td>
<td><code>hdv_init</code>: This value points the hard disk initialization routine or is 0 to indicate that no hard disk is installed.</td>
</tr>
<tr>
<td>0x0000046E</td>
<td>LONG</td>
<td>RAM</td>
<td><code>swv_vec</code>: The vector pointed to by this routine is called when the system detects a change in monitors (normally this points to the reset handler).</td>
</tr>
<tr>
<td>0x00000472</td>
<td>LONG</td>
<td>RAM</td>
<td><code>hdv_bpb</code>: This vector is used when <code>Getbpb()</code> is called. A value of 0 indicates that no hard disk is attached.</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>Applications installing themselves here should expect parameters to be located on the stack as they would be for the actual function call beginning at 4(sp). If the installed process services the call it should RTS, otherwise, leaving the stack intact, should JMP through the old vector value.</td>
</tr>
<tr>
<td>0x00000476</td>
<td>LONG</td>
<td>RAM</td>
<td><code>hdv_rw</code>: This vector is used when <code>Rwabs()</code> is called. A value of 0 here indicates that no hard disk is attached.</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>Applications installing themselves here should expect parameters to be located on the stack as they would be for the actual function call beginning at 4(sp). If the installed process services the call it should RTS, otherwise, leaving the stack intact, should JMP through the old vector value.</td>
</tr>
<tr>
<td>0x0000047A</td>
<td>LONG</td>
<td>RAM</td>
<td><code>hdv_boot</code>: This vector is JSR'ed through to boot from the hard disk. A value of 0 here indicates that no hard disk is attached. If the installed process services the call it should RTS, otherwise, leaving the stack intact, should JMP through the old vector value.</td>
</tr>
<tr>
<td>0x0000047E</td>
<td>LONG</td>
<td>RAM</td>
<td><code>hdv_mediach</code>: This vector is used when <code>Mediach()</code> is called. A value of 0 here indicates that no hard disk is attached.</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>Applications installing themselves here should expect parameters to be located on the stack as they would be for the actual function call beginning at 4(sp). If the installed process services the call it should RTS, otherwise, leaving the stack intact, should JMP through the old vector value.</td>
</tr>
<tr>
<td>Location(s)</td>
<td>Size</td>
<td>Type</td>
<td>Meaning</td>
</tr>
<tr>
<td>------------</td>
<td>--------</td>
<td>-------</td>
<td>-------------------------------------------------------------------------</td>
</tr>
<tr>
<td>0x00000482</td>
<td>WORD</td>
<td>RAM</td>
<td>_cmdload: During boot if this location contains a non-zero value, the system will attempt to load “COMMAND.PRG” from the boot device rather than initializing the GEM Desktop.</td>
</tr>
<tr>
<td>0x00000484</td>
<td>BYTE</td>
<td>RAM</td>
<td>_conterm: This location contains a bit array which determine several system attributes as follows:</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td><strong>Bit</strong></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>0</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>1</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>2</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>3</td>
</tr>
<tr>
<td>0x00000485</td>
<td>BYTE</td>
<td>RAM</td>
<td>Reserved</td>
</tr>
<tr>
<td>0x00000486</td>
<td>LONG</td>
<td>RAM</td>
<td>_trp14ret: This value is used by Trap #14 OS code to store the return address.</td>
</tr>
<tr>
<td>0x0000048A</td>
<td>LONG</td>
<td>RAM</td>
<td>_criticret: This value is used by <em>etv_critic</em> handling code to store the return address.</td>
</tr>
<tr>
<td>0x0000048E – 0x0000049D</td>
<td>BYTE</td>
<td>RAM</td>
<td>_themd: This is the <strong>MD</strong> (Memory Descriptor structure) initialized by the <strong>BIOS</strong> at boot and returned by <strong>Getmpb()</strong>.</td>
</tr>
<tr>
<td>0x0000049E</td>
<td>LONG</td>
<td>RAM</td>
<td>_md: This is a pointer to additional <strong>MD</strong> structures.</td>
</tr>
<tr>
<td>0x000004A2</td>
<td>LONG</td>
<td>RAM</td>
<td>_savptr: This is a pointer to the buffer which the <strong>BIOS</strong> uses to save internal registers.</td>
</tr>
<tr>
<td>0x000004A6</td>
<td>WORD</td>
<td>RAM</td>
<td>_nflops: This value indicates the number of floppy drives currently connected to the system.</td>
</tr>
<tr>
<td>0x000004A8</td>
<td>LONG</td>
<td>RAM</td>
<td>_con_state: This is a vector to internal console output routines which is set to various VT-52 ESC functions.</td>
</tr>
<tr>
<td>0x000004AC</td>
<td>WORD</td>
<td>RAM</td>
<td>_save_row: This value contains the row number of the cursor temporarily when using the ESC-Y VT-52 sequence.</td>
</tr>
<tr>
<td>0x000004AE</td>
<td>LONG</td>
<td>RAM</td>
<td>_sav_contxt: This points to a temporary buffer where the processor context is saved.</td>
</tr>
<tr>
<td>0x000004B2 – 0x000004B6</td>
<td>LONG</td>
<td>RAM</td>
<td>_bcb: The first longword here points to a <strong>BCB</strong> (Buffer Control Block) used to store data sectors. The second longword points to a <strong>BCB</strong> which is used to store FAT and directory sectors.</td>
</tr>
<tr>
<td>0x000004BA</td>
<td>LONG</td>
<td>RAM</td>
<td>_hz_200: This value is an ongoing counter for the internal 200Hz clock. It is used as a seed value for the <strong>Random()</strong> function.</td>
</tr>
<tr>
<td>Location(s)</td>
<td>Size</td>
<td>Type</td>
<td>Meaning</td>
</tr>
<tr>
<td>------------</td>
<td>--------</td>
<td>--------</td>
<td>-------------------------------------------------------------------------</td>
</tr>
<tr>
<td>0x000004BE</td>
<td>LONG</td>
<td>RAM</td>
<td>the_env: This longword is the default environment string (four zeros).</td>
</tr>
<tr>
<td>0x000004C2</td>
<td>LONG</td>
<td>RAM</td>
<td>_drvbits: Each of 32 bits in this longword represents a drive connected to the system. Bit #0 is A, Bit #1 is B and so on. If at least one floppy is connected to the system, both floppy bits will always be set because of virtual swapping.</td>
</tr>
<tr>
<td>0x000004C6</td>
<td>LONG</td>
<td>RAM</td>
<td>_dskbufp: This variable points to a 1K disk operation buffer and is also used by some graphics functions.</td>
</tr>
<tr>
<td>0x000004CA</td>
<td>LONG</td>
<td>RAM</td>
<td>_autopath: This variable points to the GEMDOS path specification of the directory to load ‘AUTO’ folder programs from (may be NULL to indicate default).</td>
</tr>
<tr>
<td>0x000004CE – 0x000004EA</td>
<td>LONG</td>
<td>RAM</td>
<td>_vbl_list: This area is used by the system for the initial deferred vertical blank list.</td>
</tr>
<tr>
<td>0x000004EE</td>
<td>WORD</td>
<td>RAM</td>
<td>_prt_cnt: This value is used by the ALT-HELP screen dump code and is initialized to 0xFFFF. Each time ALT-HELP is pressed, this value is incremented. Custom screen dump code should check this value on entry and if 0 begin a screen dump, otherwise, abort the dump, reset the value to 0xFFFF and return.</td>
</tr>
<tr>
<td>0x000004F0</td>
<td>WORD</td>
<td>RAM</td>
<td>_prtabt: Flag is set to abort printing because of a timeout.</td>
</tr>
</tbody>
</table>
### Location(s) Size

<table>
<thead>
<tr>
<th>Location(s)</th>
<th>Size</th>
<th>Type</th>
<th>Meaning</th>
</tr>
</thead>
</table>
| 0x000004F2  | LONG | RAM  | _sysbase_: This value points to the beginning of the TOS operating system. The beginning of the OS contains a structure as follows:

```c
typedef struct _osheader {
    /* BRA to Reset Code */
    UWORD os_entry;
    /* TOS Version */
    UWORD os_version;
    /* Reset Code */
    VOID *reseth;
    /* Pointer to OSBASE */
    struct _osheader *os_beg;
    /* Pointer to OS end*/
    VOID *os_end;
    /* Reserved */
    LONG os_rsv1;
    /* Memory Usage PB */
    GEM_MUPB *os_magic;
    /* OS Date $YYYYMMDD */
    LONG os_date;
    /* OS Conf. Bits */
    UWORD os_conf;
    /* DOS OS Date */
    UWORD os_dosdate;
    /* As of TOS 1.2 */
    /* Base of OS Pool */
    char **p_root;
    /* Key. Shift State */
    char **pkbshift;
    /* Current process */
    BASEPAGE **p_run;
    /* Reserved */
    char *p_rsv2;
} OSHEADER;
```

<p>| 0x000004F6  | LONG | RAM  | <em>shell_p</em>: Normally not utilized, this vector allows a shell process to be installed which expects to be called with a pointer to a CLI-type command to be at 4(sp). If a command handler does not exist, this value will be NULL. |
| 0x000004FA  | LONG | RAM  | end_os: This value points to the end of RAM utilized by TOS (copied into membot). |</p>
<table>
<thead>
<tr>
<th>Location(s)</th>
<th>Size</th>
<th>Type</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>0x000004FE</td>
<td>LONG</td>
<td>RAM</td>
<td><code>exec_os</code>: This vector is jumped through when operating system initialization is complete (normally points to the Desktop/AES startup code).</td>
</tr>
<tr>
<td>0x00000502</td>
<td>LONG</td>
<td>RAM</td>
<td><code>scr_dump</code>: The routine pointed to by this value is called each time the user pressed ALT-HELP.</td>
</tr>
<tr>
<td>0x00000506</td>
<td>LONG</td>
<td>RAM</td>
<td><code>prv_lsto</code>: This vector is called to check the status of the ‘PRN:’ output device by the <code>Prtblk()</code> routine.</td>
</tr>
<tr>
<td>0x0000050A</td>
<td>LONG</td>
<td>RAM</td>
<td><code>prv_lst</code>: This vector is called to output a byte to the ‘PRN:’ device by the <code>Prtblk()</code> routine.</td>
</tr>
<tr>
<td>0x0000050E</td>
<td>LONG</td>
<td>RAM</td>
<td><code>prv_auxo</code>: This vector is called to check the status of the ‘AUX:’ output device by the <code>Prtblk()</code> routine.</td>
</tr>
<tr>
<td>0x00000512</td>
<td>LONG</td>
<td>RAM</td>
<td><code>prv_aux</code>: This vector is called to output a byte to the ‘AUX:’ device by the <code>Prtblk()</code> routine.</td>
</tr>
</tbody>
</table>
### System Variables – B.15

<table>
<thead>
<tr>
<th>Location(s)</th>
<th>Size</th>
<th>Type</th>
<th>Meaning</th>
</tr>
</thead>
</table>
| 0x00000516  | LONG | RAM  | *pun_ptr: This points to a structure used by AHDI as follows:*

```c
/* # supported drives */
#define MAXUNITS 16

typedef struct
{
    /* Maximum # of drives
     * supported by system,
     * including floppies.
    */
    WORD puns;
    /* Bit 0-2 indicates
     * the physical ACSI unit
     * it resides on.
     * Bit 7 = 0 indicates
     * that the drive exists
    */
    BYTE pun[MAXUNITS];
    /* Indicates offset in
     * physical sectors (512
     * bytes) to the start of
     * partition.
    */
    LONG prt_start[MAXUNITS];

    /* The following are
     * only present as of
     * AHDI 3.0. */

    /* Cookie is $41484449 */
    LONG P_cookie;
    /* Points to P_cookie */
    LONG *P_cookptr;
    /* Version of AHDI */
    UWORD P_version;
    /* Size of the largest
     * logical sector. */
    UWORD P_max_sector;
    /* Reserved */
    LONG reserved[MAXUNITS];
} PUN_INFO;
```

| 0x0000051A  | LONG | RAM  | *memval3: Will equal $5555AAAA if coldstart was successful. See memvalid($420).* |
## B.16 – Memory Map

<table>
<thead>
<tr>
<th>Location(s)</th>
<th>Size</th>
<th>Type</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>0x0000051E – 0x0000053A</td>
<td>LONG</td>
<td>RAM</td>
<td>xconstat: This location contains eight pointers to the BIOS Bconstat() functions for eight BIOS devices.</td>
</tr>
<tr>
<td>0x0000053E – 0x0000055A</td>
<td>LONG</td>
<td>RAM</td>
<td>xconin: This location contains eight pointers to the BIOS Bconin() functions for eight BIOS devices.</td>
</tr>
<tr>
<td>0x0000055E – 0x0000056A</td>
<td>LONG</td>
<td>RAM</td>
<td>xcostat: This location contains eight pointers to the BIOS Bcostat() functions for eight BIOS devices.</td>
</tr>
<tr>
<td>0x0000057E – 0x0000059A</td>
<td>LONG</td>
<td>RAM</td>
<td>xconout: This location contains eight pointers to the BIOS Bconout() functions for eight BIOS devices.</td>
</tr>
<tr>
<td>0x0000059E</td>
<td>WORD</td>
<td>RAM</td>
<td>_longframe: If this value is 0 then the processor uses short stack frames, otherwise it uses long stack frames. This value is of interest to applications which intercept TRAP handlers. When using short stack frames, the first parameter will be found at 6(sp), otherwise at 8(sp).</td>
</tr>
<tr>
<td>0x000005A0</td>
<td>LONG</td>
<td>RAM</td>
<td>_p_cookies: This is a pointer to the system Cookie Jar.</td>
</tr>
<tr>
<td>0x000005A4</td>
<td>LONG</td>
<td>RAM</td>
<td>ramtop: If ramvalid is correct, this is a pointer to the end of alternative RAM.</td>
</tr>
<tr>
<td>0x000005A8</td>
<td>LONG</td>
<td>RAM</td>
<td>ramvalid: This value should be $1357BD13 to indicate that ramtop is correct.</td>
</tr>
<tr>
<td>0x000005AC</td>
<td>LONG</td>
<td>RAM</td>
<td>bell_hhook: This vector is jumped through to sound the system bell.</td>
</tr>
<tr>
<td>0x000005B0</td>
<td>LONG</td>
<td>RAM</td>
<td>kcl_hhook: This vector is jumped through to sound system key clicks. The scancode of the current character is placed in the low byte of D0.</td>
</tr>
</tbody>
</table>

### System RAM / Expansion

<table>
<thead>
<tr>
<th>Location(s)</th>
<th>Size</th>
<th>Type</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>0x000005B4 – 0x009FFFFFFF</td>
<td>BYTE</td>
<td>RAM/ROM</td>
<td>This area contains whatever remaining ST compatible RAM is available. Additional space at this location is utilized by the operating system. Memory locations below 0x00E00000 on a machine other than the Mega STe or below 0x00A00000 on a Mega STe that are not part of this RAM may be utilized by hardware developers.</td>
</tr>
<tr>
<td>0x00A00000 – 0x00DEFFFFF</td>
<td>BYTE</td>
<td>VME/RAM</td>
<td>On a Mega STe, this area is mapped to VME A24:D16 address space, otherwise it may be mapped to additional ST compatible RAM or I/O space. Falcon030 computers use this address space for RAM.</td>
</tr>
<tr>
<td>Location(s)</td>
<td>Size</td>
<td>Type</td>
<td>Meaning</td>
</tr>
<tr>
<td>---------------------</td>
<td>--------</td>
<td>------------</td>
<td>-------------------------------------------------------------------------</td>
</tr>
<tr>
<td>0x00DF0000 – 0x00DFFFFF</td>
<td>BYTE</td>
<td>VME/RAM</td>
<td>On a Mega STe, this area is mapped to VME A16:D16 address space, otherwise it may be mapped to additional ST compatible RAM or I/O space. Falcon030 computers use this address space for RAM.</td>
</tr>
<tr>
<td>0x00E00000 – 0x00EFFFFF</td>
<td>BYTE</td>
<td>ROM</td>
<td>Operating system ROM's as of TOS 1.06.</td>
</tr>
</tbody>
</table>

### IDE Controller

<table>
<thead>
<tr>
<th>Location(s)</th>
<th>Size</th>
<th>Type</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>0x00F00000</td>
<td>OW</td>
<td>I/O</td>
<td>Data Register</td>
</tr>
<tr>
<td>0x00F00004</td>
<td>OB</td>
<td>I/O</td>
<td>Error Register as follows:</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>Bit 7</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>Bad Block Mark</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>Uncorrectable Error</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>ID Field Not Found</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>Command Aborted</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>Track 0 Not Found</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>DAM Not Found</td>
</tr>
<tr>
<td>0x00F00006</td>
<td>N/A</td>
<td></td>
<td>Unused</td>
</tr>
<tr>
<td>0x00F00008</td>
<td>OB</td>
<td>I/O</td>
<td>Sector Count Register</td>
</tr>
<tr>
<td>0x00F0000A</td>
<td>N/A</td>
<td>I/O</td>
<td>Unused</td>
</tr>
<tr>
<td>0x00F0000C</td>
<td>OB</td>
<td>I/O</td>
<td>Sector Number Register</td>
</tr>
<tr>
<td>0x00F0000E</td>
<td>N/A</td>
<td>I/O</td>
<td>Unused</td>
</tr>
<tr>
<td>0x00F00010</td>
<td>OB</td>
<td>I/O</td>
<td>Cylinder Low Register (this register is written with the low eight bits of the ten bit cylinder number).</td>
</tr>
<tr>
<td>0x00F00012</td>
<td>N/A</td>
<td>I/O</td>
<td>Unused</td>
</tr>
<tr>
<td>0x00F00014</td>
<td>OB</td>
<td>I/O</td>
<td>Cylinder High Register (this register is written with the high two bits of the ten bit cylinder number).</td>
</tr>
<tr>
<td>0x00F00016</td>
<td>N/A</td>
<td>I/O</td>
<td>Unused</td>
</tr>
<tr>
<td>0x00F00018</td>
<td>OB</td>
<td>I/O</td>
<td>Drive Head Register as follows:</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>Drive Select (0 = Master, 1 = Slave)</td>
</tr>
<tr>
<td></td>
<td>Bit 7</td>
<td></td>
<td>Bit 0</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>Head Number (0-15)</td>
</tr>
</tbody>
</table>

---

**THE ATARI COMPENDIUM**
### B.18 – Memory Map

<table>
<thead>
<tr>
<th>Location(s)</th>
<th>Size</th>
<th>Type</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>0x00F0001A – 0x00F0001D</td>
<td>N/A</td>
<td>I/O</td>
<td>Unused</td>
</tr>
<tr>
<td>0x00F0001E</td>
<td>OB</td>
<td>I/O</td>
<td>Status Register (on read) as follows:</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td><img src="" alt="Status Register Diagram" /></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>Command Register (on write). The IDE registers must be completely setup prior to writing the command byte here.</td>
</tr>
<tr>
<td>0x00F00020 – 0x00F00036</td>
<td>N/A</td>
<td>I/O</td>
<td>Unused</td>
</tr>
<tr>
<td>0x00F00038</td>
<td>OB</td>
<td>I/O</td>
<td>Alternate Status Register (on read)</td>
</tr>
<tr>
<td>0x00F00040 – 0x00F00036</td>
<td>N/A</td>
<td>I/O</td>
<td>Alternate Command Register (on write)</td>
</tr>
<tr>
<td>0x00FA0000 – 0x00FBFFFF</td>
<td>BYTE</td>
<td>ROM</td>
<td>Cartridge ROM</td>
</tr>
<tr>
<td>0x00FC0000 – 0x00FEFFFF</td>
<td>BYTE</td>
<td>ROM</td>
<td>On pre TOS 2.00 machines, this location marked the beginning of the operating system ROM’s.</td>
</tr>
<tr>
<td>0x00FF0000 – 0x00FF7FFF</td>
<td>N/A</td>
<td>N/A</td>
<td>Unassigned</td>
</tr>
</tbody>
</table>

**ROM/Reserved Hardware Space**

<table>
<thead>
<tr>
<th>Location(s)</th>
<th>Size</th>
<th>Type</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>0x00FA0000 – 0x00FBFFFF</td>
<td>BYTE</td>
<td>ROM</td>
<td>Cartridge ROM</td>
</tr>
<tr>
<td>0x00FC0000 – 0x00FEFFFF</td>
<td>BYTE</td>
<td>ROM</td>
<td>On pre TOS 2.00 machines, this location marked the beginning of the operating system ROM’s.</td>
</tr>
<tr>
<td>0x00FF0000 – 0x00FF7FFF</td>
<td>N/A</td>
<td>N/A</td>
<td>Unassigned</td>
</tr>
<tr>
<td>Location(s)</td>
<td>Size</td>
<td>ST</td>
<td>STe</td>
</tr>
<tr>
<td>-------------</td>
<td>------</td>
<td>----</td>
<td>-----</td>
</tr>
<tr>
<td>0x00FF8000</td>
<td>OB</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>0x00FF8002 – 0x00FF8004</td>
<td>N/A</td>
<td></td>
<td></td>
</tr>
<tr>
<td>0x00FF8006</td>
<td>BYTE</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>0x00FF8007</td>
<td>BYTE</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>0x00FF8008 – 0x00FF81FF</td>
<td>N/A</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>0x00FF8200</td>
<td>OB</td>
<td></td>
<td></td>
</tr>
<tr>
<td>0x00FF8202</td>
<td>OB</td>
<td></td>
<td></td>
</tr>
<tr>
<td>0x00FF8204</td>
<td>OB</td>
<td></td>
<td></td>
</tr>
<tr>
<td>0x00FF8206</td>
<td>OB</td>
<td></td>
<td></td>
</tr>
<tr>
<td>0x00FF8208</td>
<td>OB</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Location(s)</td>
<td>Size</td>
<td>Type</td>
<td>Meaning</td>
</tr>
<tr>
<td>-------------------</td>
<td>------</td>
<td>------</td>
<td>-------------------------------------------------------------------------</td>
</tr>
<tr>
<td>0x00FF820A</td>
<td>BYTE</td>
<td>I/O</td>
<td>Video Shifter Sync Mode as follows:</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>Bit 7</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>Bit 0</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>1= 60 Hz, 0 = 50 Hz</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>1 = External, 0 = Internal Sync</td>
</tr>
<tr>
<td>0x00FF820C</td>
<td>OB</td>
<td>I/O</td>
<td>Video Base Address Low</td>
</tr>
<tr>
<td>0x00FF820E</td>
<td>OB</td>
<td>I/O</td>
<td>Line Width Register (width of scanline in WORDs - 1). On a Falcon030, this is a WORD value.</td>
</tr>
<tr>
<td>0x00FF8210</td>
<td>WORD</td>
<td>I/O</td>
<td>Falcon030 Line Width Register (width of scanline in WORDs)</td>
</tr>
<tr>
<td>0x00FF8212 – 0x00FF823F</td>
<td>N/A</td>
<td>I/O</td>
<td>Unassigned</td>
</tr>
<tr>
<td>0x00FF8240</td>
<td>WORD</td>
<td>I/O</td>
<td>ST/e Compatible Palette Register #0: ST layout is as follows:</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>XXXX XRRR XGGG XBBB</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>STe layout is as follows:</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>XXXX RRRR GGGG BBBB</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>For compatibility, STe bit arrangement per nibble is 0-3-2-1. These registers are simulated for compatibility on newer model machines.</td>
</tr>
<tr>
<td>0x00FF8242</td>
<td>WORD</td>
<td>I/O</td>
<td>ST/e Compatible Palette Register #1</td>
</tr>
<tr>
<td>0x00FF8244</td>
<td>WORD</td>
<td>I/O</td>
<td>ST/e Compatible Palette Register #2</td>
</tr>
<tr>
<td>0x00FF8246</td>
<td>WORD</td>
<td>I/O</td>
<td>ST/e Compatible Palette Register #3</td>
</tr>
<tr>
<td>0x00FF8248</td>
<td>WORD</td>
<td>I/O</td>
<td>ST/e Compatible Palette Register #4</td>
</tr>
<tr>
<td>0x00FF824A</td>
<td>WORD</td>
<td>I/O</td>
<td>ST/e Compatible Palette Register #5</td>
</tr>
<tr>
<td>0x00FF824C</td>
<td>WORD</td>
<td>I/O</td>
<td>ST/e Compatible Palette Register #6</td>
</tr>
<tr>
<td>0x00FF824E</td>
<td>WORD</td>
<td>I/O</td>
<td>ST/e Compatible Palette Register #7</td>
</tr>
<tr>
<td>0x00FF8250</td>
<td>WORD</td>
<td>I/O</td>
<td>ST/e Compatible Palette Register #8</td>
</tr>
<tr>
<td>0x00FF8252</td>
<td>WORD</td>
<td>I/O</td>
<td>ST/e Compatible Palette Register #9</td>
</tr>
<tr>
<td>0x00FF8254</td>
<td>WORD</td>
<td>I/O</td>
<td>ST/e Compatible Palette Register #10</td>
</tr>
<tr>
<td>0x00FF8256</td>
<td>WORD</td>
<td>I/O</td>
<td>ST/e Compatible Palette Register #11</td>
</tr>
<tr>
<td>0x00FF8258</td>
<td>WORD</td>
<td>I/O</td>
<td>ST/e Compatible Palette Register #12</td>
</tr>
<tr>
<td>0x00FF825A</td>
<td>WORD</td>
<td>I/O</td>
<td>ST/e Compatible Palette Register #13</td>
</tr>
<tr>
<td>Location(s)</td>
<td>Size</td>
<td>Type</td>
<td>Meaning</td>
</tr>
<tr>
<td>--------------</td>
<td>-------</td>
<td>------</td>
<td>-------------------------------------------------------------------------</td>
</tr>
<tr>
<td>0x00FF825C</td>
<td>WORD</td>
<td>I/O</td>
<td>ST/e Compatible Palette Register #14</td>
</tr>
<tr>
<td>0x00FF825E</td>
<td>WORD</td>
<td>I/O</td>
<td>ST/e Compatible Palette Register #15</td>
</tr>
<tr>
<td>0x00FF8260</td>
<td>EB</td>
<td>I/O</td>
<td>ST Video Shifter Mode as follows:</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>Bit 7</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>Bit 0</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>00 = 320x200, 4 plane</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>01 = 640x200, 2 plane</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>10 = 640x400, 1 plane</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>11 = Reserved</td>
</tr>
<tr>
<td>0x00FF8262</td>
<td>EB</td>
<td>I/O</td>
<td>TT030 Video Shifter Mode as follows:</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>Smear Mode</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>Hyper Mono Mode</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>Bit 15</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>Bit 8</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>000 = 320x200, 4 plane</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>001 = 640x200, 2 plane</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>010 = 640x400, 1 plane</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>011 = 640x480, 4 plane</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>110 = 1280x960, 1 plane</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>111 = 320x480, 8 plane</td>
</tr>
<tr>
<td>0x00FF8264</td>
<td>OB</td>
<td>I/O</td>
<td>Horizontal Scroll Register</td>
</tr>
<tr>
<td>0x00FF8266</td>
<td>WORD</td>
<td>I/O</td>
<td>SPSHIFT Control Register as follows:</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>Bit</td>
</tr>
<tr>
<td>4</td>
<td>Enable Bitplane Mode</td>
<td></td>
<td></td>
</tr>
<tr>
<td>5</td>
<td>Use External VSYNC</td>
<td></td>
<td></td>
</tr>
<tr>
<td>6</td>
<td>Use External HSYNC</td>
<td></td>
<td></td>
</tr>
<tr>
<td>8</td>
<td>Enable Truecolor Mode</td>
<td></td>
<td></td>
</tr>
<tr>
<td>10</td>
<td>Enable 2-Color Mode</td>
<td></td>
<td></td>
</tr>
<tr>
<td>0x00FF8268 – 0x00FF827D</td>
<td>N/A</td>
<td>Unassigned</td>
<td></td>
</tr>
</tbody>
</table>
## B.22 – Memory Map

<table>
<thead>
<tr>
<th>Location(s)</th>
<th>Size</th>
<th>Type</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>0x00FF827E</td>
<td>EB</td>
<td>I/O</td>
<td>STACY Display State as follows:</td>
</tr>
<tr>
<td>0x00FF8280</td>
<td>WORD</td>
<td>I/O</td>
<td>Horizontal Hold Counter</td>
</tr>
<tr>
<td>0x00FF8282</td>
<td>WORD</td>
<td>I/O</td>
<td>Horizontal Hold Timer</td>
</tr>
<tr>
<td>0x00FF8284</td>
<td>WORD</td>
<td>I/O</td>
<td>Horizontal Border Begin</td>
</tr>
<tr>
<td>0x00FF8286</td>
<td>WORD</td>
<td>I/O</td>
<td>Horizontal Border End</td>
</tr>
<tr>
<td>0x00FF8288</td>
<td>WORD</td>
<td>I/O</td>
<td>Horizontal Display Begin</td>
</tr>
<tr>
<td>0x00FF828A</td>
<td>WORD</td>
<td>I/O</td>
<td>Horizontal Display End</td>
</tr>
<tr>
<td>0x00FF828C</td>
<td>WORD</td>
<td>I/O</td>
<td>HSS</td>
</tr>
<tr>
<td>0x00FF828E</td>
<td>WORD</td>
<td>I/O</td>
<td>HFS</td>
</tr>
<tr>
<td>0x00FF8290</td>
<td>WORD</td>
<td>I/O</td>
<td>HEE</td>
</tr>
<tr>
<td>0x00FF8292</td>
<td>N/A</td>
<td></td>
<td>Unassigned</td>
</tr>
<tr>
<td>0x00FF8299</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>0x00FF829A</td>
<td>WORD</td>
<td>I/O</td>
<td>Vertical Frequency Counter</td>
</tr>
<tr>
<td>0x00FF829C</td>
<td>WORD</td>
<td>I/O</td>
<td>Vertical Frequency Timer</td>
</tr>
<tr>
<td>0x00FF829E</td>
<td>WORD</td>
<td>I/O</td>
<td>Vertical Border Begin</td>
</tr>
<tr>
<td>0x00FF82A6</td>
<td>WORD</td>
<td>I/O</td>
<td>Vertical Border End (in half lines)</td>
</tr>
<tr>
<td>0x00FF82A8</td>
<td>WORD</td>
<td>I/O</td>
<td>Vertical Display Begin</td>
</tr>
<tr>
<td>0x00FF82AA</td>
<td>WORD</td>
<td>I/O</td>
<td>Vertical Display End</td>
</tr>
<tr>
<td>0x00FF82AC</td>
<td>WORD</td>
<td>I/O</td>
<td>VSS</td>
</tr>
<tr>
<td>0x00FF82AE</td>
<td>N/A</td>
<td></td>
<td>Unassigned</td>
</tr>
<tr>
<td>0x00FF82C1</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>0x00FF82C2</td>
<td>WORD</td>
<td>I/O</td>
<td>VCO - Video Control as follows:</td>
</tr>
</tbody>
</table>

### Bit 7 Bit 0

- Bit 7: Backlight Off
- Bit 0: Display Off

### Bit 3 Bit 0

- Bit 3: Quarter Pixel Width
- Bit 2: Halve Pixel Width
- Bit 1: Interlace Mode
- Bit 0: Line Doubling
<table>
<thead>
<tr>
<th>Location(s)</th>
<th>Size</th>
<th>Type</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>0x00FF82C4 – 0x00FF83FF</td>
<td>N/A</td>
<td>I/O</td>
<td>Unassigned</td>
</tr>
<tr>
<td>0x00FF8400 – 0x00FF85FE</td>
<td>WORD</td>
<td>I/O</td>
<td>TT030 Palette Registers #0 – #255: Each palette register is a longword which is arranged as follows: XXXX RRRR GGGG BBBB. Unlike the ST registers, each nibble is properly formatted in the manner 3–2–1–0.</td>
</tr>
<tr>
<td>0x00FF8600 – 0x00FF8602</td>
<td>WORD</td>
<td>I/O</td>
<td>Reserved</td>
</tr>
<tr>
<td>0x00FF8604</td>
<td>WORD</td>
<td>I/O</td>
<td>DMA Sector Count (on write)_DMA Data Register (on read)</td>
</tr>
<tr>
<td>0x00FF8606</td>
<td>WORD</td>
<td>I/O</td>
<td>DMA Status (on read) as follows:</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>Bit 2 Bit 0 Data Request Inactive Block Count Zero ERROR</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>DMA Mode Control (on write) as follows:</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>Bit 8 Select Block Count Register Bit 0</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>Destination Select (_DRQ) 0 = Floppy, 1 = ACSI</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>Destination Select (_CS) 0 = Floppy, 1 = ACSI</td>
</tr>
<tr>
<td>0x00FF8608</td>
<td>OB</td>
<td>I/O</td>
<td>DMA Pointer High</td>
</tr>
<tr>
<td>0x00FF860A</td>
<td>OB</td>
<td>I/O</td>
<td>DMA Pointer Mid</td>
</tr>
<tr>
<td>0x00FF860C</td>
<td>OB</td>
<td>I/O</td>
<td>DMA Pointer Low</td>
</tr>
<tr>
<td>0x00FF860E – 0x00FF86FF</td>
<td>N/A</td>
<td>I/O</td>
<td>Unassigned</td>
</tr>
</tbody>
</table>
### B.24 – Memory Map

<table>
<thead>
<tr>
<th>Location(s)</th>
<th>Size</th>
<th>Type</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>0x00FF8700</td>
<td>OB</td>
<td>I/O</td>
<td>SCSI DMA Control Register as follows:</td>
</tr>
<tr>
<td>0x00FF8702</td>
<td>OB</td>
<td>I/O</td>
<td>I/O SCSI DMA Pointer Upper-Middle</td>
</tr>
<tr>
<td>0x00FF8704</td>
<td>OB</td>
<td>I/O</td>
<td>I/O SCSI DMA Pointer Lower-Middle</td>
</tr>
<tr>
<td>0x00FF8706</td>
<td>OB</td>
<td>I/O</td>
<td>I/O SAS DMA Pointer Lower</td>
</tr>
<tr>
<td>0x00FF8708</td>
<td>OB</td>
<td>I/O</td>
<td>I/O Byte Count Upper</td>
</tr>
<tr>
<td>0x00FF870A</td>
<td>OB</td>
<td>I/O</td>
<td>I/O Byte Count Upper-Middle</td>
</tr>
<tr>
<td>0x00FF870C</td>
<td>OB</td>
<td>I/O</td>
<td>I/O Byte Count Lower-Middle</td>
</tr>
<tr>
<td>0x00FF870E</td>
<td>OB</td>
<td>I/O</td>
<td>I/O Byte Count Lower</td>
</tr>
<tr>
<td>0x00FF8710</td>
<td>WORD</td>
<td>I/O</td>
<td>I/O SCSI DMA Data Residue Register High</td>
</tr>
<tr>
<td>0x00FF8712</td>
<td>WORD</td>
<td>I/O</td>
<td>I/O SCSI DMA Data Residue Register Low</td>
</tr>
<tr>
<td>0x00FF8714</td>
<td>OB</td>
<td>I/O</td>
<td>SCSI DMA Control Register as follows:</td>
</tr>
<tr>
<td>0x00FF8716 – 0x00FF871F</td>
<td>N/A</td>
<td>I/O</td>
<td>Unassigned</td>
</tr>
<tr>
<td>0x00FF8780</td>
<td>OB</td>
<td>I/O</td>
<td>SCSI Controller Data Register</td>
</tr>
<tr>
<td>0x00FF8782</td>
<td>OB</td>
<td>I/O</td>
<td>SCSI Controller Initiator Command Register</td>
</tr>
<tr>
<td>0x00FF8784</td>
<td>OB</td>
<td>I/O</td>
<td>SCSI Controller Mode Register</td>
</tr>
<tr>
<td>0x00FF8786</td>
<td>OB</td>
<td>I/O</td>
<td>SCSI Controller Target Command Register</td>
</tr>
<tr>
<td>0x00FF8788</td>
<td>OB</td>
<td>I/O</td>
<td>SCSI Controller ID Select/Control Register</td>
</tr>
<tr>
<td>0x00FF878A</td>
<td>OB</td>
<td>I/O</td>
<td>SCSI Controller DMA Start/DMA Status</td>
</tr>
<tr>
<td>0x00FF878C</td>
<td>OB</td>
<td>I/O</td>
<td>SCSI Controller DMA Target Receive/Input Data</td>
</tr>
<tr>
<td>0x00FF878E</td>
<td>OB</td>
<td>I/O</td>
<td>SCSI Controller DMA Initiator Receive/Reset</td>
</tr>
<tr>
<td>0x00FF8790 – 0x00FF879F</td>
<td>N/A</td>
<td>I/O</td>
<td>Unassigned</td>
</tr>
</tbody>
</table>

**SCSI DMA Control**

- **0x00FF8700**: SCSI DMA Pointer Upper
- **0x00FF8702**: SCSI DMA Pointer Upper-Middle
- **0x00FF8704**: SCSI DMA Pointer Lower-Middle
- **0x00FF8706**: SCSI DMA Pointer Lower
- **0x00FF8708**: Byte Count Upper
- **0x00FF870A**: Byte Count Upper-Middle
- **0x00FF870C**: Byte Count Lower-Middle
- **0x00FF870E**: Byte Count Lower
- **0x00FF8710**: SCSI DMA Data Residue Register High
- **0x00FF8712**: SCSI DMA Data Residue Register Low
- **0x00FF8714**: SCSI DMA Control Register

**SCSI Controller Registers**

- **0x00FF8780**: SCSI Controller Data Register
- **0x00FF8782**: SCSI Controller Initiator Command Register
- **0x00FF8784**: SCSI Controller Mode Register
- **0x00FF8786**: SCSI Controller Target Command Register
- **0x00FF8788**: SCSI Controller ID Select/Control Register
- **0x00FF878A**: SCSI Controller DMA Start/DMA Status
- **0x00FF878C**: SCSI Controller DMA Target Receive/Input Data
- **0x00FF878E**: SCSI Controller DMA Initiator Receive/Reset

---

**THE ATARI COMPENDIUM**
<table>
<thead>
<tr>
<th>Location(s)</th>
<th>Size</th>
<th>Type</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>0x00FF8800</td>
<td>EB</td>
<td>I/O</td>
<td>PSG Read (Read only on I/O port B) / PSG Register Select (WO). Reading this location yields data from the parallel interface. Writing to bits 0–3 of this location selects a PSG register to address as follows:</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>Value Register</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>0000 Channel A Fine Tune</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>0001 Channel A Coarse Tune</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>0010 Channel B Fine Tune</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>0011 Channel B Coarse Tune</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>0100 Channel C Fine Tune</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>0101 Channel C Coarse Tune</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>0110 Noise Generator Control</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>0111 Mixer Control – I/O Enable</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>1000 Channel A Amplitude</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>1001 Channel B Amplitude</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>1010 Channel C Amplitude</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>1011 Envelope Period Fine Tune</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>1100 Envelope Period Coarse Tune</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>1110 I/O Port A Select (Write only)</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>1111 I/O Port B Select</td>
</tr>
<tr>
<td>0x00FF8802</td>
<td>EB</td>
<td>I/O</td>
<td>When I/O Port A is selected, this location contains the PSG Write Data (WO) register as follows:</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>Bit 7</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>Bit 0</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>Falcon = IDE Drive On/Off</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>TT = SCC A (0 = LAN, 1 = Serial2)</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>Falcon = Internal Speaker On/Off</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>Others = Monitor Jack GPO Pin</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>Centronics _STROBE</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>RS232 Data Terminal Ready</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>RS232 Request to Send</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>Floppy _Drive0 Select</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>Floppy _Drive1 Select</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>Floppy _Side0/1 Select</td>
</tr>
<tr>
<td>0x00FF8804 – 0x00FF88FF</td>
<td>N/A</td>
<td>I/O</td>
<td>Unassigned</td>
</tr>
</tbody>
</table>

**THE ATARI COMPENDIUM**
### DMA Sound System

<table>
<thead>
<tr>
<th>Location(s)</th>
<th>Size</th>
<th>Type</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>0x00FF8900</td>
<td>BYTE</td>
<td>I/O</td>
<td>Sound DMA Control as follows:</td>
</tr>
<tr>
<td>0x00FF8901</td>
<td>BYTE</td>
<td>I/O</td>
<td>Additional sound DMA control as follows:</td>
</tr>
<tr>
<td>0x00FF8902</td>
<td>OB</td>
<td>I/O</td>
<td>Frame Base Address High</td>
</tr>
<tr>
<td>0x00FF8904</td>
<td>OB</td>
<td>I/O</td>
<td>Frame Base Address Mid</td>
</tr>
<tr>
<td>0x00FF8906</td>
<td>OB</td>
<td>I/O</td>
<td>Frame Base Address Low</td>
</tr>
<tr>
<td>0x00FF8908</td>
<td>OB</td>
<td>I/O</td>
<td>Frame Address Counter High</td>
</tr>
<tr>
<td>0x00FF890A</td>
<td>OB</td>
<td>I/O</td>
<td>Frame Address Counter Mid</td>
</tr>
<tr>
<td>0x00FF890C</td>
<td>OB</td>
<td>I/O</td>
<td>Frame Address Counter Low</td>
</tr>
<tr>
<td>0x00FF890E</td>
<td>OB</td>
<td>I/O</td>
<td>Frame End Address High</td>
</tr>
<tr>
<td>0x00FF8910</td>
<td>OB</td>
<td>I/O</td>
<td>Frame End Address Mid</td>
</tr>
<tr>
<td>0x00FF8912</td>
<td>OB</td>
<td>I/O</td>
<td>Frame End Address Low</td>
</tr>
<tr>
<td>0x00FF8914 – 0x00FF8919</td>
<td>N/A</td>
<td>I/O</td>
<td>Unassigned</td>
</tr>
<tr>
<td>Location(s)</td>
<td>Size</td>
<td>Type</td>
<td>Meaning</td>
</tr>
<tr>
<td>-------------</td>
<td>--------</td>
<td>------</td>
<td>----------------------------------------------</td>
</tr>
<tr>
<td>0x00FF8920</td>
<td>BYTE</td>
<td>I/O</td>
<td>Sound mode control as follows:</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>Bit 0</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>Bit 7</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>00 = Monitor Track 0</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>01 = Monitor Track 1</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>10 = Monitor Track 2</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>11 = Monitor Track 3</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>Bit 7</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>00 = Play 1 Track</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>01 = Play 2 Tracks</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>10 = Play 3 Tracks</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>11 = Play 4 Tracks</td>
</tr>
<tr>
<td>0x00FF8921</td>
<td>BYTE</td>
<td>I/O</td>
<td>Additional sound mode control as follows:</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>Bit 0</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>Bit 7</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>00 = 8-bit Stereo</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>01 = 16-bit Stereo (Falcon)</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>10 = 8-bit Mono</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>Bit 7</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>00 = 6258 Hz</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>01 = 12517 Hz</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>10 = 25033 Hz</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>11 = 50066 Hz</td>
</tr>
</tbody>
</table>

**MICROWIRE**

<table>
<thead>
<tr>
<th>Location(s)</th>
<th>Size</th>
<th>Type</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>0x00FF8922</td>
<td>WORD</td>
<td>I/O</td>
<td>MICROWIRE Data Register</td>
</tr>
<tr>
<td>0x00FF8924</td>
<td>WORD</td>
<td>I/O</td>
<td>MICROWIRE Mask Register</td>
</tr>
<tr>
<td>0x00FF8926 - 0x00FF8929</td>
<td>N/A</td>
<td>I/O</td>
<td>Unassigned</td>
</tr>
</tbody>
</table>
### B.28 – Memory Map

<table>
<thead>
<tr>
<th>Location(s)</th>
<th>Size</th>
<th>Type</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>0x00FF8930</td>
<td>WORD</td>
<td>I/O</td>
<td>Falcon030 DSP/DMA Controller as follows:</td>
</tr>
</tbody>
</table>

- Bit 0
  - 00 = 25.175 MHz Clock
  - 01 = External Clock
  - 10 = 32 MHz Clock
  - 0 = Handshake Enable

- Bit 7
  - 0 = Internal Sync
  - 1 = External Sync

- Bit 8
  - 0 = DMA In, 1 = All

- Bit 12
  - 0 = 25.175 MHz Clock
  - 01 = External Clock
  - 10 = 32 MHz Clock
  - 0 = Enable Handshake

---

**Falcon030 DSP/DMA Controller**

<table>
<thead>
<tr>
<th>Location(s)</th>
<th>Size</th>
<th>Type</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>0x00FF8930</td>
<td>WORD</td>
<td>I/O</td>
<td>Falcon030 DSP/DMA Controller as follows:</td>
</tr>
</tbody>
</table>

- Bit 0
  - 00 = 25.175 MHz Clock
  - 01 = External Clock
  - 10 = 32 MHz Clock
  - 0 = Handshake Enable

- Bit 7
  - 0 = Internal Sync
  - 1 = External Sync

- Bit 8
  - 0 = DMA In, 1 = All

- Bit 12
  - 0 = 25.175 MHz Clock
  - 01 = External Clock
  - 10 = 32 MHz Clock
  - 0 = Enable Handshake

---

**THE ATARI COMPENDIUM**
Falcon030 DSP/DMA Controller – B.29

<table>
<thead>
<tr>
<th>Location(s)</th>
<th>Size</th>
<th>Type</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>0x00FF8932</td>
<td>WORD</td>
<td>I/O</td>
<td>DMA Crossbar Input Select Controller as follows:</td>
</tr>
</tbody>
</table>

- **Bit 0**
  - 00 = DMA Output
  - 01 = DSP Output
  - 10 = External Input
  - 11 = ADC Input

- **Bit 7**
  - 0 = Handshake Enable

- **Bit 8**
  - 00 = DMA Output
  - 01 = DSP Output
  - 10 = External Input
  - 11 = ADC Input

- **Bit 12**
  - 00 = DMA Output
  - 01 = DSP Output
  - 10 = External Input
  - 11 = ADC Input

- **Bit 13**
  - 0 = Enable Handshake

---

The Atari Compendium
<table>
<thead>
<tr>
<th>Location(s)</th>
<th>Size</th>
<th>Type</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>0x00FF8934</td>
<td>BYTE</td>
<td>I/O</td>
<td>Frequency Divider External Sync (0 = STe/TT030 Compatible Prescaler, 1-15 = Divide by 256 and then the value given)</td>
</tr>
<tr>
<td>0x00FF8935</td>
<td>BYTE</td>
<td>I/O</td>
<td>Frequency Divider Internal Sync as follows:</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td><strong>Value</strong></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>0</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>1</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>2</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>3</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>4</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>5</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>6</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>7</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>8</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>9</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>10</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>11</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>12</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>13</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>14</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>15</td>
</tr>
<tr>
<td>0x00FF8936</td>
<td>BYTE</td>
<td>I/O</td>
<td>Record Tracks Select as follows:</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>Bit 1/0</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>00 = Record 1 Track</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>01 = Record 2 Tracks</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>10 = Record 3 Tracks</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>11 = Record 4 Tracks</td>
</tr>
<tr>
<td>0x00FF8937</td>
<td>BYTE</td>
<td>I/O</td>
<td>CODEC Input Source as follows:</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>Bit 1/0</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>Multiplexer</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>ADC/DAC</td>
</tr>
</tbody>
</table>
### Real Time Clock (146818A) – B.31

<table>
<thead>
<tr>
<th>Location(s)</th>
<th>Size</th>
<th>Type</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>0x00FF8938</td>
<td>BYTE</td>
<td>I/O</td>
<td>CODEC ADC Input as follows:</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>Bit 1/0</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>0 = Left Channel Mic</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>1 = Left Channel PSG</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>0 = Right Channel Mic</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>1 = Right Channel PSG</td>
</tr>
<tr>
<td>0x00FF8939</td>
<td>BYTE</td>
<td>I/O</td>
<td>Gain settings (0–15 per channel) as follows:</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>Bit 7 Bit 0</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>L L L L L R R R R</td>
</tr>
<tr>
<td>0x00FF893A</td>
<td>BYTE</td>
<td>I/O</td>
<td>Attenuation settings (0–15 per channel) as follows:</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>Bit 7 Bit 0</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>L L L L L R R R R</td>
</tr>
<tr>
<td>0x00FF8940</td>
<td>OB</td>
<td>I/O</td>
<td>GPIO Data direction as follows:</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>Bit 2 Bit 0</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>0 = Read</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>1 = Write</td>
</tr>
<tr>
<td>0x00FF8942</td>
<td>OB</td>
<td>I/O</td>
<td>GPIO Data (low three bits). Read or write by setting direction bits above.</td>
</tr>
<tr>
<td>0x00FF8944 – 0x00FF895F</td>
<td>N/A</td>
<td>I/O</td>
<td>Unassigned</td>
</tr>
</tbody>
</table>

**Real Time Clock (146818A)**

<table>
<thead>
<tr>
<th>Location(s)</th>
<th>Size</th>
<th>Type</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>0x00FF8960</td>
<td>OB</td>
<td>I/O</td>
<td>Real Time Clock Address Register</td>
</tr>
<tr>
<td>0x00FF8962</td>
<td>OB</td>
<td>I/O</td>
<td>Real Time Clock Data Register</td>
</tr>
<tr>
<td>0x00FF8964 – 0x00FF89FF</td>
<td>N/A</td>
<td>I/O</td>
<td>Unassigned</td>
</tr>
</tbody>
</table>

---

**The Atari Compendium**
### B.32 – Memory Map

<table>
<thead>
<tr>
<th>Location(s)</th>
<th>Size</th>
<th>Type</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>0x00FF8A00 – 0x00FF8A1E</td>
<td>WORD</td>
<td>I/O</td>
<td>BLITTER Bit-Block Transfer Processor</td>
</tr>
<tr>
<td>0x00FF8A20</td>
<td>WORD</td>
<td>I/O</td>
<td>BLITTER Source X Increment</td>
</tr>
<tr>
<td>0x00FF8A22</td>
<td>WORD</td>
<td>I/O</td>
<td>BLITTER Source Y Increment</td>
</tr>
<tr>
<td>0x00FF8A24</td>
<td>WORD</td>
<td>I/O</td>
<td>BLITTER Source Address (bits 7–0 are bits 23–16 of address)</td>
</tr>
<tr>
<td>0x00FF8A26</td>
<td>WORD</td>
<td>I/O</td>
<td>BLITTER Source Address (bits 15–1 are bits 15–1 of address, bit 0 must be 0)</td>
</tr>
<tr>
<td>0x00FF8A28</td>
<td>WORD</td>
<td>I/O</td>
<td>BLITTER Endmask 1</td>
</tr>
<tr>
<td>0x00FF8A2A</td>
<td>WORD</td>
<td>I/O</td>
<td>BLITTER Endmask 2</td>
</tr>
<tr>
<td>0x00FF8A2C</td>
<td>WORD</td>
<td>I/O</td>
<td>BLITTER Endmask 3</td>
</tr>
<tr>
<td>0x00FF8A2E</td>
<td>WORD</td>
<td>I/O</td>
<td>BLITTER Destination X Increment</td>
</tr>
<tr>
<td>0x00FF8A30</td>
<td>WORD</td>
<td>I/O</td>
<td>BLITTER Destination Y Increment</td>
</tr>
<tr>
<td>0x00FF8A32</td>
<td>WORD</td>
<td>I/O</td>
<td>BLITTER Destination (bits 7–0 are bits 23–16 of address)</td>
</tr>
<tr>
<td>0x00FF8A34</td>
<td>WORD</td>
<td>I/O</td>
<td>BLITTER Destination (bits 15–1 are bits 15–1 of address, bit 0 must be 0)</td>
</tr>
<tr>
<td>0x00FF8A36</td>
<td>WORD</td>
<td>I/O</td>
<td>BLITTER X Count</td>
</tr>
<tr>
<td>0x00FF8A38</td>
<td>WORD</td>
<td>I/O</td>
<td>BLITTER Y Count</td>
</tr>
<tr>
<td>0x00FF8A3A</td>
<td>BYTE</td>
<td>I/O</td>
<td>BLITTER HOP</td>
</tr>
<tr>
<td>0x00FF8A3B</td>
<td>BYTE</td>
<td>I/O</td>
<td>BLITTER Configuration as follows:</td>
</tr>
<tr>
<td>0x00FF8A3C</td>
<td>BYTE</td>
<td>I/O</td>
<td>BUSY</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>HOG</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>SMUDGE</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>Bit 0</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>LINE NUMBER</td>
</tr>
</tbody>
</table>

**THE ATARI COMPENDIUM**
### SCC DMA Registers – B.33

<table>
<thead>
<tr>
<th>Location(s)</th>
<th>Size</th>
<th>Type</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>0x00FF8A3D</td>
<td>BYTE</td>
<td>I/O</td>
<td>BLITTER Configuration as follows:</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td><img src="image.png" alt="Diagram" /></td>
</tr>
<tr>
<td>0x00FF8A3E–0x00FF8BFF</td>
<td>N/A</td>
<td>I/O</td>
<td>Unassigned</td>
</tr>
<tr>
<td>0x00FF8C00</td>
<td>OB</td>
<td>I/O</td>
<td>SCC DMA Pointer Upper</td>
</tr>
<tr>
<td>0x00FF8C02</td>
<td>OB</td>
<td>I/O</td>
<td>SCC DMA Pointer Upper-Middle</td>
</tr>
<tr>
<td>0x00FF8C04</td>
<td>OB</td>
<td>I/O</td>
<td>SCC DMA Pointer Lower-Middle</td>
</tr>
<tr>
<td>0x00FF8C06</td>
<td>OB</td>
<td>I/O</td>
<td>SCC DMA Pointer Lower</td>
</tr>
<tr>
<td>0x00FF8C08</td>
<td>OB</td>
<td>I/O</td>
<td>SCC Byte Count Upper</td>
</tr>
<tr>
<td>0x00FF8C0A</td>
<td>OB</td>
<td>I/O</td>
<td>SCC Byte Count Upper-Middle</td>
</tr>
<tr>
<td>0x00FF8C0C</td>
<td>OB</td>
<td>I/O</td>
<td>SCC Byte Count Lower-Middle</td>
</tr>
<tr>
<td>0x00FF8C0E</td>
<td>OB</td>
<td>I/O</td>
<td>SCC Byte Count Lower</td>
</tr>
<tr>
<td>0x00FF8C10</td>
<td>WORD</td>
<td>I/O</td>
<td>SCC Data Residue Register High (RO)</td>
</tr>
<tr>
<td>0x00FF8C12</td>
<td>WORD</td>
<td>I/O</td>
<td>SCC Data Residue Register Low (RO)</td>
</tr>
<tr>
<td>0x00FF8C14</td>
<td>OB</td>
<td>I/O</td>
<td>SCC DMA Control Register as follows:</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td><img src="image.png" alt="Diagram" /></td>
</tr>
<tr>
<td>0x00FF8C16–0x00FF8C7E</td>
<td>N/A</td>
<td>I/O</td>
<td>Unassigned</td>
</tr>
</tbody>
</table>

---

**THE ATARI COMPENDIUM**
## B.34 – Memory Map

<table>
<thead>
<tr>
<th>Location(s)</th>
<th>Size</th>
<th>Type</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

### SCC Ports (85C30)

<table>
<thead>
<tr>
<th>Address</th>
<th>Type</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>0x00FF8C80</td>
<td>OB</td>
<td>I/O SCC A Control</td>
</tr>
<tr>
<td>0x00FF8C82</td>
<td>OB</td>
<td>I/O SCC A Data</td>
</tr>
<tr>
<td>0x00FF8C84</td>
<td>OB</td>
<td>I/O SCC B Control</td>
</tr>
<tr>
<td>0x00FF8C86</td>
<td>OB</td>
<td>I/O SCC B Data</td>
</tr>
<tr>
<td>0x00FF8C88 - 0x00FF8DFF</td>
<td>N/A</td>
<td>I/O Unassigned</td>
</tr>
</tbody>
</table>

### System Control Unit

<table>
<thead>
<tr>
<th>Address</th>
<th>Type</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>0x00FF8E00</td>
<td>OB</td>
<td>I/O SCU System Interrupt Mask</td>
</tr>
<tr>
<td>0x00FF8E02</td>
<td>OB</td>
<td>I/O SCU System Interrupt State (RO)</td>
</tr>
<tr>
<td>0x00FF8E04</td>
<td>OB</td>
<td>I/O SCU System Interrupter: Set Bit #0 to generate VME interrupt IRQ1.</td>
</tr>
<tr>
<td>0x00FF8E06</td>
<td>OB</td>
<td>I/O VME Interrupter: Set Bit #0 to generate VME interrupt IRQ3.</td>
</tr>
<tr>
<td>0x00FF8E08</td>
<td>OB</td>
<td>I/O SCU General Purpose Register 1</td>
</tr>
<tr>
<td>0x00FF8E0A</td>
<td>OB</td>
<td>I/O SCU General Purpose Register 2</td>
</tr>
<tr>
<td>0x00FF8E0C</td>
<td>OB</td>
<td>I/O VME Interrupt Mask</td>
</tr>
<tr>
<td>0x00FF8E0E</td>
<td>OB</td>
<td>I/O VME Interrupt State (RO)</td>
</tr>
<tr>
<td>0x00FF8E10 - 0x00FF8E1F</td>
<td>N/A</td>
<td>Unassigned</td>
</tr>
</tbody>
</table>

### Mega STe Cache/Processor Control

<table>
<thead>
<tr>
<th>Address</th>
<th>Type</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>0x00FF8E20</td>
<td>OB</td>
<td>I/O Mega STe Cache/Processor Control Register as follows:</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Value</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>0xFF</td>
<td>16 MHz w/Cache</td>
</tr>
<tr>
<td>0xFE</td>
<td>16 MHz</td>
</tr>
<tr>
<td>0xF4</td>
<td>8 MHz</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Address</th>
<th>Type</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>0x00FF8E22 - 0x00FF8EFF</td>
<td>N/A</td>
<td>Unassigned</td>
</tr>
</tbody>
</table>

### Extended Joystick/Paddle/Light Gun Ports

<table>
<thead>
<tr>
<th>Address</th>
<th>Type</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>0x00FF9200</td>
<td>WORD</td>
<td>I/O Joystick Fire Button Matrix Register</td>
</tr>
<tr>
<td>0x00FF9202</td>
<td>WORD</td>
<td>I/O Joystick Direction Matrix Register</td>
</tr>
<tr>
<td>0x00FF9204 - 0x00FF920F</td>
<td>N/A</td>
<td>I/O Unassigned</td>
</tr>
<tr>
<td>0x00FF9210</td>
<td>WORD</td>
<td>I/O Paddle 0 X Direction</td>
</tr>
<tr>
<td>0x00FF9212</td>
<td>WORD</td>
<td>I/O Paddle 0 Y Direction</td>
</tr>
<tr>
<td>0x00FF9214</td>
<td>WORD</td>
<td>I/O Paddle 1 X Direction</td>
</tr>
<tr>
<td>0x00FF9216</td>
<td>WORD</td>
<td>I/O Paddle 1 Y Direction</td>
</tr>
<tr>
<td>Location(s)</td>
<td>Size</td>
<td>Type</td>
</tr>
<tr>
<td>------------------------</td>
<td>--------</td>
<td>-------</td>
</tr>
<tr>
<td>0x00FF9218 – 0x00FF921F</td>
<td>N/A</td>
<td>I/O</td>
</tr>
<tr>
<td>0x00FF9220</td>
<td>WORD</td>
<td>I/O</td>
</tr>
<tr>
<td>0x00FF9222</td>
<td>WORD</td>
<td>I/O</td>
</tr>
<tr>
<td>0x00FF9224 – 0x00FF97FF</td>
<td>N/A</td>
<td>I/O</td>
</tr>
</tbody>
</table>

**Falcon030 VIDEL Palette Registers**

<table>
<thead>
<tr>
<th>Location(s)</th>
<th>Size</th>
<th>Type</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>0x00FF9800 – 0x00FF9BFC</td>
<td>LONG</td>
<td>I/O</td>
<td>Falcon030 Palette Registers 0-255 as follows:</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>RRRRRR-- GGGGGG-- ------------ BBBBBB--</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Location(s)</th>
<th>Size</th>
<th>Type</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>0x00FF9C00 – 0x00FFA1FF</td>
<td>N/A</td>
<td>I/O</td>
<td>Unassigned</td>
</tr>
</tbody>
</table>
### DSP Host Interface

<table>
<thead>
<tr>
<th>Location(s)</th>
<th>Size</th>
<th>Type</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>0x00FFA200</td>
<td>BYTE</td>
<td>I/O</td>
<td>Interrupt Control Register (DSP X:$FFE9) as follows:</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td><strong>Bit #7</strong></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>INIT – Setting this bit forces initialization of the host interface.</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td><strong>Bits #6–5</strong></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>DMA Mode Control as follows:</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td><strong>Value</strong></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>%00</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>%01</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>%10</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>%11</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td><strong>Bit #4–3</strong></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>Host Flags 1 &amp; 0 respectively (HF1 &amp; HF0)</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td><strong>Bit #2</strong></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>Unused</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td><strong>Bits #1–0</strong></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>Data Transfer Mode as follows:</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td><strong>Value</strong></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>%00</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>%01</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>%10</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>%11</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td><strong>Value</strong></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>%00</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>%01</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>%10</td>
</tr>
<tr>
<td>Location(s)</td>
<td>Size</td>
<td>Type</td>
<td>Meaning</td>
</tr>
<tr>
<td>-------------</td>
<td>----------</td>
<td>----------</td>
<td>-------------------------------------------------------------------------</td>
</tr>
<tr>
<td>0x00FFA201</td>
<td>BYTE</td>
<td>I/O</td>
<td>Command Vector Register (DSP X:$FFE9) as follows:</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>Host Command Bit</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>Bit 0</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>Bit 7</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>Host Vector (0-31)</td>
</tr>
<tr>
<td>0x00FFA202</td>
<td>BYTE</td>
<td>I/O</td>
<td>Interrupt Status Register (DSP X:$FFE8) as follows:</td>
</tr>
<tr>
<td>0x00FFA203</td>
<td>BYTE</td>
<td>I/O</td>
<td>Interrupt Vector Register (This register contains the 680x0 exception vector used for DSP exceptions).</td>
</tr>
<tr>
<td>0x00FFA204</td>
<td>BYTE</td>
<td>I/O</td>
<td>Unused</td>
</tr>
<tr>
<td>0x00FFA205</td>
<td>BYTE</td>
<td>I/O</td>
<td>DSP WORD High (DSP X:$FFEB)</td>
</tr>
<tr>
<td>0x00FFA206</td>
<td>BYTE</td>
<td>I/O</td>
<td>DSP WORD Middle (DSP X:$FFEB)</td>
</tr>
<tr>
<td>0x00FFA207</td>
<td>BYTE</td>
<td>I/O</td>
<td>DSP WORD Low (DSP X:$FFEB)</td>
</tr>
<tr>
<td>0x00FFA208–</td>
<td>N/A</td>
<td>N/A</td>
<td>Undefined</td>
</tr>
<tr>
<td>0x00FFFF9FF</td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

### ST Multi-Function Peripheral Port (68901) – B.37

<table>
<thead>
<tr>
<th>Location(s)</th>
<th>Size</th>
<th>Type</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>0x00FFFA00</td>
<td>OB</td>
<td>I/O</td>
<td>MFP-ST General Purpose Pins (Parallel port data register on Atari machines).</td>
</tr>
<tr>
<td>0x00FFFA02</td>
<td>OB</td>
<td>I/O</td>
<td>MFP-ST Active Edge Register as follows:</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>Monochrome Monitor Detect</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>RS-232 Ring Indicator</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>FDC/HDC Interrupt</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>Keyboard/MIDI Interrupt</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>Bit 0</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>Bit 7</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>Unused</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>RS-232 Clear To Send</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>RS-232 Carrier Detect</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>Centronics Busy</td>
</tr>
</tbody>
</table>

On a Falcon030, the MFP is not actually used for serial communications.

<table>
<thead>
<tr>
<th>Location(s)</th>
<th>Size</th>
<th>Type</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>0x00FFFA04</td>
<td>OB</td>
<td>I/O</td>
<td>MFP-ST Data Direction Register. Each bit is individually programmed (0 = input, 1 = output).</td>
</tr>
</tbody>
</table>
## B.38 – Memory Map

<table>
<thead>
<tr>
<th>Location(s)</th>
<th>Size</th>
<th>Type</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>0x00FFFA06</td>
<td>OB</td>
<td>I/O</td>
<td>MFP-ST Interrupt Enable Register A as follows:</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>![Diagram of Interrupt Enable Register A]</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>On a Falcon030, the MFP is not actually used for serial communications.</td>
</tr>
<tr>
<td>0x00FFFA08</td>
<td>OB</td>
<td>I/O</td>
<td>MFP-ST Interrupt Enable Register B as follows:</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>![Diagram of Interrupt Enable Register B]</td>
</tr>
<tr>
<td>0x00FFFA0A</td>
<td>OB</td>
<td>I/O</td>
<td>MFP-ST Interrupt Pending Register A (see mapping at 0x00FFFA06).</td>
</tr>
<tr>
<td>0x00FFFA0C</td>
<td>OB</td>
<td>I/O</td>
<td>MFP-ST Interrupt Pending Register B (see mapping at 0x00FFFA08).</td>
</tr>
<tr>
<td>0x00FFFA0E</td>
<td>OB</td>
<td>I/O</td>
<td>MFP-ST Interrupt In-Service Register A (see mapping at 0x00FFFA06).</td>
</tr>
<tr>
<td>0x00FFFA10</td>
<td>OB</td>
<td>I/O</td>
<td>MFP-ST Interrupt In-Service Register B (see mapping at 0x00FFFA08).</td>
</tr>
<tr>
<td>0x00FFFA12</td>
<td>OB</td>
<td>I/O</td>
<td>MFP-ST Interrupt Mask Register A (see mapping at 0x00FFFA06).</td>
</tr>
<tr>
<td>0x00FFFA14</td>
<td>OB</td>
<td>I/O</td>
<td>MFP-ST Interrupt Mask Register B (see mapping at 0x00FFFA08).</td>
</tr>
<tr>
<td>Location(s)</td>
<td>Size</td>
<td>Type</td>
<td>Meaning</td>
</tr>
<tr>
<td>-------------</td>
<td>------</td>
<td>------</td>
<td>---------</td>
</tr>
<tr>
<td>0x00FFFA16</td>
<td>OB</td>
<td>I/O</td>
<td>MFP-ST Vector Register. Bit 3 is set to 1 to indicate software End-of-Interrupt mode and 0 to indicate automatic End-of-Interrupt mode.</td>
</tr>
<tr>
<td>0x00FFFA18</td>
<td>OB</td>
<td>I/O</td>
<td>MFP-ST Timer A Control Register. Interpret bits 3-0 as follows:</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td><strong>Value</strong></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>0000</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>0001</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>0010</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>0011</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>0100</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>0101</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>0110</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>0111</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>1000</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>1xxx</td>
</tr>
<tr>
<td>0x00FFFA1A</td>
<td>OB</td>
<td>I/O</td>
<td>MFP-ST Timer B Control Register (see Timer A).</td>
</tr>
<tr>
<td>0x00FFFA1C</td>
<td>OB</td>
<td>I/O</td>
<td>MFP-ST Timer C &amp; D Control Register. Interpret bits 6-4 for Timer C and bits 2-0 for Timer D as follows:</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td><strong>Value</strong></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>0000</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>0001</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>0010</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>0011</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>0100</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>0101</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>0110</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>0111</td>
</tr>
<tr>
<td>0x00FFFA1E</td>
<td>OB</td>
<td>I/O</td>
<td>MFP-ST Timer A Data Register.</td>
</tr>
<tr>
<td>0x00FFFA20</td>
<td>OB</td>
<td>I/O</td>
<td>MFP-ST Timer B Data Register.</td>
</tr>
<tr>
<td>0x00FFFA22</td>
<td>OB</td>
<td>I/O</td>
<td>MFP-ST Timer C Data Register.</td>
</tr>
<tr>
<td>0x00FFFA24</td>
<td>OB</td>
<td>I/O</td>
<td>MFP-ST Timer D Data Register.</td>
</tr>
<tr>
<td>0x00FFFA26</td>
<td>OB</td>
<td>I/O</td>
<td>MFP-ST Sync Character Register.</td>
</tr>
</tbody>
</table>
### MFP-ST USART Control Register as follows:

<table>
<thead>
<tr>
<th>Bit 7</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>00</td>
<td>8 bits</td>
</tr>
<tr>
<td>01</td>
<td>7 bits</td>
</tr>
<tr>
<td>10</td>
<td>6 bits</td>
</tr>
<tr>
<td>11</td>
<td>5 bits</td>
</tr>
</tbody>
</table>

- **Clock**
  - (If set, divide by 16.)
  - 00 = Synchronous
  - 01 = 1 Stop, 1 Start
  - 10 = 1 Stop, 1½ Start
  - 11 = 1 Stop, 2 Start

- **Bit 0**
  - Unused
  - If set, ignore parity.
  - 1 = Even parity
  - 0 = Odd parity

### MFP-ST Receiver Status Register as follows:

<table>
<thead>
<tr>
<th>Bit 7</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>Frame Error</td>
</tr>
<tr>
<td>1</td>
<td>Parity Error</td>
</tr>
<tr>
<td>2</td>
<td>Overrun Error</td>
</tr>
<tr>
<td>3</td>
<td>Buffer Full</td>
</tr>
<tr>
<td>4</td>
<td>Search/Break Detected</td>
</tr>
<tr>
<td>5</td>
<td>Match/Character in Progress</td>
</tr>
<tr>
<td>6</td>
<td>Synchronous Strip Enable</td>
</tr>
<tr>
<td>7</td>
<td>Receiver Enable Bit</td>
</tr>
</tbody>
</table>

### MFP-ST Transmitter Status Register as follows:

<table>
<thead>
<tr>
<th>Bit 7</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>Buffer Empty</td>
</tr>
<tr>
<td>1</td>
<td>Underrun Error</td>
</tr>
<tr>
<td>2</td>
<td>Auto Turnaround</td>
</tr>
<tr>
<td>3</td>
<td>End of Transmission</td>
</tr>
<tr>
<td>4</td>
<td>Break</td>
</tr>
<tr>
<td>5</td>
<td>High Bit</td>
</tr>
<tr>
<td>6</td>
<td>Low Bit</td>
</tr>
<tr>
<td>7</td>
<td>Transmitter Enable</td>
</tr>
</tbody>
</table>
### MFP-ST USART Data Register.
- **Address:** 0x00FFFA2E
- **Type:** OB
- **Description:** I/O MFP-ST USART Data Register.

### Unassigned
- **Address:** 0x00FFFA30 – 0x00FFFA3F
- **Type:** N/A
- **Description:** I/O Unassigned

### 68881 Math Co-Processor in Peripheral Mode

<table>
<thead>
<tr>
<th>Address</th>
<th>Type</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>0x00FFFA40</td>
<td>WORD</td>
<td>FPCIR Status Register (available as a Mega Bus card accessed in 68881 peripheral mode)</td>
</tr>
<tr>
<td>0x00FFFA42</td>
<td>WORD</td>
<td>FPCTL Control Register (available as a Mega Bus card accessed in 68881 peripheral mode)</td>
</tr>
<tr>
<td>0x00FFFA44</td>
<td>WORD</td>
<td>FPSAV Save Register (available as a Mega Bus card accessed in 68881 peripheral mode)</td>
</tr>
<tr>
<td>0x00FFFA46</td>
<td>WORD</td>
<td>FPREST Restore Register (available as a Mega Bus card accessed in 68881 peripheral mode)</td>
</tr>
<tr>
<td>0x00FFFA48</td>
<td>WORD</td>
<td>FPOPR Operation Word Register (available as a Mega Bus card accessed in 68881 peripheral mode)</td>
</tr>
<tr>
<td>0x00FFFA4A</td>
<td>WORD</td>
<td>FPCMD Command Register (available as a Mega Bus card accessed in 68881 peripheral mode)</td>
</tr>
<tr>
<td>0x00FFFA4C</td>
<td>WORD</td>
<td>FPRES Reserved (available as a Mega Bus card accessed in 68881 peripheral mode)</td>
</tr>
<tr>
<td>0x00FFFA4E</td>
<td>WORD</td>
<td>FPCCR Condition Code Register (available as a Mega Bus card accessed in 68881 peripheral mode)</td>
</tr>
<tr>
<td>0x00FFFA50</td>
<td>LONG</td>
<td>FPOP Operand Register (available as a Mega Bus card accessed in 68881 peripheral mode)</td>
</tr>
<tr>
<td>0x00FFFA54</td>
<td>WORD</td>
<td>FPSLCT Register Select (available as a Mega Bus card accessed in 68881 peripheral mode)</td>
</tr>
<tr>
<td>0x00FFFA56</td>
<td>WORD</td>
<td>Reserved</td>
</tr>
<tr>
<td>0x00FFFA58</td>
<td>LONG</td>
<td>FPIADR Instruction Address (available as a Mega Bus card accessed in 68881 peripheral mode)</td>
</tr>
<tr>
<td>0x00FFFA5C</td>
<td>LONG</td>
<td>FPOADR Operand Address (available as a Mega Bus card accessed in 68881 peripheral mode)</td>
</tr>
<tr>
<td>0x00FFFA54 – 0x00FFFA7F</td>
<td>N/A</td>
<td>Unassigned</td>
</tr>
</tbody>
</table>

### TT030 Multi-Function Peripheral Port (68901)

<table>
<thead>
<tr>
<th>Address</th>
<th>Type</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>0x00FFFA80</td>
<td>OB</td>
<td>MFP-TT030 GPIP (see 0x00FFFFFA00).</td>
</tr>
<tr>
<td>0x00FFFA82</td>
<td>OB</td>
<td>MFP-TT030 AER (see 0x00FFFFFA02).</td>
</tr>
<tr>
<td>0x00FFFA84</td>
<td>OB</td>
<td>MFP-TT030 DDR (see 0x00FFFFFA04).</td>
</tr>
<tr>
<td>0x00FFFA86</td>
<td>OB</td>
<td>MFP-TT030 IERA (see 0x00FFFFFA06).</td>
</tr>
<tr>
<td>0x00FFFA88</td>
<td>OB</td>
<td>MFP-TT030 IERB (see 0x00FFFFFA08).</td>
</tr>
<tr>
<td>0x00FFFA8A</td>
<td>OB</td>
<td>MFP-TT030 IPRA (see 0x00FFFFFA0A).</td>
</tr>
<tr>
<td>0x00FFFA8C</td>
<td>OB</td>
<td>MFP-TT030 IPRB (see 0x00FFFFFA0C).</td>
</tr>
</tbody>
</table>
## B.42 – Memory Map

<table>
<thead>
<tr>
<th>Location(s)</th>
<th>Size</th>
<th>Type</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>0x00FFFA8E</td>
<td>OB</td>
<td>I/O</td>
<td>MFP-TT030 ISRA (see 0x00FFFA0E).</td>
</tr>
<tr>
<td>0x00FFFA90</td>
<td>OB</td>
<td>I/O</td>
<td>MFP-TT030 ISRB (see 0x00FFFA10).</td>
</tr>
<tr>
<td>0x00FFFA92</td>
<td>OB</td>
<td>I/O</td>
<td>MFP-TT030 IMRA (see 0x00FFFA12).</td>
</tr>
<tr>
<td>0x00FFFA94</td>
<td>OB</td>
<td>I/O</td>
<td>MFP-TT030 IMRB (see 0x00FFFA14).</td>
</tr>
<tr>
<td>0x00FFFA96</td>
<td>OB</td>
<td>I/O</td>
<td>MFP-TT030 VR (see 0x00FFFA16).</td>
</tr>
<tr>
<td>0x00FFFA98</td>
<td>OB</td>
<td>I/O</td>
<td>MFP-TT030 TACR (see 0x00FFFA18).</td>
</tr>
<tr>
<td>0x00FFFA9A</td>
<td>OB</td>
<td>I/O</td>
<td>MFP-TT030 TBCR (see 0x00FFFA1A).</td>
</tr>
<tr>
<td>0x00FFFA9C</td>
<td>OB</td>
<td>I/O</td>
<td>MFP-TT030 TCDCR (see 0x00FFFA1C).</td>
</tr>
<tr>
<td>0x00FFFA9E</td>
<td>OB</td>
<td>I/O</td>
<td>MFP-TT030 TADR (see 0x00FFFA1E).</td>
</tr>
<tr>
<td>0x00FFFAA0</td>
<td>OB</td>
<td>I/O</td>
<td>MFP-TT030 TBDR (see 0x00FFFA20).</td>
</tr>
<tr>
<td>0x00FFFAA2</td>
<td>OB</td>
<td>I/O</td>
<td>MFP-TT030 TCDR (see 0x00FFFA22).</td>
</tr>
<tr>
<td>0x00FFFAA4</td>
<td>OB</td>
<td>I/O</td>
<td>MFP-TT030 TDDR (see 0x00FFFA24).</td>
</tr>
<tr>
<td>0x00FFFAA6</td>
<td>OB</td>
<td>I/O</td>
<td>MFP-TT030 SCR (see 0x00FFFA26).</td>
</tr>
<tr>
<td>0x00FFFAA8</td>
<td>OB</td>
<td>I/O</td>
<td>MFP-TT030 UCR (see 0x00FFFA28).</td>
</tr>
<tr>
<td>0x00FFFFAA</td>
<td>OB</td>
<td>I/O</td>
<td>MFP-TT030 RSR (see 0x00FFFA2A).</td>
</tr>
<tr>
<td>0x00FFFFAC</td>
<td>OB</td>
<td>I/O</td>
<td>MFP-TT030 TSR (see 0x00FFFA2C).</td>
</tr>
<tr>
<td>0x00FFFFAE</td>
<td>OB</td>
<td>I/O</td>
<td>MFP-TT030 UDR (see 0x00FFFA2E).</td>
</tr>
<tr>
<td>0x00FFFFAB0–0x00FFFFBFF</td>
<td>N/A</td>
<td>I/O</td>
<td>Undefined</td>
</tr>
</tbody>
</table>

T H E  A T A R I  C O M P E N D I U M
<table>
<thead>
<tr>
<th>Location(s)</th>
<th>Size</th>
<th>Type</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>0x00FFFC00</td>
<td>EB</td>
<td>I/O</td>
<td>Keyboard ACIA Control (when written) as follows:</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Bit #7</td>
<td></td>
<td></td>
<td>Enables receive interrupts</td>
</tr>
<tr>
<td>Bits #6–5</td>
<td></td>
<td></td>
<td>Configures transmitter interrupts as follows:</td>
</tr>
<tr>
<td>Value</td>
<td>Meaning</td>
<td></td>
<td></td>
</tr>
<tr>
<td>%00</td>
<td>RTS low, Disable Interrupts</td>
<td></td>
<td></td>
</tr>
<tr>
<td>%01</td>
<td>RTS low, Enable Interrupts</td>
<td></td>
<td></td>
</tr>
<tr>
<td>%10</td>
<td>RTS high, Disable Interrupts</td>
<td></td>
<td></td>
</tr>
<tr>
<td>%11</td>
<td>RTS low, Disable Interrupts</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>Send a break on Interrupt</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Bits #4–2</td>
<td></td>
<td></td>
<td>Configure Port Settings as follows:</td>
</tr>
<tr>
<td>Value</td>
<td>Data Bits–Parity–Stop Bits</td>
<td></td>
<td></td>
</tr>
<tr>
<td>%000</td>
<td>7-E-2</td>
<td></td>
<td></td>
</tr>
<tr>
<td>%001</td>
<td>7-O-2</td>
<td></td>
<td></td>
</tr>
<tr>
<td>%010</td>
<td>7-E-1</td>
<td></td>
<td></td>
</tr>
<tr>
<td>%011</td>
<td>7-O-1</td>
<td></td>
<td></td>
</tr>
<tr>
<td>%100</td>
<td>8-N-2</td>
<td></td>
<td></td>
</tr>
<tr>
<td>%101</td>
<td>8-N-1</td>
<td></td>
<td></td>
</tr>
<tr>
<td>%110</td>
<td>8-E-1</td>
<td></td>
<td></td>
</tr>
<tr>
<td>%111</td>
<td>8-O-1</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Bits #1–0</td>
<td></td>
<td></td>
<td>Set Clock Divisor as follows:</td>
</tr>
<tr>
<td>Value</td>
<td>Meaning</td>
<td></td>
<td></td>
</tr>
<tr>
<td>%00</td>
<td>Normal</td>
<td></td>
<td></td>
</tr>
<tr>
<td>%01</td>
<td>Divide by 16</td>
<td></td>
<td></td>
</tr>
<tr>
<td>%10</td>
<td>Divide by 256</td>
<td></td>
<td></td>
</tr>
<tr>
<td>%11</td>
<td>Master Reset</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
### Memory Map

<table>
<thead>
<tr>
<th>Location(s)</th>
<th>Type</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>0x00FFFC02</td>
<td>EB</td>
<td>I/O Keyboard ACIA Data</td>
</tr>
<tr>
<td>0x00FFFC04</td>
<td>EB</td>
<td>I/O MIDI ACIA Control (see keyboard ACIA control register for details)</td>
</tr>
<tr>
<td>0x00FFFC06</td>
<td>EB</td>
<td>I/O MIDI ACIA Data</td>
</tr>
</tbody>
</table>
| 0x00FFFC20  | OB   | I/O Bank 0: Seconds-Ones (0–9)  
Bank 1: Clock output frequency as follows: |
|             |      | **Value** | **Meaning** |
|             |      | 0          | Open-Collector “CLKOUT” |
|             |      | 1          | 16384 Hz |
|             |      | 2          | 1024 Hz |
|             |      | 3          | 128 Hz |
|             |      | 4          | 16 Hz |
|             |      | 5          | 1 Hz |
|             |      | 6          | 1/60 Hz |
|             |      | 7          | Open-Collector “CLKOUT” |
| 0x00FFFC22  | OB   | I/O Bank 0: Seconds-Tens (0–5)  
Bank 1: Setting bit #0 will reset the seconds register to the 0 and, if the seconds register is currently between 30–59, increment the minutes register. |
| 0x00FFFC24  | OB   | I/O Bank 0: Minutes-Ones (0–9)  
Bank 1: Alarm Minutes-Ones (0–9) |

---

**Keyboard ACIA Control (when read) as follows:**

- **Bit 0:** Framing Error
- **Bit 1:** Receiver Overrun
- **Bit 2:** Parity Error
- **Bit 3:** Interrupt Request
- **Bit 4:** Clear to Send
- **Bit 5:** Data Carrier Detect
- **Bit 6:** Receiver Full
- **Bit 7:** Transmitter Empty

---

**MIDI ACIA (6850)**

- **Value** | **Meaning** |
- 0          | Open-Collector “CLKOUT” |
- 1          | 16384 Hz |
- 2          | 1024 Hz |
- 3          | 128 Hz |
- 4          | 16 Hz |
- 5          | 1 Hz |
- 6          | 1/60 Hz |
- 7          | Open-Collector “CLKOUT” |

---

**Mega ST Real Time Clock (RP5C15)**

- **Value** | **Meaning** |
- 0          | Open-Collector “CLKOUT” |
- 1          | 16384 Hz |
- 2          | 1024 Hz |
- 3          | 128 Hz |
- 4          | 16 Hz |
- 5          | 1 Hz |
- 6          | 1/60 Hz |
- 7          | Open-Collector “CLKOUT” |

---

**The Atari Compendium**
<table>
<thead>
<tr>
<th>Location(s)</th>
<th>Size</th>
<th>Type</th>
<th>Meaning</th>
</tr>
</thead>
</table>
| 0x00FFFC26   | OB   | I/O  | Bank 0: Minutes–Tens (0–5)  
Bank 1: Alarm Minutes–Tens (0–5) |
| 0x00FFFC28   | OB   | I/O  | Bank 0: Hour–Ones (0–9)  
Bank 1: Alarm Hour–Ones (0–9) |
| 0x00FFFC2A   | OB   | I/O  | Bank 0: Hour–Tens (0–2), in 24 hour mode, otherwise (0–1) in 12 hour mode with Bit 1 being set for PM, cleared for AM.  
Bank 1: Alarm Hour–Tens (as in bank 0) |
| 0x00FFFC2C   | OB   | I/O  | Bank 0: Day of Week (0–6), 0 = Sunday  
Bank 1: Alarm Day of Week (0–6), 0 = Sunday |
| 0x00FFFC2E   | OB   | I/O  | Bank 0: Date–Ones (0–9)  
Bank 1: Alarm Date–Ones (0–9) |
| 0x00FFFC30   | OB   | I/O  | Bank 0: Date–Tens (0–3)  
Bank 1: Alarm Date–Tens (0–3) |
| 0x00FFFC32   | OB   | I/O  | Bank 0: Month–Ones (0–9)  
Bank 1: Not Used |
| 0x00FFFC34   | OB   | I/O  | Bank 0: Month–Tens (0–1)  
Bank 1: If Bit #1 is set then clock is in 24 hour mode, otherwise, it is in 12 hour mode. |
| 0x00FFFC36   | OB   | I/O  | Bank 0: Year–Ones (0–9). The value for Year represents the (Year - 1980).  
Bank 1: Leap Year Register (0–3), 0 = Leap Year |
| 0x00FFFC38   | OB   | I/O  | Bank 0: Year–Tens (0–9)  
Bank 1: Not Used |
| 0x00FFFC3A   | OB   | I/O  | Mode Register as follows:  
0 = Clock Stop  
0 = Alarm off |
| 0x00FFFC3C   | OB   | I/O  | Test Register (lower nibble must equal zero to show confirm proper functioning) |
### B.46 – Memory Map

<table>
<thead>
<tr>
<th>Location(s)</th>
<th>Size</th>
<th>Type</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>0x00FFFC3E</td>
<td>OB</td>
<td>I/O</td>
<td>Reset Register as follows:</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>0 = 0 Hz Alarm Pulse</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>0 = 16 Hz Alarm Pulse</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>Bit 0</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>1 = Clock Reset</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>1 = Alarm Reset</td>
</tr>
</tbody>
</table>

### Expansion Area

<table>
<thead>
<tr>
<th>Location(s)</th>
<th>Size</th>
<th>Type</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>0x01000000 – 0x01FFFFFF</td>
<td>N/A</td>
<td>RAM</td>
<td>TT030 Fast Ram (Unsuitable for direct DMA and Video Shifter transfers)</td>
</tr>
<tr>
<td>0x02000000 – 0xFDFFFFFF</td>
<td>N/A</td>
<td>RSVD</td>
<td>Reserved</td>
</tr>
<tr>
<td>0xFE000000 – 0xFEFEFFFF</td>
<td>N/A</td>
<td>VME</td>
<td>VME A24:D16 Addressable Area</td>
</tr>
<tr>
<td>0xFEFF0000 – 0xFEFFFFF</td>
<td>N/A</td>
<td>VME</td>
<td>VME A16:D16 Addressable Area</td>
</tr>
</tbody>
</table>

### Shadow Image

<table>
<thead>
<tr>
<th>Location(s)</th>
<th>Size</th>
<th>Type</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>0xFF000000 – 0xFFFFFFF</td>
<td>N/A</td>
<td>Image</td>
<td>This area is a ‘shadow’ image of 0x00000000 – 0x00FFFFFF to remain compatible with the ST.</td>
</tr>
</tbody>
</table>
NATIVE FILE FORMATS
The .GEM File Format

Files ending in ‘.GEM’ are graphic metafiles created by GDOS. They are usually used to represent vector graphics but may also be used to store links to bitmap images and textual information.

Two primary versions of GEM files exist. Version 1 files are guaranteed not to contain bezier curves whereas version 3 files may. Version 3.xx files are also commonly referred to as GEM/3 files.

The Metafile Header

GEM metafiles begin with a header as follows:

<table>
<thead>
<tr>
<th>WORD</th>
<th>Contents</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>Magic number (0xFFFF).</td>
</tr>
<tr>
<td>1</td>
<td>Header length in WORDs.</td>
</tr>
<tr>
<td>2</td>
<td>Version number (major * 100 + minor).</td>
</tr>
<tr>
<td>3</td>
<td>NDC Flag as follows:</td>
</tr>
<tr>
<td></td>
<td><strong>Value</strong>  <strong>Meaning</strong></td>
</tr>
<tr>
<td></td>
<td>0</td>
</tr>
<tr>
<td></td>
<td>2</td>
</tr>
<tr>
<td>4</td>
<td>Minimum X extent.</td>
</tr>
<tr>
<td>5</td>
<td>Minimum Y extent.</td>
</tr>
<tr>
<td>6</td>
<td>Maximum X extent.</td>
</tr>
<tr>
<td>7</td>
<td>Maximum Y extent.</td>
</tr>
<tr>
<td>8</td>
<td>Page width in tenths of millimeters.</td>
</tr>
<tr>
<td>9</td>
<td>Page height in tenths of millimeters.</td>
</tr>
<tr>
<td>10</td>
<td>Lower Left X value of coordinate system.</td>
</tr>
<tr>
<td>11</td>
<td>Lower Left Y value of coordinate system.</td>
</tr>
<tr>
<td>12</td>
<td>Upper Right X value of coordinate system.</td>
</tr>
<tr>
<td>13</td>
<td>Upper Right Y value of coordinate system.</td>
</tr>
<tr>
<td>...</td>
<td>Other information may appear in the header</td>
</tr>
<tr>
<td></td>
<td>following which is currently undefined. Use</td>
</tr>
<tr>
<td></td>
<td>WORD #1 to skip any unknown information.</td>
</tr>
</tbody>
</table>

The definition of WORDs 4–13 is defined by the creator of the file using three metafile commands. WORDs 4–7 are set with the v_meta_extents() function. WORDs 8–9 are defined with the vm_pagesize() function. WORDs 10–13 are defined with vm_coords(). If the creator fails to specify defaults for any of these values, the appropriate values will be set to 0 in the header. If zeros appear for WORDs 10–13, the default NDC coordinate system should be assumed.
Metafile Records
Following the header will appear a list of records of varying length which, when translated, can be ‘played back’ on the destination VDI device. Each record is formatted as follows:

<table>
<thead>
<tr>
<th>WORD</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>Opcode of VDI function.</td>
</tr>
<tr>
<td>1</td>
<td>Number of PTSIN elements.</td>
</tr>
<tr>
<td>2</td>
<td>Number of INTIN elements.</td>
</tr>
<tr>
<td>3</td>
<td>Function sub-ID.</td>
</tr>
<tr>
<td>4...</td>
<td>PTSIN elements.</td>
</tr>
<tr>
<td>...</td>
<td>INTIN elements.</td>
</tr>
</tbody>
</table>

The list of records is terminated with an opcode of 0xFFFF (this record is written when a v_clswk() call is made by the creator).

When playing back GEM files, the application must translate all coordinates from the metafile coordinate system to that of the destination device. In addition, text metrics should be appropriately converted. If an unknown opcode is discovered it should be played after any elements of the PTSIN array are translated (making the assumption that they should be).

Metafile Sub-Opcodes
GEM metafiles support the use of special sub-opcodes for implementing reserved and user-defined functions. GEM metafile translators should ignore sub-opcodes they don’t understand. Each sub-opcode can be identified with the primary opcode of 5, function ID of 99 and the first (required) member of INTIN being the sub-opcode ID. The currently defined sub-opcodes are as follows:

<table>
<thead>
<tr>
<th>INTIN[0]</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>10</td>
<td>Start Group.</td>
</tr>
<tr>
<td>11</td>
<td>End Group.</td>
</tr>
<tr>
<td>49</td>
<td>Set No Line Style.</td>
</tr>
<tr>
<td>50</td>
<td>Set Attribute Shadow On.</td>
</tr>
<tr>
<td>51</td>
<td>Set Attribute Shadow Off.</td>
</tr>
<tr>
<td>80</td>
<td>Start Draw Area Type Primitive.</td>
</tr>
<tr>
<td>81</td>
<td>End Draw Area Type Primitive.</td>
</tr>
</tbody>
</table>

None of the pre-defined sub-opcodes use additional INTIN or PTSIN elements though user-defined sub-opcodes may.

Opcodes from 0–100 are reserved for use by Atari. Sub-opcodes from 101-65535 are available for use by developers but should be registered with Atari to avoid possible conflicts.
The .IMG File Format

The IMG file format was designed to support raster images with a varying number of planes. In practice, almost all IMG files currently available are simple black and white single plane images because the original file format did not specify a method of storing palette information with the file. To fill this need, several unofficial extensions to the format were put into use (some of which were incorrectly implemented by applications supporting them). The color extension which will be discussed here to cover color images is the ‘XIMG’ format.

The IMG Header

Image headers consist of at least 8 WORDs as follows:

<table>
<thead>
<tr>
<th>WORD</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>Image file version (Usually 0x0001).</td>
</tr>
<tr>
<td>1</td>
<td>Header length in WORDs.</td>
</tr>
<tr>
<td>2</td>
<td>Number of planes.</td>
</tr>
<tr>
<td>3</td>
<td>Pattern definition length.</td>
</tr>
<tr>
<td>4</td>
<td>Source device pixel width (in microns).</td>
</tr>
<tr>
<td>5</td>
<td>Source device pixel height (in microns).</td>
</tr>
<tr>
<td>6</td>
<td>Scan line width (in pixels).</td>
</tr>
<tr>
<td>7</td>
<td>Number of scan lines.</td>
</tr>
</tbody>
</table>

Some IMG files will have additional header information which should be skipped or interpreted as discussed below.

Interpreting Extra Palette Information

If WORD #2 is set to 1, then the image data consists of one plane (i.e. monochrome) and any extra header information should be ignored.

If WORD #2 is set to 16 or 24 then the image data consists of that many planes of high color or true color data and any extra header information should be ignored. In a high color image, planes appear in the order RRRRRR GGGGGG BBBBBB. In a true-color image, planes appear in the order RRRRRRRR GGGGGGGG BBBBBBBB.

If WORD #2 is set to 2, 4, or 8, the image consists of palette based color image data. If no extra header information is given then the creator did not specify palette data for this image. If extra header WORDs appears they may be useful in determining the color palette. The two primary extensions to the IMG format are ‘XIMG’ and ‘STTT’. ‘STTT’ will not be discussed here as it does not serve well as a machine or device independent format. The ‘XIMG’ header extension is as follows:
## Image Data Format

Each scanline contains data in VDI device independent format which must be converted using the VDI call `vr_trnfm()`. Each scanline is padded to the nearest byte. Every plane for each scanline should appear prior to the beginning of data for the next scanline. This allows interpreters to decompress and transform the image data a scanline at a time to conserve on time and memory. A sample ordering for a four-plane image is listed below:

<table>
<thead>
<tr>
<th>Scanline #0 – Plane #0</th>
<th>Scanline #0 – Plane #1</th>
<th>Scanline #0 – Plane #2</th>
<th>Scanline #0 – Plane #3</th>
</tr>
</thead>
<tbody>
<tr>
<td>Scanline #1 – Plane #0</td>
<td>Scanline #1 – Plane #1</td>
<td>Scanline #1 – Plane #2</td>
<td>Scanline #1 – Plane #3</td>
</tr>
</tbody>
</table>

etc.

## Image Compression

Each scanline is individually compressed. This means that compression codes should not transgress over scanline boundaries. This enables decompression routines to work scanline by scanline.

Scanline data should consist of two components, a vertical replication count and encoded scanline data. In practice, however, some older .IMG files may not contain a vertical replication count for each scan line.

The vertical replication count specifies the number of times the following scanline data should be used to replicate an image row. It is formatted as follows:

<table>
<thead>
<tr>
<th>BYTE</th>
<th>Contents</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0x00</td>
</tr>
<tr>
<td>1</td>
<td>0x00</td>
</tr>
<tr>
<td>2</td>
<td>0xFF</td>
</tr>
<tr>
<td>3</td>
<td>Replication Count</td>
</tr>
</tbody>
</table>

Immediately following the vertical replication count is the encoded scanline data. This run-length encoding can by looking for three separate flag BYTEs. A 0x80 BYTE indicates the beginning of a bit-string item. A bit-string item is formatted as follows:
A pattern-run item begins with a **BYTE** of 0x00. It specifies a fixed number of times that the pattern which follows it should be repeated. It is formatted as follows:

<table>
<thead>
<tr>
<th><strong>BYTE</strong></th>
<th>Contents</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0x80</td>
</tr>
<tr>
<td>1</td>
<td>Byte count ‘n’.</td>
</tr>
<tr>
<td>2...</td>
<td>‘n’ <strong>BYTE</strong>s of unencoded data.</td>
</tr>
</tbody>
</table>

Finally, a solid-run item begins with any other **BYTE** code. If the high order bit is set then this indicates a run of black pixels, otherwise it indicates a run of white pixels. The lower 7 bits of the byte indicates the length of the run in bytes. For example a **BYTE** code of 0x83 indicates a run of 24 black pixels (3 bytes).

### The .FNT File Format

Filenames ending with the extension `.FNT` represent bitmap font files. These files may be utilized by loading them through any version of **GDOS**. FNT files are composed of a file header, font data, a character offset table, and (optionally) a horizontal offset table.

#### The FNT Header

Font files begin with a header 88 **BYTE**s long. **WORD** and **LONG** format entries in the header must be byte-swapped as they appear in Intel (`‘Little Endian’`) format. The font header is formatted as follows:

<table>
<thead>
<tr>
<th><strong>BYTE(s)</strong></th>
<th>Contents</th>
<th>Related VDI Call</th>
</tr>
</thead>
<tbody>
<tr>
<td>0 – 1</td>
<td>Face ID (must be unique).</td>
<td><code>vqt_name()</code></td>
</tr>
<tr>
<td>2 – 3</td>
<td>Face size (in points).</td>
<td><code>vst_point()</code></td>
</tr>
<tr>
<td>4 – 35</td>
<td>Face name.</td>
<td><code>vqt_name()</code></td>
</tr>
<tr>
<td>36 – 37</td>
<td>Lowest character index in face</td>
<td><code>vqt_fontinfo()</code></td>
</tr>
<tr>
<td></td>
<td>(usually 32 for disk-loaded fonts).</td>
<td></td>
</tr>
<tr>
<td>38 – 39</td>
<td>Highest character index in face</td>
<td><code>vqt_fontinfo()</code></td>
</tr>
<tr>
<td>40 – 41</td>
<td>Top line distance expressed as a</td>
<td><code>vqt_fontinfo()</code></td>
</tr>
<tr>
<td></td>
<td>positive offset from baseline.</td>
<td></td>
</tr>
<tr>
<td>42 – 43</td>
<td>Ascent line distance expressed as a</td>
<td><code>vqt_fontinfo()</code></td>
</tr>
<tr>
<td></td>
<td>positive offset from baseline.</td>
<td></td>
</tr>
<tr>
<td>44 – 45</td>
<td>Half line distance expressed as a</td>
<td><code>vqt_fontinfo()</code></td>
</tr>
<tr>
<td></td>
<td>positive offset from baseline.</td>
<td></td>
</tr>
<tr>
<td>46 – 47</td>
<td>Descent line distance expressed as a</td>
<td><code>vqt_fontinfo()</code></td>
</tr>
<tr>
<td></td>
<td>positive offset from baseline.</td>
<td></td>
</tr>
<tr>
<td>48 – 49</td>
<td>Bottom line distance expressed as a positive offset from baseline.</td>
<td>vqt_fontinfo()</td>
</tr>
<tr>
<td>50 – 51</td>
<td>Width of the widest character.</td>
<td>N/A</td>
</tr>
<tr>
<td>52 – 53</td>
<td>Width of the widest character cell.</td>
<td>vqt_fontinfo()</td>
</tr>
<tr>
<td>54 – 55</td>
<td>Left offset.</td>
<td>vqt_fontinfo()</td>
</tr>
<tr>
<td>56 – 57</td>
<td>Right offset.</td>
<td>vqt_fontinfo()</td>
</tr>
<tr>
<td>58 – 59</td>
<td>Thickening size (in pixels).</td>
<td>vqt_fontinfo()</td>
</tr>
<tr>
<td>60 – 61</td>
<td>Underline size (in pixels).</td>
<td>vqt_fontinfo()</td>
</tr>
<tr>
<td>62 – 63</td>
<td>Lightening mask (used to eliminate pixels, usually 0x5555).</td>
<td>N/A</td>
</tr>
<tr>
<td>64 – 65</td>
<td>Skewing mask (rotated to determine when to perform additional rotation on a character when skewing, usually 0x5555).</td>
<td>N/A</td>
</tr>
<tr>
<td>66 – 67</td>
<td>Font flags as follows:</td>
<td></td>
</tr>
<tr>
<td>Bit</td>
<td>Meaning (if Set)</td>
<td></td>
</tr>
<tr>
<td>0</td>
<td>Contains System Font</td>
<td></td>
</tr>
<tr>
<td>1</td>
<td>Horizontal Offset Tables should be used.</td>
<td></td>
</tr>
<tr>
<td>2</td>
<td>Font data need not be byte-swapped.</td>
<td></td>
</tr>
<tr>
<td>3</td>
<td>Font is mono-spaced.</td>
<td></td>
</tr>
<tr>
<td>68 – 71</td>
<td>Offset from start of file to horizontal offset table.</td>
<td>vqt_width()</td>
</tr>
<tr>
<td>72 – 75</td>
<td>Offset from start of file to character offset table.</td>
<td>vqt_width()</td>
</tr>
<tr>
<td>76 – 79</td>
<td>Offset from start of file to font data.</td>
<td>N/A</td>
</tr>
<tr>
<td>80 – 81</td>
<td>Form width (in bytes).</td>
<td>N/A</td>
</tr>
<tr>
<td>82 – 83</td>
<td>Form height (in scanlines).</td>
<td>N/A</td>
</tr>
<tr>
<td>84 – 87</td>
<td>Pointer to the next font (set by GDOS after loading).</td>
<td>N/A</td>
</tr>
</tbody>
</table>

**Font Data**

The binary font data is arranged on a single raster form. The raster’s height is the same as the font’s height. The raster’s width is the sum of the character width’s padded to end on a WORD boundary.

There is no padding between characters. Each character may overlap BYTE boundaries. Only the last character in a font is padded to make the width of the form end on an even WORD boundary.

If bit #2 of the font flags header item is cleared, each WORD in the font data must be byte-swapped.
Character Offset Table
The Character Offset Table is an array of WORDs which specifies the distance (in pixels) from the previous character to the next. The first entry is the distance from the start of the raster form to the left side of the first character. One succeeding entry follows for each character in the font yielding (number of characters + 1) entries in the table. Each entry must be byte-swapped as it appears in Intel (‘Little Endian’) format.

Horizontal Offset Table
The Horizontal Offset Table is an optional array of positive or negative WORD values which when added to the values in the character offset table yield the true spacing information for each character. One entry appears in the table for each character. This table is not often used.

The .RSC File Format
Resource files contain application specific data which is generally loaded at run-time. RSC files contain OBJECT trees (see the discussion of the OBJECT structure in Chapter 6: AES), strings, and images.

Two resource file formats are currently supported. TOS versions less than 4.0 support the original RSC format while TOS 4.0 and greater will now support the older format and a new extensible format. The original format will be discussed first followed by an explanation of the changes incurred by the newer format.

The RSC Header
Resource files begin with an 18 WORD header as follows:

<table>
<thead>
<tr>
<th>WORD</th>
<th>Field Name</th>
<th>Contents</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>rsh_vrsn</td>
<td>Contains the version number of the resource file. This value is 0x0000 or 0x0001 in old format RSC files and has the third bit set (i.e. 0x0004) in the new file format.</td>
</tr>
<tr>
<td>1</td>
<td>rsh_object</td>
<td>Contains an offset from the beginning of the file to the OBJECT structures.</td>
</tr>
<tr>
<td>2</td>
<td>rsh_tedinfo</td>
<td>Contains an offset from the beginning of the file to the TEDINFO structures.</td>
</tr>
<tr>
<td>3</td>
<td>rsh_iconblk</td>
<td>Contains an offset from the beginning of the file to the ICONBLK structures.</td>
</tr>
<tr>
<td>4</td>
<td>rsh_bitblk</td>
<td>Contains an offset from the beginning of the file to the BITBLK structures.</td>
</tr>
<tr>
<td>5</td>
<td>rsh_frstr</td>
<td>Contains an offset from the beginning of the file to the string pointer table.</td>
</tr>
<tr>
<td>6</td>
<td>rsh_string</td>
<td>Contains an offset from the beginning of the file to the string data.</td>
</tr>
<tr>
<td>7</td>
<td>rsh_imdata</td>
<td>Contains an offset from the beginning of the file to the image data.</td>
</tr>
<tr>
<td>8</td>
<td>rsh_frimg</td>
<td>Contains an offset from the beginning of the file to the image pointer table.</td>
</tr>
</tbody>
</table>
Many of the header entries represent offsets from the beginning of the file. These offsets are expressed as positive unsigned **WORDS** making the standard file a maximum size of 64k bytes.

### Object Trees

Each RSC file may contain a number of object trees. `rsh_object` contains an offset from the beginning of the file to the object trees (stored consecutively). The **LONG** array pointed to by `rsh_trindex` can be used to separate the object trees in the list. There are `rsh_ntree` **LONGs** in this array. Each array entry can be used as an array index to a different object tree. After being loaded in memory by `rsrc_load()`, the members at `rsh_trindex` are filled in with the absolute pointers to their respective trees.

Each individual **OBJECT** is stored differently on disk then in memory. In the file, pointers to **TEDINFOs**, **BITBLKs**, and **ICONBLKs** are stored as absolute indexes into the arrays of these members stored in the file. Therefore a **G_TEXT OBJECT** whose `ob_spec` field would normally point a **TEDINFO** in memory would contain the value 0 if that **TEDINFO** were the first **TEDINFO** contained in the file.

String pointers are represented on disk by their absolute offset from the beginning of the file. Image pointers in **BITBLK** and **ICONBLK** structures are likewise pointed to through absolute file offsets, not indexes.
Free Strings and Images

`rsh_frstr` points to a table of `LONG`s which each specify an offset from the start of the RSC file to a free string. `rsh_frimg` points to a table of `LONG`s which each specify an offset from the start of the file to a `BITBLK` structure.

AES 3.30 Resource Format

Beginning with AES 3.30, the resource file format was altered to allow for new `OBJECT` types. The only `OBJECT` which currently takes advantage of this format is `G_CICON`. `G_CICON`s can only be stored in files of the new format. The new format can be identified by the third bit of `rsh_vrsn` being set.

The Extension Array

Immediately following the old resource data (using `rsh_rssize` as an offset) an extension array is added. The first entry in this array is a `LONG` containing the true size of the RSC file. Notice that values such as these are now stored as `LONG`s to allow the size of RSC files to exceed 64k. Due to the method in which some older resource elements were stored many components of RSC files will still be constrained to 64k.

Following the file size is a `LONG` word for each extension present followed by a 0L which terminates the array. Currently only one extension exists (`CICONBLK`) and it always occupies the first extension slot. As additional extensions are added, a value of -1L for any entry will indicate that there are no resource elements of that type in the file. For example an extension array that does contain `CICONBLK`s would look like this.

<table>
<thead>
<tr>
<th>...basic resource file...</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>LONG</code> filesize</td>
</tr>
<tr>
<td><code>LONG</code> cicon_offset</td>
</tr>
<tr>
<td>0L</td>
</tr>
</tbody>
</table>

The CICONBLK Extension

The `G_CICON` object type adds the ability to display color icons from the AES. The `ob_spec` of the object indexes a `CICONBLK` structure stored in the extension area. Each `CICONBLK` must contain a monochrome icon and a color icon for as many different resolutions as desired. When drawn, the AES will pick the icon that is the closest match for the current screen display. If there is no color icon present which the AES is able to convert, the monochrome icon is displayed.

The `cicon_offset` pointer gives an offset from the beginning of the resource file to a file segment which contains the `CICON` data. This segment contains a `CICONBLK` pointer table followed by the actual `CICONBLK`s.

The `CICONBLK` pointer table is simply a longword 0L for each `CICONBLK` present in the file. These pointers are filled in by the AES when loaded. The list is terminated by a -1L.
Immediately following the pointer table is one of the following variable length structures for each **CICONBLK**:

ICONBLK monoicon; /* This is the standard monochrome resource. */
LONG n_cicons; /* Number of ICONs of different resolutions. */
WORD mono_data[x]; /* Monochrome bitmap data. */
WORD mono_mask[x]; /* Monochrome bitmap mask. */
CHAR icon_text[12]; /* Icon text (maximum of 12 characters). */

/* for each resolution supported (n_cicons) include the following structure */

WORD num_planes; /* Number of planes this icon was intended for */
LONG col_data; /* Placeholder (calculated upon loading). */
LONG col_mask; /* Placeholder (calculated upon loading). */
LONG sel_data; /* Placeholder (must be non-zero if 'selected' data exists */
LONG sel_mask; /* Placeholder (calculated upon loading). */
LONG next_res; /* 1L = more icons follow */
WORD color_data[n]; /* n WORDs of image data (n is num_planes*WORDS in mono icon). */
WORD color_mask[n]; /* n WORDs of image mask. */
WORD select_data[n]; /* Only present if sel_data is non-zero. */
WORD select_mask[n]; /* Only present if sel_data is non-zero. */

**CICON Images**

All color image data is stored in **VDI** device independent format on disk and is automatically converted by **vr_trnfm()** upon **rsrc_load()**\(^1\).

---

\(^1\) Due to a bug in some versions of the **VDI** the seventh **WORD** of color icon image data may not contain the value 0x0001. If it does, the **VDI** may incorrectly display the icon.
GEMDOS/BIOS Errors

Upon return from most GEMDOS and BIOS functions, register D0 contains a longword error code describing the failure or success of an operation. The BIOS uses error codes -1 to -31 while GEMDOS uses error codes -32 and lower. A return value of 0 always indicates success. The error codes and their meanings are as follows:

<table>
<thead>
<tr>
<th>Name</th>
<th>BIOS #</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>E_OK</td>
<td>0</td>
<td>No error</td>
</tr>
<tr>
<td>ERROR</td>
<td>-1</td>
<td>Generic error</td>
</tr>
<tr>
<td>EDRVNR</td>
<td>-2</td>
<td>Drive not ready</td>
</tr>
<tr>
<td>EUNCMD</td>
<td>-3</td>
<td>Unknown command</td>
</tr>
<tr>
<td>E_CRC</td>
<td>-4</td>
<td>CRC error</td>
</tr>
<tr>
<td>EBADDRQ</td>
<td>-5</td>
<td>Bad request</td>
</tr>
<tr>
<td>E_SEEK</td>
<td>-6</td>
<td>Seek error</td>
</tr>
<tr>
<td>EMEDIA</td>
<td>-7</td>
<td>Unknown media</td>
</tr>
<tr>
<td>ESECNF</td>
<td>-8</td>
<td>Sector not found</td>
</tr>
<tr>
<td>EPAPER</td>
<td>-9</td>
<td>Out of paper</td>
</tr>
<tr>
<td>EWRITF</td>
<td>-10</td>
<td>Write fault</td>
</tr>
<tr>
<td>EREADF</td>
<td>-11</td>
<td>Read fault</td>
</tr>
<tr>
<td>EWRPRO</td>
<td>-12</td>
<td>Device is write protected</td>
</tr>
<tr>
<td>E_CHNG</td>
<td>-14</td>
<td>Media change detected</td>
</tr>
<tr>
<td>EUNDEV</td>
<td>-15</td>
<td>Unknown device</td>
</tr>
<tr>
<td>EBADSF</td>
<td>-16</td>
<td>Bad sectors on format</td>
</tr>
<tr>
<td>EOTHER</td>
<td>-17</td>
<td>Insert other disk (request)</td>
</tr>
<tr>
<td>EINVFN</td>
<td>-32</td>
<td>Invalid function</td>
</tr>
<tr>
<td>EFILNF</td>
<td>-33</td>
<td>File not found</td>
</tr>
<tr>
<td>EPTHNF</td>
<td>-34</td>
<td>Path not found</td>
</tr>
<tr>
<td>ENHNDL</td>
<td>-35</td>
<td>No more handles</td>
</tr>
<tr>
<td>EACCDN</td>
<td>-36</td>
<td>Access denied</td>
</tr>
<tr>
<td>EHNDL</td>
<td>-37</td>
<td>Invalid handle</td>
</tr>
<tr>
<td>ENSMEM</td>
<td>-39</td>
<td>Insufficient memory</td>
</tr>
<tr>
<td>EIMBA</td>
<td>-40</td>
<td>Invalid memory block address</td>
</tr>
<tr>
<td>EDRIVE</td>
<td>-46</td>
<td>Invalid drive specification</td>
</tr>
<tr>
<td>ENSAME</td>
<td>-48</td>
<td>Cross device rename</td>
</tr>
<tr>
<td>ENMFILE</td>
<td>-49</td>
<td>No more files</td>
</tr>
<tr>
<td>ELOCKED</td>
<td>-58</td>
<td>Record is already locked</td>
</tr>
<tr>
<td>ENSLOCK</td>
<td>-59</td>
<td>Invalid lock removal request</td>
</tr>
<tr>
<td>ERANGE</td>
<td>-64</td>
<td>Range error</td>
</tr>
<tr>
<td>ENAMETOOLONG</td>
<td>-64</td>
<td>Range error</td>
</tr>
<tr>
<td>EINTRN</td>
<td>-65</td>
<td>Internal error</td>
</tr>
<tr>
<td>EPLFMT</td>
<td>-66</td>
<td>Invalid program load format</td>
</tr>
<tr>
<td>EGSBF</td>
<td>-67</td>
<td>Memory block growth failure</td>
</tr>
<tr>
<td>ELOOP</td>
<td>-80</td>
<td>Too many symbolic links</td>
</tr>
<tr>
<td>EMOUNT</td>
<td>-200</td>
<td>Mount point crossed (indicator)</td>
</tr>
</tbody>
</table>
Atari ASCII Table

All Atari operating system calls use the Atari ASCII character set as the default method for encoding text strings. Strings encoded in this manner are composed of unsigned bytes representing a uniquely defined character as the following table specifies. Unless otherwise noted, a NULL character (ASCII 0) is used to indicate the end of string.
<table>
<thead>
<tr>
<th>Dec</th>
<th>Hex</th>
<th>Char</th>
<th>Dec</th>
<th>Hex</th>
<th>Char</th>
<th>Dec</th>
<th>Hex</th>
<th>Char</th>
</tr>
</thead>
<tbody>
<tr>
<td>102</td>
<td>0x66</td>
<td>f</td>
<td>143</td>
<td>0x8F</td>
<td>Æ</td>
<td>184</td>
<td>0xB8</td>
<td>Ö</td>
</tr>
<tr>
<td>103</td>
<td>0x67</td>
<td>g</td>
<td>144</td>
<td>0x90</td>
<td>E</td>
<td>185</td>
<td>0xB9</td>
<td>Ä</td>
</tr>
<tr>
<td>104</td>
<td>0x68</td>
<td>h</td>
<td>145</td>
<td>0x91</td>
<td>æ</td>
<td>186</td>
<td>0xBA</td>
<td>Æ</td>
</tr>
<tr>
<td>105</td>
<td>0x69</td>
<td>i</td>
<td>146</td>
<td>0x92</td>
<td>Æ</td>
<td>187</td>
<td>0xBB</td>
<td>Ö</td>
</tr>
<tr>
<td>106</td>
<td>0x6A</td>
<td>j</td>
<td>147</td>
<td>0x93</td>
<td>Æ</td>
<td>188</td>
<td>0xBC</td>
<td>Å</td>
</tr>
<tr>
<td>107</td>
<td>0x6B</td>
<td>k</td>
<td>148</td>
<td>0x94</td>
<td>Æ</td>
<td>189</td>
<td>0xBD</td>
<td>Ø</td>
</tr>
<tr>
<td>108</td>
<td>0x6C</td>
<td>l</td>
<td>149</td>
<td>0x95</td>
<td>Æ</td>
<td>190</td>
<td>0xBE</td>
<td>ø</td>
</tr>
<tr>
<td>109</td>
<td>0x6D</td>
<td>m</td>
<td>150</td>
<td>0x96</td>
<td>Æ</td>
<td>191</td>
<td>0xBF</td>
<td>ø</td>
</tr>
<tr>
<td>110</td>
<td>0x6E</td>
<td>n</td>
<td>151</td>
<td>0x97</td>
<td>Æ</td>
<td>192</td>
<td>0xC0</td>
<td>Ù</td>
</tr>
<tr>
<td>111</td>
<td>0x6F</td>
<td>o</td>
<td>152</td>
<td>0x98</td>
<td>Æ</td>
<td>193</td>
<td>0xC1</td>
<td>Ù</td>
</tr>
<tr>
<td>112</td>
<td>0x70</td>
<td>p</td>
<td>153</td>
<td>0x99</td>
<td>Æ</td>
<td>194</td>
<td>0xC2</td>
<td>Ù</td>
</tr>
<tr>
<td>113</td>
<td>0x71</td>
<td>q</td>
<td>154</td>
<td>0x9A</td>
<td>Æ</td>
<td>195</td>
<td>0xC3</td>
<td>Ù</td>
</tr>
<tr>
<td>114</td>
<td>0x72</td>
<td>r</td>
<td>155</td>
<td>0x9B</td>
<td>Æ</td>
<td>196</td>
<td>0xC4</td>
<td>Ù</td>
</tr>
<tr>
<td>115</td>
<td>0x73</td>
<td>s</td>
<td>156</td>
<td>0x9C</td>
<td>Æ</td>
<td>197</td>
<td>0xC5</td>
<td>Ù</td>
</tr>
<tr>
<td>116</td>
<td>0x74</td>
<td>t</td>
<td>157</td>
<td>0x9D</td>
<td>Æ</td>
<td>198</td>
<td>0xC6</td>
<td>Ù</td>
</tr>
<tr>
<td>117</td>
<td>0x75</td>
<td>u</td>
<td>158</td>
<td>0x9E</td>
<td>Æ</td>
<td>199</td>
<td>0xC7</td>
<td>Ù</td>
</tr>
<tr>
<td>118</td>
<td>0x76</td>
<td>v</td>
<td>159</td>
<td>0x9F</td>
<td>Æ</td>
<td>200</td>
<td>0xC8</td>
<td>Ù</td>
</tr>
<tr>
<td>119</td>
<td>0x77</td>
<td>w</td>
<td>160</td>
<td>0xA0</td>
<td>Æ</td>
<td>201</td>
<td>0xC9</td>
<td>Ù</td>
</tr>
<tr>
<td>120</td>
<td>0x78</td>
<td>x</td>
<td>161</td>
<td>0xA1</td>
<td>Æ</td>
<td>202</td>
<td>0xCA</td>
<td>Ù</td>
</tr>
<tr>
<td>121</td>
<td>0x79</td>
<td>y</td>
<td>162</td>
<td>0xA2</td>
<td>Æ</td>
<td>203</td>
<td>0xCB</td>
<td>Ù</td>
</tr>
<tr>
<td>122</td>
<td>0x7A</td>
<td>z</td>
<td>163</td>
<td>0xA3</td>
<td>Æ</td>
<td>204</td>
<td>0xCC</td>
<td>Ù</td>
</tr>
<tr>
<td>123</td>
<td>0x7B</td>
<td>{</td>
<td>164</td>
<td>0xA4</td>
<td>Æ</td>
<td>205</td>
<td>0xCD</td>
<td>Ù</td>
</tr>
<tr>
<td>124</td>
<td>0x7C</td>
<td>}</td>
<td>165</td>
<td>0xA5</td>
<td>Æ</td>
<td>206</td>
<td>0xCE</td>
<td>Ù</td>
</tr>
<tr>
<td>125</td>
<td>0x7D</td>
<td>~</td>
<td>166</td>
<td>0xA6</td>
<td>Æ</td>
<td>207</td>
<td>0xCF</td>
<td>Ù</td>
</tr>
<tr>
<td>126</td>
<td>0x7E</td>
<td>¬</td>
<td>167</td>
<td>0xA7</td>
<td>Æ</td>
<td>208</td>
<td>0xD0</td>
<td>Ù</td>
</tr>
<tr>
<td>127</td>
<td>0x7F</td>
<td>Ç</td>
<td>168</td>
<td>0xA8</td>
<td>Æ</td>
<td>209</td>
<td>0xD1</td>
<td>Ù</td>
</tr>
<tr>
<td>128</td>
<td>0x80</td>
<td>à</td>
<td>169</td>
<td>0xA9</td>
<td>Æ</td>
<td>210</td>
<td>0xD2</td>
<td>Ù</td>
</tr>
<tr>
<td>129</td>
<td>0x81</td>
<td>á</td>
<td>170</td>
<td>0xAA</td>
<td>Æ</td>
<td>211</td>
<td>0xD3</td>
<td>Ù</td>
</tr>
<tr>
<td>130</td>
<td>0x82</td>
<td>è</td>
<td>171</td>
<td>0xAB</td>
<td>Æ</td>
<td>212</td>
<td>0xD4</td>
<td>Ù</td>
</tr>
<tr>
<td>131</td>
<td>0x83</td>
<td>é</td>
<td>172</td>
<td>0xAC</td>
<td>Æ</td>
<td>213</td>
<td>0xD5</td>
<td>Ù</td>
</tr>
<tr>
<td>132</td>
<td>0x84</td>
<td>ë</td>
<td>173</td>
<td>0xAD</td>
<td>Æ</td>
<td>214</td>
<td>0xD6</td>
<td>Ù</td>
</tr>
<tr>
<td>133</td>
<td>0x85</td>
<td>í</td>
<td>174</td>
<td>0xAE</td>
<td>Æ</td>
<td>215</td>
<td>0xD7</td>
<td>Ù</td>
</tr>
<tr>
<td>134</td>
<td>0x86</td>
<td>í</td>
<td>175</td>
<td>0xAF</td>
<td>Æ</td>
<td>216</td>
<td>0xD8</td>
<td>Ù</td>
</tr>
<tr>
<td>135</td>
<td>0x87</td>
<td>è</td>
<td>176</td>
<td>0xB0</td>
<td>Æ</td>
<td>217</td>
<td>0xD9</td>
<td>Ù</td>
</tr>
<tr>
<td>136</td>
<td>0x88</td>
<td>´</td>
<td>177</td>
<td>0xB1</td>
<td>Æ</td>
<td>218</td>
<td>0xDA</td>
<td>Ù</td>
</tr>
<tr>
<td>137</td>
<td>0x89</td>
<td>©</td>
<td>178</td>
<td>0xB2</td>
<td>Æ</td>
<td>219</td>
<td>0xDB</td>
<td>Ù</td>
</tr>
<tr>
<td>138</td>
<td>0x8A</td>
<td>ä</td>
<td>179</td>
<td>0xB3</td>
<td>Æ</td>
<td>220</td>
<td>0xDC</td>
<td>Ù</td>
</tr>
<tr>
<td>139</td>
<td>0x8B</td>
<td>ä</td>
<td>180</td>
<td>0xB4</td>
<td>Æ</td>
<td>221</td>
<td>0xDD</td>
<td>Ù</td>
</tr>
<tr>
<td>140</td>
<td>0x8C</td>
<td>ä</td>
<td>181</td>
<td>0xB5</td>
<td>Æ</td>
<td></td>
<td></td>
<td>Ù</td>
</tr>
<tr>
<td>141</td>
<td>0x8D</td>
<td>ä</td>
<td>182</td>
<td>0xB6</td>
<td>Æ</td>
<td></td>
<td></td>
<td>Ù</td>
</tr>
<tr>
<td>142</td>
<td>0x8E</td>
<td>ä</td>
<td>183</td>
<td>0xB7</td>
<td>Æ</td>
<td></td>
<td></td>
<td>Ù</td>
</tr>
<tr>
<td>Dec</td>
<td>Hex</td>
<td>Char</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>-----</td>
<td>------</td>
<td>------</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>222</td>
<td>0xDE</td>
<td>🇪‍🇺‍рактион</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>223</td>
<td>0xDF</td>
<td>🇪‍🇺‍icamente</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>224</td>
<td>0xE0</td>
<td>🇪‍🇺‍α</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>225</td>
<td>0xE1</td>
<td>🇪‍🇺‍β</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>226</td>
<td>0xE2</td>
<td>🇪‍🇺‍Г</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>227</td>
<td>0xE3</td>
<td>🇪‍🇺‍τ</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>228</td>
<td>0xE4</td>
<td>🇪‍🇺‍ν</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>229</td>
<td>0xE5</td>
<td>🇪‍🇺‍ο</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>230</td>
<td>0xE6</td>
<td>🇪‍🇺‍μ</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>231</td>
<td>0xE7</td>
<td>🇪‍🇺‍τ</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>232</td>
<td>0xE8</td>
<td>🇪‍🇺‍θ</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>233</td>
<td>0xE9</td>
<td>🇪‍🇺‍Θ</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>234</td>
<td>0xEA</td>
<td>🇪‍🇺‍Ω</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>235</td>
<td>0xEB</td>
<td>🇪‍🇺‍Φ</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>236</td>
<td>0xEC</td>
<td>🇪‍🇺‍Φ</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>237</td>
<td>0xED</td>
<td>🇪‍🇺‍Φ</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>238</td>
<td>0xEE</td>
<td>🇪‍🇺‍Ε</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>239</td>
<td>0xEF</td>
<td>🇪‍🇺‍Ε</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>240</td>
<td>0xF0</td>
<td>🇪‍🇺‍Ⅰ</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>241</td>
<td>0xF1</td>
<td>🇪‍🇺‍Ⅰ</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>242</td>
<td>0xF2</td>
<td>🇪‍🇺‍Ⅰ</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>243</td>
<td>0xF3</td>
<td>🇪‍🇺‍Ⅰ</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>244</td>
<td>0xF4</td>
<td>🇪‍🇺‍Ⅰ</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>245</td>
<td>0xF5</td>
<td>🇪‍🇺‍Ⅰ</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>246</td>
<td>0xF6</td>
<td>🇪‍🇺‍Ⅰ</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>247</td>
<td>0xF7</td>
<td>🇪‍🇺‍Ⅰ</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>248</td>
<td>0xF8</td>
<td>🇪‍🇺‍Ⅰ</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>249</td>
<td>0xF9</td>
<td>🇪‍🇺‍Ⅰ</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>250</td>
<td>0xFA</td>
<td>🇪‍🇺‍Ⅰ</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>251</td>
<td>0xFB</td>
<td>🇪‍🇺‍Ⅰ</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>252</td>
<td>0xFC</td>
<td>🇪‍🇺‍Ⅰ</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>253</td>
<td>0xFD</td>
<td>🇪‍🇺‍Ⅰ</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>254</td>
<td>0xFE</td>
<td>🇪‍🇺‍Ⅰ</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>255</td>
<td>0xFF</td>
<td>🇪‍🇺‍Ⅰ</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
IKBD Scan Codes

The AES, VDI, and BIOS, all contain functions which return scan codes from the Intelligent Keyboard Controller (IKBD). These scan codes can be used to determine exactly which key was struck (not simply the ASCII value).

One thing that must be considered when relying on scan codes is that they identify a physical vector on the keyboard, not a key definition. The scancode for a letter on an American keyboard, for instance, may be different than the scancode for the same letter on a German keyboard. The XBIOS function Keytbl() can be used to look up the ASCII value assigned to a scancode to ensure that keystrokes are correctly processed.

Scancodes for keyboard modifiers (SHIFT, ALT, etc.) are never returned by an OS call. However, when handling the IKBD directly, the following scancodes may be encountered:

<table>
<thead>
<tr>
<th>Key</th>
<th>Scancode</th>
</tr>
</thead>
<tbody>
<tr>
<td>Left-Shift</td>
<td>42 (0x2A)</td>
</tr>
<tr>
<td>Right-Shift</td>
<td>54 (0x36)</td>
</tr>
<tr>
<td>Control</td>
<td>29 (0x1D)</td>
</tr>
<tr>
<td>Alternate</td>
<td>56 (0x38)</td>
</tr>
<tr>
<td>Caps Lock</td>
<td>58 (0x3A)</td>
</tr>
</tbody>
</table>

The values shown in the following table contain the IKBD scancode of each keyboard key in the high BYTE and the ASCII code in the low BYTE. Keys with no corresponding ASCII value will always have zero as the low byte. These values are valid for all Atari computers with US keyboards:

<table>
<thead>
<tr>
<th>Key</th>
<th>Unshifted</th>
<th>Key</th>
<th>Shifted</th>
<th>w/CTRL</th>
<th>w/ALT</th>
</tr>
</thead>
<tbody>
<tr>
<td>a</td>
<td>0x1E61</td>
<td>A</td>
<td>0x1E41</td>
<td>0x1E01</td>
<td>0x1E00</td>
</tr>
<tr>
<td>b</td>
<td>0x3062</td>
<td>B</td>
<td>0x3042</td>
<td>0x3002</td>
<td>0x3000</td>
</tr>
<tr>
<td>c</td>
<td>0x2E63</td>
<td>C</td>
<td>0x2E43</td>
<td>0x2E03</td>
<td>0x2E00</td>
</tr>
<tr>
<td>d</td>
<td>0x2064</td>
<td>D</td>
<td>0x2044</td>
<td>0x2004</td>
<td>0x2000</td>
</tr>
<tr>
<td>e</td>
<td>0x1265</td>
<td>E</td>
<td>0x1245</td>
<td>0x1205</td>
<td>0x1200</td>
</tr>
<tr>
<td>f</td>
<td>0x2166</td>
<td>F</td>
<td>0x2146</td>
<td>0x2106</td>
<td>0x2100</td>
</tr>
<tr>
<td>g</td>
<td>0x2267</td>
<td>G</td>
<td>0x2247</td>
<td>0x2207</td>
<td>0x2200</td>
</tr>
<tr>
<td>h</td>
<td>0x2368</td>
<td>H</td>
<td>0x2348</td>
<td>0x2308</td>
<td>0x2300</td>
</tr>
<tr>
<td>i</td>
<td>0x1769</td>
<td>I</td>
<td>0x1749</td>
<td>0x1709</td>
<td>0x1700</td>
</tr>
<tr>
<td>j</td>
<td>0x246A</td>
<td>J</td>
<td>0x244A</td>
<td>0x240A</td>
<td>0x2400</td>
</tr>
<tr>
<td>k</td>
<td>0x256B</td>
<td>K</td>
<td>0x254B</td>
<td>0x250B</td>
<td>0x2500</td>
</tr>
<tr>
<td>l</td>
<td>0x266C</td>
<td>L</td>
<td>0x264C</td>
<td>0x260C</td>
<td>0x2600</td>
</tr>
<tr>
<td>m</td>
<td>0x326D</td>
<td>M</td>
<td>0x324D</td>
<td>0x320D</td>
<td>0x3200</td>
</tr>
<tr>
<td>n</td>
<td>0x316E</td>
<td>N</td>
<td>0x314E</td>
<td>0x310E</td>
<td>0x3100</td>
</tr>
<tr>
<td>o</td>
<td>0x186F</td>
<td>O</td>
<td>0x184F</td>
<td>0x180F</td>
<td>0x1800</td>
</tr>
<tr>
<td>p</td>
<td>0x1970</td>
<td>P</td>
<td>0x1950</td>
<td>0x1910</td>
<td>0x1900</td>
</tr>
<tr>
<td>q</td>
<td>0x1071</td>
<td>Q</td>
<td>0x1051</td>
<td>0x1011</td>
<td>0x1000</td>
</tr>
<tr>
<td>r</td>
<td>0x1372</td>
<td>R</td>
<td>0x1352</td>
<td>0x1312</td>
<td>0x1300</td>
</tr>
<tr>
<td>s</td>
<td>0x1F73</td>
<td>S</td>
<td>0x1F53</td>
<td>0x1F13</td>
<td>0x1F00</td>
</tr>
<tr>
<td>t</td>
<td>0x1474</td>
<td>T</td>
<td>0x1454</td>
<td>0x1414</td>
<td>0x1400</td>
</tr>
</tbody>
</table>
### F.4 – IKBD Scan Codes

<table>
<thead>
<tr>
<th>Key Unshifted</th>
<th>Key Shifted</th>
<th>w/CTRL</th>
<th>w/ALT</th>
</tr>
</thead>
<tbody>
<tr>
<td>u 0x1675</td>
<td>U 0x1655</td>
<td>0x1615</td>
<td>0x1600</td>
</tr>
<tr>
<td>v 0x2F76</td>
<td>V 0x2F56</td>
<td>0x2F16</td>
<td>0x2F00</td>
</tr>
<tr>
<td>w 0x1177</td>
<td>W 0x1157</td>
<td>0x1117</td>
<td>0x1100</td>
</tr>
<tr>
<td>x 0x2D78</td>
<td>X 0x2D58</td>
<td>0x2D18</td>
<td>0x2D00</td>
</tr>
<tr>
<td>y 0x1579</td>
<td>Y 0x1559</td>
<td>0x1519</td>
<td>0x1500</td>
</tr>
<tr>
<td>z 0x2C7A</td>
<td>Z 0x2C5A</td>
<td>0x2C1A</td>
<td>0x2C00</td>
</tr>
<tr>
<td>1 0x0231</td>
<td>! 0x0221</td>
<td>0x0211</td>
<td>0x7800</td>
</tr>
<tr>
<td>2 0x0332</td>
<td>@ 0x0340</td>
<td>0x0300</td>
<td>0x7900</td>
</tr>
<tr>
<td>3 0x0433</td>
<td># 0x0423</td>
<td>0x0413</td>
<td>0x7A00</td>
</tr>
<tr>
<td>4 0x0534</td>
<td>$ 0x0524</td>
<td>0x0514</td>
<td>0x7B00</td>
</tr>
<tr>
<td>5 0x0635</td>
<td>% 0x0625</td>
<td>0x0615</td>
<td>0x7C00</td>
</tr>
<tr>
<td>6 0x0736</td>
<td>^ 0x075E</td>
<td>0x071E</td>
<td>0x7D00</td>
</tr>
<tr>
<td>7 0x0837</td>
<td>&amp; 0x0826</td>
<td>0x0817</td>
<td>0x7E00</td>
</tr>
<tr>
<td>8 0x0938</td>
<td>* 0x092A</td>
<td>0x0918</td>
<td>0x7F00</td>
</tr>
<tr>
<td>9 0x0A39</td>
<td>( 0x0A28</td>
<td>0x0A19</td>
<td>0x8000</td>
</tr>
<tr>
<td>0 0x0B30</td>
<td>) 0x0B29</td>
<td>0x0B10</td>
<td>0x8100</td>
</tr>
<tr>
<td>- 0x0C2D</td>
<td>_ 0x0C5F</td>
<td>0x0C1F</td>
<td>0x8200</td>
</tr>
<tr>
<td>= 0x0D3D</td>
<td>+ 0x0D2B</td>
<td>0x0D1D</td>
<td>0x8300</td>
</tr>
<tr>
<td>` 0x2960</td>
<td>~ 0x297E</td>
<td>0x2900</td>
<td>0x2960</td>
</tr>
<tr>
<td>\ 0x2B5C</td>
<td></td>
<td>0x2B7C</td>
<td>0x2B1C</td>
</tr>
<tr>
<td>[ 0x1A5B</td>
<td>{ 0x1A7B</td>
<td>0x1A1B</td>
<td>0x1A5B</td>
</tr>
<tr>
<td>] 0x1B5D</td>
<td>} 0x1B7D</td>
<td>0x1B1D</td>
<td>0x1B5D</td>
</tr>
<tr>
<td>; 0x273B</td>
<td>: 0x273A</td>
<td>0x271B</td>
<td>0x273B</td>
</tr>
<tr>
<td>' 0x2827</td>
<td>&quot; 0x2822</td>
<td>0x2807</td>
<td>0x2827</td>
</tr>
<tr>
<td>, 0x332C</td>
<td>&lt; 0x333C</td>
<td>0x330C</td>
<td>0x332C</td>
</tr>
<tr>
<td>. 0x342E</td>
<td>&gt; 0x343E</td>
<td>0x340E</td>
<td>0x342E</td>
</tr>
<tr>
<td>/ 0x352F</td>
<td>? 0x353F</td>
<td>0x250F</td>
<td>0x352E</td>
</tr>
<tr>
<td>SPACE 0x3920</td>
<td>0x3900</td>
<td>0x3920</td>
<td></td>
</tr>
<tr>
<td>ESC 0x011B</td>
<td>0x011B</td>
<td>0x011B</td>
<td></td>
</tr>
<tr>
<td>BKSP 0x0E08</td>
<td>0x0E08</td>
<td>0x0E08</td>
<td></td>
</tr>
<tr>
<td>DEL 0x537F</td>
<td>0x531F</td>
<td>0x537F</td>
<td></td>
</tr>
<tr>
<td>RETURN 0x100D</td>
<td>0x1009</td>
<td>0x100A</td>
<td></td>
</tr>
<tr>
<td>TAB 0x0F09</td>
<td>0x0F09</td>
<td>0x0F09</td>
<td></td>
</tr>
<tr>
<td>Nmpad (</td>
<td>0x6328</td>
<td>0x6308</td>
<td></td>
</tr>
<tr>
<td>Nmpad )</td>
<td>0x6429</td>
<td>0x6408</td>
<td></td>
</tr>
<tr>
<td>Nmpad /</td>
<td>0x652F</td>
<td>0x650F</td>
<td></td>
</tr>
<tr>
<td>Nmpad *</td>
<td>0x662A</td>
<td>0x660A</td>
<td></td>
</tr>
<tr>
<td>Nmpad _</td>
<td>0x4A2D</td>
<td>0x4A1F</td>
<td></td>
</tr>
<tr>
<td>Nmpad +</td>
<td>0x4E2B</td>
<td>0x4E0B</td>
<td></td>
</tr>
<tr>
<td>Nmpad .</td>
<td>0x712E</td>
<td>0x710E</td>
<td></td>
</tr>
<tr>
<td>Nmpad ENTER</td>
<td>0x720D</td>
<td>0x720A</td>
<td></td>
</tr>
<tr>
<td>Nmpad 0</td>
<td>0x7030</td>
<td>0x7010</td>
<td></td>
</tr>
<tr>
<td>Nmpad 1</td>
<td>0x6D31</td>
<td>0x6D11</td>
<td></td>
</tr>
<tr>
<td>Nmpad 2</td>
<td>0x6E32</td>
<td>0x6E00</td>
<td></td>
</tr>
<tr>
<td>Nmpad 3</td>
<td>0x6F33</td>
<td>0x6F13</td>
<td></td>
</tr>
<tr>
<td>Nmpad 4</td>
<td>0x6A34</td>
<td>0x6A14</td>
<td></td>
</tr>
<tr>
<td>Nmpad 5</td>
<td>0x6B35</td>
<td>0x6B15</td>
<td></td>
</tr>
<tr>
<td>Nmpad 6</td>
<td>0x6C36</td>
<td>0x6C1E</td>
<td></td>
</tr>
</tbody>
</table>

1Atari computers with TOS 2.0 or higher do not generate scan codes for the ALT-Numeric Keypad numbers. Instead they allow the user to enter any key by holding ALT while typing the ASCII code number and then releasing ALT to generate the keypress.

**The Atari Compendium**
<table>
<thead>
<tr>
<th>Key</th>
<th>Unshifted</th>
<th>Key</th>
<th>Shifted</th>
<th>w/CTRL</th>
<th>w/ALT</th>
</tr>
</thead>
<tbody>
<tr>
<td>Nmpad 7</td>
<td>0x6737</td>
<td>0x6737</td>
<td>0x6717</td>
<td>0x6737</td>
<td></td>
</tr>
<tr>
<td>Nmpad 8</td>
<td>0x6838</td>
<td>0x6838</td>
<td>0x6818</td>
<td>0x6838</td>
<td></td>
</tr>
<tr>
<td>Nmpad 9</td>
<td>0x6939</td>
<td>0x6939</td>
<td>0x6919</td>
<td>0x6939</td>
<td></td>
</tr>
<tr>
<td>HELP</td>
<td>0x6200</td>
<td>0x6200</td>
<td>0x6200</td>
<td></td>
<td>Alt-Help</td>
</tr>
<tr>
<td>UNDO</td>
<td>0x6100</td>
<td>0x6100</td>
<td>0x6100</td>
<td>0x6100</td>
<td></td>
</tr>
<tr>
<td>INSERT</td>
<td>0x5200</td>
<td>0x5230</td>
<td>0x5200</td>
<td></td>
<td>Left Mouse Button</td>
</tr>
<tr>
<td>CLR/HOME</td>
<td>0x4700</td>
<td>0x4737</td>
<td>0x7700</td>
<td></td>
<td>Right Mouse Button</td>
</tr>
<tr>
<td>UP-ARROW</td>
<td>0x4800</td>
<td>0x4838</td>
<td>0x4800</td>
<td></td>
<td>Mouse Up</td>
</tr>
<tr>
<td>DOWN-ARROW</td>
<td>0x5000</td>
<td>0x5032</td>
<td>0x5000</td>
<td></td>
<td>Mouse Down</td>
</tr>
<tr>
<td>LEFT-ARROW</td>
<td>0x4B00</td>
<td>0x4B34</td>
<td>0x7300</td>
<td></td>
<td>Mouse Left</td>
</tr>
<tr>
<td>RIGHT-ARROW</td>
<td>0x4D00</td>
<td>0x4D36</td>
<td>0x7400</td>
<td></td>
<td>Mouse Right</td>
</tr>
<tr>
<td>F1</td>
<td>0x3B00</td>
<td>0x5400</td>
<td>0x3B00</td>
<td>0x3B00</td>
<td></td>
</tr>
<tr>
<td>F2</td>
<td>0x3C00</td>
<td>0x5500</td>
<td>0x3C00</td>
<td>0x3C00</td>
<td></td>
</tr>
<tr>
<td>F3</td>
<td>0x3D00</td>
<td>0x5600</td>
<td>0x3D00</td>
<td>0x3D00</td>
<td></td>
</tr>
<tr>
<td>F4</td>
<td>0x3E00</td>
<td>0x5700</td>
<td>0x3E00</td>
<td>0x3E00</td>
<td></td>
</tr>
<tr>
<td>F5</td>
<td>0x3F00</td>
<td>0x5800</td>
<td>0x3F00</td>
<td>0x3F00</td>
<td></td>
</tr>
<tr>
<td>F6</td>
<td>0x4000</td>
<td>0x5900</td>
<td>0x4000</td>
<td>0x4000</td>
<td></td>
</tr>
<tr>
<td>F7</td>
<td>0x4100</td>
<td>0x5A00</td>
<td>0x4100</td>
<td>0x4100</td>
<td></td>
</tr>
<tr>
<td>F8</td>
<td>0x4200</td>
<td>0x5B00</td>
<td>0x4200</td>
<td>0x4200</td>
<td></td>
</tr>
<tr>
<td>F9</td>
<td>0x4300</td>
<td>0x5C00</td>
<td>0x4300</td>
<td>0x4300</td>
<td></td>
</tr>
<tr>
<td>F10</td>
<td>0x4400</td>
<td>0x5D00</td>
<td>0x4400</td>
<td>0x4400</td>
<td></td>
</tr>
</tbody>
</table>

2 This key does not generate a keycode, rather it triggers the screen dump interrupt.
3 Keycodes marked by an asterisk are mouse-equivalent keys and generate mouse events rather than keycodes.
The Speedo Font Header

This section provides detailed information about the contents of the buffer returned by the `vqt_fontheader()` call. First, here are some general notes about the values you will be using:

Character strings are only NULL terminated if they do not completely fill their assigned field.

All integers are signed (unless otherwise noted) and in Big-Endian format (most significant byte first).

Outline Resolution Units (ORUs) are the basic unit of measurement for Speedo characters. There are usually 1000 ORUs per Em square (width of the letter 'M') though you can check this value in the font header itself.

6-byte Transformation Parameters consist of a WORD Y offset (expressed in ORUs) followed by a UWORD X-scaling factor (expressed in units of 1/4096) and a similar UWORD Y-scaling factor (also expressed in units of 1/4096).

The following table details the information returned by the `vqt_fontheader()` function call:

<table>
<thead>
<tr>
<th>Offset</th>
<th>Field</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>Format Identifier</td>
<td>An 8-byte character string consisting of &quot;D1.0&quot; CR LF NULL NULL</td>
</tr>
<tr>
<td>8</td>
<td>Font Size</td>
<td>A LONG specifying the size of the font file in bytes.</td>
</tr>
<tr>
<td>12</td>
<td>Minimum Font Buffer Size</td>
<td>A LONG specifying the minimum size buffer required to load the non-image data of the font.</td>
</tr>
<tr>
<td>16</td>
<td>Minimum Character Buffer Size</td>
<td>A WORD specifying the minimum size buffer required to hold the largest character in the font.</td>
</tr>
<tr>
<td>18</td>
<td>Header Size</td>
<td>A WORD specifying the size of the font header.</td>
</tr>
<tr>
<td>20</td>
<td>Font ID</td>
<td>A WORD containing the Bitstream font ID number.</td>
</tr>
<tr>
<td>22</td>
<td>Font Version Number</td>
<td>A WORD containing the font revision number.</td>
</tr>
<tr>
<td>24</td>
<td>Font Full Name</td>
<td>A 70-byte character string containing the full name of the font.</td>
</tr>
<tr>
<td>94</td>
<td>Manufacturing Date</td>
<td>A 10-byte character string containing the manufacturing date of the font as DD Mon YY.</td>
</tr>
<tr>
<td>104</td>
<td>Character Set Name</td>
<td>A 66-byte character string containing the name of the character set used for the font (ex: &quot;Bitstream International Character Set&quot;).</td>
</tr>
</tbody>
</table>
Vendor ID

A 2-byte character string identifying the manufacturer of the font. This is usually the first two characters in the font filename. Bitstream fonts use 'BX'.

Character Set ID

A 2-byte character string identifying the character set used for this font. This is usually the second 2 characters in the font filename. The Bitstream International Character Set is '00'.

Copyright Notice

A 78-byte character string containing the copyright notice for the font.

Number of Character Indexes in Character Set

A WORD specifying the number of character indexes available in the font's character set. This does not necessarily mean that every index is actually used.

Total Number of Character Indexes in Font

A WORD indicating the number of character indexes available in the font's character set in addition to any supplementary characters needed to create compound characters.

Index of First Character

A WORD containing the first available character in a font.

Number of Kerning Tracks

A WORD specifying the total number of kerning tracks.

Number of Kerning Pairs

A WORD specifying the total number of kerning pairs.

Font Flags

Bit 0 of this BYTE is set to indicate extended mode. Extended mode fonts require a higher quality of font rendering (such as chess pieces). If Bit 0 is clear, the font is in Compact mode (the default). Bits 1–7 are currently reserved.

Classification Flags

A BYTE value whose bits indicate the font classification as follows: Bit Meaning

0 Italic

1 Monospace

2 Serif

3 Display

4–7 Reserved

Family Classification

A BYTE indicating the family classification of the font as follows: Value Meaning

0 Don't Care

1 Serif

2 Sans Serif

3 Monospace

4 Script

5 Decorative

Font Form Classification

A BYTE classifying the width and weight of characters in the font as follows: Bits 0–3 Meaning

0–3 (Reserved)

4 Condensed

5 (Reserved for 34 condensed)

6 Semi-Condensed

7 (Reserved for 14 condensed)

8 Normal
9 (Reserved for 34 expanded)
10 Semi-Expanded
11 (Reserved for 14 expanded)
12 Expanded
13–15 (Reserved)
Bits 4–7 Meaning
0 (Reserved)
1 Thin
2 Ultralight
3 Extralight
4 Light
5 Book
6 Normal
7 Medium
8 Semibold
9 Demibold
10 Bold
11 Extrabold
12 Ultrabold
13 Heavy
14 Black
15 (Reserved)

266 Short Font Name A 32-byte character string containing the abbreviation of the name of the Postscript compatible font.
298 Short Face Name A 16-byte character string containing the abbreviation of the typeface family name.
314 Font Form A 14-byte character string containing the font form classification (as above).
328 Italic Angle A WORD indicating the number of 1/256 degrees that characters are slanted clockwise.
330 ORUs per Em A WORD indicating the number of Outline Resolution Units (ORUs) per Em.
332 Width of Word Space A WORD value which expresses the width of a 'word space' (i.e. ASCII 32) in ORUs.
334 Width of Em Space A WORD value which expresses the width of Em space in ORUs (this is not always the same as the number of ORUs in the letter 'M').
336 Width of En Space A WORD value which expresses the width of En space in ORUs. This is always half the width of Em space (not the width of the letter 'N').
338 Width of Thin Space A WORD value which expresses the width of 'thin space' in ORUs. This is the width applied between two words and is normally the same as 'word space'. 
Width of Figure Space: A WORD value which expresses the width of 'figure space' in ORUs. This is the width of tabular characters in the font.

XMIN (Min X coordinate in font): A WORD indicating the minimum X coordinate used in the font.

YMIN (Min Y coordinate in font): A WORD indicating the minimum Y coordinate used in the font.

XMAX (Max X coordinate in font): A WORD indicating the maximum X coordinate used in the font.

YMAX (Max Y coordinate in font): A WORD indicating the maximum Y coordinate used in the font.

Underline Position: A WORD value indicating the distance the center of an underline should be applied from the baseline of the font.

Underline Thickness: A WORD value indicating the thickness an underline applied to this font should be (in ORUs).

Small Caps: A 6-byte Transformation Parameter used for small capitals (eg: abcdeff).

Display Superiors: A 6-byte Transformation Parameter used for display superiors (eg: $350).

Footnote Superiors: A 6-byte Transformation Parameter used for footnote superiors (eg: see footnote1).

Alpha Superiors: A 6-byte Transformation Parameter used for alpha superiors (eg: Sra).

Chemical Inferiors: A 6-byte Transformation Parameter used for chemical inferiors (eg: H20).

Small Numerators: A 6-byte Transformation Parameter used for small numerators (eg: ).

Small Denominators: A 6-byte Transformation Parameter used for small denominators (see above).

Medium Numerators: A 6-byte Transformation Parameter used for medium numerators (eg: ).

Medium Denominators: A 6-byte Transformation Parameter used for medium denominators (see above).

Large Numerators: A 6-byte Transformation Parameter used for large numerators (eg: ).

Large Denominators: A 6-byte Transformation Parameter used for large denominators (see above).
The Bitstream International Character Set

The `vst_charmap()` and `vqt_get_table()` functions provide access to the entire Speedo character set by specifying characters as **WORD** size Bitstream index values rather than **BYTE** size ASCII values. The following table lists the available Bitstream Speedo index and ID numbers.

All current Atari calls refer to Bitstream indexes rather than character ID. There is an important difference between these two. Characters never change ID numbers between fonts, however they may change index positions. When specifying character indexes with Atari calls it is important to note which character set the font was created with to provide accurate mapping. The following table lists indexes for the most common set, the Bitstream International Character Set represented in the typeface ‘Swiss 721’.

<table>
<thead>
<tr>
<th>IDX</th>
<th>ID</th>
<th>Char</th>
<th>IDX</th>
<th>ID</th>
<th>Char</th>
<th>IDX</th>
<th>ID</th>
<th>Char</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>32</td>
<td></td>
<td>5</td>
<td>37</td>
<td>%</td>
<td>10</td>
<td>42</td>
<td>*</td>
</tr>
<tr>
<td>1</td>
<td>33</td>
<td>!</td>
<td>6</td>
<td>38</td>
<td>&amp;</td>
<td>11</td>
<td>43</td>
<td>+</td>
</tr>
<tr>
<td>2</td>
<td>34</td>
<td>II</td>
<td>7</td>
<td>39</td>
<td>,</td>
<td>12</td>
<td>44</td>
<td>,</td>
</tr>
<tr>
<td>3</td>
<td>35</td>
<td>#</td>
<td>8</td>
<td>40</td>
<td>(</td>
<td>13</td>
<td>45</td>
<td>-</td>
</tr>
<tr>
<td>4</td>
<td>36</td>
<td>$</td>
<td>9</td>
<td>41</td>
<td>)</td>
<td>14</td>
<td>46</td>
<td></td>
</tr>
<tr>
<td>IDX</td>
<td>ID</td>
<td>Char</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>-----</td>
<td>----</td>
<td>------</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>15</td>
<td>47</td>
<td>/</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>16</td>
<td>48</td>
<td>0</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>17</td>
<td>49</td>
<td>1</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>18</td>
<td>50</td>
<td>2</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>19</td>
<td>51</td>
<td>3</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>20</td>
<td>52</td>
<td>4</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>21</td>
<td>53</td>
<td>5</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>22</td>
<td>54</td>
<td>6</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>23</td>
<td>55</td>
<td>7</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>24</td>
<td>56</td>
<td>8</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>25</td>
<td>57</td>
<td>9</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>26</td>
<td>58</td>
<td>:</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>27</td>
<td>59</td>
<td>;</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>28</td>
<td>60</td>
<td>&lt;</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>29</td>
<td>61</td>
<td>=</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>30</td>
<td>62</td>
<td>&gt;</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>31</td>
<td>63</td>
<td>?</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>32</td>
<td>64</td>
<td>@</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>33</td>
<td>65</td>
<td>A</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>34</td>
<td>66</td>
<td>B</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>35</td>
<td>67</td>
<td>C</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>36</td>
<td>68</td>
<td>D</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>37</td>
<td>69</td>
<td>E</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>38</td>
<td>70</td>
<td>F</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

**The Atari Compendium**
<table>
<thead>
<tr>
<th>IDX</th>
<th>ID</th>
<th>Char</th>
</tr>
</thead>
<tbody>
<tr>
<td>39</td>
<td>71</td>
<td>G</td>
</tr>
<tr>
<td>40</td>
<td>72</td>
<td>H</td>
</tr>
<tr>
<td>41</td>
<td>73</td>
<td>I</td>
</tr>
<tr>
<td>42</td>
<td>74</td>
<td>J</td>
</tr>
<tr>
<td>43</td>
<td>75</td>
<td>K</td>
</tr>
<tr>
<td>44</td>
<td>76</td>
<td>L</td>
</tr>
<tr>
<td>45</td>
<td>77</td>
<td>M</td>
</tr>
<tr>
<td>46</td>
<td>78</td>
<td>N</td>
</tr>
<tr>
<td>47</td>
<td>79</td>
<td>O</td>
</tr>
<tr>
<td>48</td>
<td>80</td>
<td>P</td>
</tr>
<tr>
<td>49</td>
<td>81</td>
<td>Q</td>
</tr>
<tr>
<td>50</td>
<td>82</td>
<td>R</td>
</tr>
<tr>
<td>51</td>
<td>83</td>
<td>S</td>
</tr>
<tr>
<td>52</td>
<td>84</td>
<td>T</td>
</tr>
<tr>
<td>53</td>
<td>85</td>
<td>U</td>
</tr>
<tr>
<td>54</td>
<td>86</td>
<td>V</td>
</tr>
<tr>
<td>55</td>
<td>87</td>
<td>W</td>
</tr>
<tr>
<td>56</td>
<td>88</td>
<td>X</td>
</tr>
<tr>
<td>57</td>
<td>89</td>
<td>Y</td>
</tr>
<tr>
<td>58</td>
<td>90</td>
<td>Z</td>
</tr>
<tr>
<td>59</td>
<td>91</td>
<td>[</td>
</tr>
<tr>
<td>60</td>
<td>92</td>
<td>\</td>
</tr>
<tr>
<td>61</td>
<td>93</td>
<td>]</td>
</tr>
<tr>
<td>62</td>
<td>94</td>
<td>^</td>
</tr>
<tr>
<td>IDX</td>
<td>ID</td>
<td>Char</td>
</tr>
<tr>
<td>-----</td>
<td>----</td>
<td>------</td>
</tr>
<tr>
<td>87</td>
<td>119</td>
<td>W</td>
</tr>
<tr>
<td>88</td>
<td>120</td>
<td>X</td>
</tr>
<tr>
<td>89</td>
<td>121</td>
<td>Y</td>
</tr>
<tr>
<td>90</td>
<td>122</td>
<td>Z</td>
</tr>
<tr>
<td>91</td>
<td>123</td>
<td>{</td>
</tr>
<tr>
<td>92</td>
<td>124</td>
<td></td>
</tr>
<tr>
<td>93</td>
<td>125</td>
<td>}</td>
</tr>
<tr>
<td>94</td>
<td>126</td>
<td>~</td>
</tr>
<tr>
<td>95</td>
<td>129</td>
<td>fi</td>
</tr>
<tr>
<td>96</td>
<td>130</td>
<td>fl</td>
</tr>
<tr>
<td>97</td>
<td>131</td>
<td>£</td>
</tr>
<tr>
<td>98</td>
<td>132</td>
<td>©</td>
</tr>
<tr>
<td>99</td>
<td>133</td>
<td>f</td>
</tr>
<tr>
<td>100</td>
<td>134</td>
<td>%</td>
</tr>
<tr>
<td>101</td>
<td>135</td>
<td>.</td>
</tr>
</tbody>
</table>

"The Atari Compendium"
<table>
<thead>
<tr>
<th>IDX</th>
<th>ID</th>
<th>Char</th>
</tr>
</thead>
<tbody>
<tr>
<td>111</td>
<td>145</td>
<td>—</td>
</tr>
<tr>
<td>112</td>
<td>146</td>
<td>—</td>
</tr>
<tr>
<td>113</td>
<td>147</td>
<td>Å</td>
</tr>
<tr>
<td>114</td>
<td>148</td>
<td>Æ</td>
</tr>
<tr>
<td>115</td>
<td>149</td>
<td>Ø</td>
</tr>
<tr>
<td>116</td>
<td>150</td>
<td>Æ</td>
</tr>
<tr>
<td>117</td>
<td>151</td>
<td>å</td>
</tr>
<tr>
<td>118</td>
<td>152</td>
<td>æ</td>
</tr>
<tr>
<td>119</td>
<td>153</td>
<td>ø</td>
</tr>
<tr>
<td>120</td>
<td>154</td>
<td>Æ</td>
</tr>
<tr>
<td>121</td>
<td>155</td>
<td>β</td>
</tr>
<tr>
<td>122</td>
<td>156</td>
<td>i</td>
</tr>
<tr>
<td>123</td>
<td>157</td>
<td>&lt;</td>
</tr>
<tr>
<td>124</td>
<td>158</td>
<td>&gt;</td>
</tr>
<tr>
<td>125</td>
<td>159</td>
<td>&lt;&lt;</td>
</tr>
<tr>
<td>126</td>
<td>160</td>
<td>&gt;&gt;</td>
</tr>
<tr>
<td>127</td>
<td>161</td>
<td>?</td>
</tr>
<tr>
<td>128</td>
<td>162</td>
<td>i</td>
</tr>
<tr>
<td>129</td>
<td>163</td>
<td>’</td>
</tr>
<tr>
<td>130</td>
<td>164</td>
<td>‚</td>
</tr>
<tr>
<td>131</td>
<td>165</td>
<td>‚</td>
</tr>
<tr>
<td>132</td>
<td>167</td>
<td>^</td>
</tr>
<tr>
<td>133</td>
<td>168</td>
<td>^</td>
</tr>
<tr>
<td>134</td>
<td>169</td>
<td>^</td>
</tr>
<tr>
<td>IDX</td>
<td>ID</td>
<td>Char</td>
</tr>
<tr>
<td>-----</td>
<td>----</td>
<td>------</td>
</tr>
<tr>
<td>135</td>
<td>170</td>
<td>..</td>
</tr>
<tr>
<td>136</td>
<td>171</td>
<td>..</td>
</tr>
<tr>
<td>137</td>
<td>172</td>
<td>~</td>
</tr>
<tr>
<td>138</td>
<td>173</td>
<td>~</td>
</tr>
<tr>
<td>139</td>
<td>174</td>
<td>⌼</td>
</tr>
<tr>
<td>140</td>
<td>175</td>
<td>⌼</td>
</tr>
<tr>
<td>141</td>
<td>176</td>
<td>$\frac{1}{3}$</td>
</tr>
<tr>
<td>142</td>
<td>177</td>
<td>-</td>
</tr>
<tr>
<td>143</td>
<td>178</td>
<td>-</td>
</tr>
<tr>
<td>144</td>
<td>179</td>
<td>⌼</td>
</tr>
<tr>
<td>145</td>
<td>180</td>
<td>⌼</td>
</tr>
<tr>
<td>146</td>
<td>181</td>
<td>⌼</td>
</tr>
<tr>
<td>147</td>
<td>182</td>
<td>⌼</td>
</tr>
<tr>
<td>148</td>
<td>183</td>
<td>⌼</td>
</tr>
<tr>
<td>149</td>
<td>184</td>
<td>⌼</td>
</tr>
<tr>
<td>150</td>
<td>185</td>
<td>1/8</td>
</tr>
<tr>
<td>151</td>
<td>186</td>
<td>1/4</td>
</tr>
<tr>
<td>152</td>
<td>187</td>
<td>3/8</td>
</tr>
<tr>
<td>153</td>
<td>188</td>
<td>1/2</td>
</tr>
<tr>
<td>154</td>
<td>189</td>
<td>5/8</td>
</tr>
<tr>
<td>155</td>
<td>190</td>
<td>3/4</td>
</tr>
<tr>
<td>156</td>
<td>191</td>
<td>7/8</td>
</tr>
<tr>
<td>157</td>
<td>192</td>
<td>1/3</td>
</tr>
<tr>
<td>158</td>
<td>193</td>
<td>2/3</td>
</tr>
</tbody>
</table>

*The Atari Compendium*
### G.14 – Speedo Character Map

<table>
<thead>
<tr>
<th>IDX</th>
<th>ID</th>
<th>Char</th>
</tr>
</thead>
<tbody>
<tr>
<td>159</td>
<td>194</td>
<td>1</td>
</tr>
<tr>
<td>160</td>
<td>195</td>
<td>2</td>
</tr>
<tr>
<td>161</td>
<td>196</td>
<td>3</td>
</tr>
<tr>
<td>162</td>
<td>197</td>
<td>4</td>
</tr>
<tr>
<td>163</td>
<td>198</td>
<td>5</td>
</tr>
<tr>
<td>164</td>
<td>199</td>
<td>6</td>
</tr>
<tr>
<td>165</td>
<td>200</td>
<td>7</td>
</tr>
<tr>
<td>166</td>
<td>201</td>
<td>8</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>IDX</th>
<th>ID</th>
<th>Char</th>
</tr>
</thead>
<tbody>
<tr>
<td>167</td>
<td>202</td>
<td>9</td>
</tr>
<tr>
<td>168</td>
<td>203</td>
<td>0</td>
</tr>
<tr>
<td>169</td>
<td>225</td>
<td>Δ</td>
</tr>
<tr>
<td>170</td>
<td>226</td>
<td>Λ</td>
</tr>
<tr>
<td>171</td>
<td>227</td>
<td>Α</td>
</tr>
<tr>
<td>172</td>
<td>228</td>
<td>Ε</td>
</tr>
<tr>
<td>173</td>
<td>229</td>
<td>δ</td>
</tr>
<tr>
<td>174</td>
<td>230</td>
<td>ι</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>IDX</th>
<th>ID</th>
<th>Char</th>
</tr>
</thead>
<tbody>
<tr>
<td>175</td>
<td>231</td>
<td>’</td>
</tr>
<tr>
<td>176</td>
<td>232</td>
<td>J</td>
</tr>
<tr>
<td>177</td>
<td>234</td>
<td>ζ</td>
</tr>
<tr>
<td>178</td>
<td>235</td>
<td>ζ</td>
</tr>
<tr>
<td>179</td>
<td>236</td>
<td>ζ</td>
</tr>
<tr>
<td>180</td>
<td>237</td>
<td>ζ</td>
</tr>
<tr>
<td>181</td>
<td>238</td>
<td>ζ</td>
</tr>
<tr>
<td>182</td>
<td>239</td>
<td>ζ</td>
</tr>
<tr>
<td>IDX</td>
<td>ID</td>
<td>Char</td>
</tr>
<tr>
<td>-----</td>
<td>----</td>
<td>------</td>
</tr>
<tr>
<td>207</td>
<td>334</td>
<td>ò</td>
</tr>
<tr>
<td>208</td>
<td>335</td>
<td>Ò</td>
</tr>
<tr>
<td>209</td>
<td>336</td>
<td>ñ</td>
</tr>
<tr>
<td>210</td>
<td>337</td>
<td>ñ</td>
</tr>
<tr>
<td>211</td>
<td>338</td>
<td>ñ</td>
</tr>
<tr>
<td>212</td>
<td>339</td>
<td>ñ</td>
</tr>
<tr>
<td>213</td>
<td>340</td>
<td>ñ</td>
</tr>
<tr>
<td>214</td>
<td>341</td>
<td>ñ</td>
</tr>
</tbody>
</table>

The Atari Compendium
<table>
<thead>
<tr>
<th>IDX</th>
<th>ID</th>
<th>Char</th>
<th>IDX</th>
<th>ID</th>
<th>Char</th>
<th>IDX</th>
<th>ID</th>
<th>Char</th>
</tr>
</thead>
<tbody>
<tr>
<td>231</td>
<td>367</td>
<td>`</td>
<td>239</td>
<td>375</td>
<td>ì</td>
<td>247</td>
<td>385</td>
<td>Ê</td>
</tr>
<tr>
<td>232</td>
<td>368</td>
<td>i’</td>
<td>240</td>
<td>376</td>
<td>ì</td>
<td>248</td>
<td>386</td>
<td>è</td>
</tr>
<tr>
<td>233</td>
<td>369</td>
<td>~í</td>
<td>241</td>
<td>377</td>
<td>i</td>
<td>249</td>
<td>387</td>
<td>È</td>
</tr>
<tr>
<td>234</td>
<td>370</td>
<td>~í</td>
<td>242</td>
<td>378</td>
<td>ì</td>
<td>250</td>
<td>388</td>
<td>ë</td>
</tr>
<tr>
<td>235</td>
<td>371</td>
<td>i</td>
<td>243</td>
<td>379</td>
<td>É</td>
<td>251</td>
<td>389</td>
<td>É</td>
</tr>
<tr>
<td>236</td>
<td>372</td>
<td>ì</td>
<td>244</td>
<td>380</td>
<td>é</td>
<td>252</td>
<td>390</td>
<td>ã</td>
</tr>
<tr>
<td>237</td>
<td>373</td>
<td>ê</td>
<td>245</td>
<td>383</td>
<td>È</td>
<td>253</td>
<td>391</td>
<td>ã</td>
</tr>
<tr>
<td>238</td>
<td>374</td>
<td>î</td>
<td>246</td>
<td>384</td>
<td>ë</td>
<td>254</td>
<td>392</td>
<td>ã</td>
</tr>
<tr>
<td>IDX</td>
<td>ID</td>
<td>Char</td>
<td>IDX</td>
<td>ID</td>
<td>Char</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>------</td>
<td>-----</td>
<td>------</td>
<td>------</td>
<td>-----</td>
<td>------</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>303</td>
<td>478</td>
<td>∞</td>
<td>311</td>
<td>488</td>
<td>−</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>304</td>
<td>479</td>
<td></td>
<td>312</td>
<td>489</td>
<td>+</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>305</td>
<td>480</td>
<td></td>
<td>313</td>
<td>500</td>
<td>Γ</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>306</td>
<td>481</td>
<td></td>
<td>314</td>
<td>501</td>
<td>Δ</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>307</td>
<td>482</td>
<td></td>
<td>315</td>
<td>505</td>
<td>Θ</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>308</td>
<td>484</td>
<td></td>
<td>316</td>
<td>515</td>
<td>Σ</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>309</td>
<td>486</td>
<td></td>
<td>317</td>
<td>518</td>
<td>Φ</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>310</td>
<td>487</td>
<td></td>
<td>318</td>
<td>521</td>
<td>Ω</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>319</td>
<td>522</td>
<td>α</td>
<td>320</td>
<td>523</td>
<td>β</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>321</td>
<td>526</td>
<td>γ</td>
<td>322</td>
<td>527</td>
<td>δ</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>323</td>
<td>529</td>
<td>ε</td>
<td>324</td>
<td>530</td>
<td>ζ</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>325</td>
<td>534</td>
<td>η</td>
<td>326</td>
<td>538</td>
<td>θ</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>μ</td>
<td></td>
<td></td>
<td>π</td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

**The Atari Compendium**
<table>
<thead>
<tr>
<th>IDX</th>
<th>ID</th>
<th>Char</th>
<th>IDX</th>
<th>ID</th>
<th>Char</th>
<th>IDX</th>
<th>ID</th>
<th>Char</th>
</tr>
</thead>
<tbody>
<tr>
<td>327</td>
<td>540</td>
<td>σ</td>
<td>335</td>
<td>567</td>
<td>□</td>
<td>343</td>
<td>577</td>
<td>●</td>
</tr>
<tr>
<td>328</td>
<td>542</td>
<td>τ</td>
<td>336</td>
<td>568</td>
<td>□</td>
<td>344</td>
<td>579</td>
<td>●</td>
</tr>
<tr>
<td>329</td>
<td>544</td>
<td>ϕ</td>
<td>337</td>
<td>570</td>
<td>′</td>
<td>345</td>
<td>581</td>
<td>▲</td>
</tr>
<tr>
<td>330</td>
<td>562</td>
<td>♩</td>
<td>338</td>
<td>571</td>
<td>&quot;</td>
<td>346</td>
<td>582</td>
<td>▼</td>
</tr>
<tr>
<td>331</td>
<td>563</td>
<td>♩</td>
<td>339</td>
<td>572</td>
<td>○</td>
<td>347</td>
<td>583</td>
<td>△</td>
</tr>
<tr>
<td>332</td>
<td>564</td>
<td>©</td>
<td>340</td>
<td>573</td>
<td>△</td>
<td>348</td>
<td>584</td>
<td>△</td>
</tr>
<tr>
<td>333</td>
<td>565</td>
<td>℗</td>
<td>341</td>
<td>575</td>
<td>●</td>
<td>349</td>
<td>585</td>
<td>◊</td>
</tr>
<tr>
<td>334</td>
<td>566</td>
<td>TM</td>
<td>342</td>
<td>576</td>
<td>●</td>
<td>350</td>
<td>586</td>
<td>♀</td>
</tr>
<tr>
<td>IDX</td>
<td>ID</td>
<td>Char</td>
<td>IDX</td>
<td>ID</td>
<td>Char</td>
<td>IDX</td>
<td>ID</td>
<td>Char</td>
</tr>
<tr>
<td>-----</td>
<td>-----</td>
<td>------</td>
<td>-----</td>
<td>-----</td>
<td>------</td>
<td>-----</td>
<td>-----</td>
<td>------</td>
</tr>
<tr>
<td>351</td>
<td>587</td>
<td><img src="image" alt="Heart" /></td>
<td>359</td>
<td>595</td>
<td><img src="image" alt="Square" /></td>
<td>367</td>
<td>609</td>
<td><img src="image" alt="acute_diacritic" /></td>
</tr>
<tr>
<td>352</td>
<td>588</td>
<td><img src="image" alt="Club" /></td>
<td>360</td>
<td>598</td>
<td><img src="image" alt="Sun" /></td>
<td>368</td>
<td>610</td>
<td><img src="image" alt="breve_diacritic" /></td>
</tr>
<tr>
<td>353</td>
<td>589</td>
<td><img src="image" alt="Diamond" /></td>
<td>361</td>
<td>599</td>
<td><img src="image" alt="Smiley" /></td>
<td>369</td>
<td>611</td>
<td><img src="image" alt="circumflex" /></td>
</tr>
<tr>
<td>354</td>
<td>590</td>
<td><img src="image" alt="Spade" /></td>
<td>362</td>
<td>600</td>
<td><img src="image" alt="breve_tilde" /></td>
<td>370</td>
<td>612</td>
<td><img src="image" alt="czech_vowel" /></td>
</tr>
<tr>
<td>355</td>
<td>591</td>
<td><img src="image" alt="Hyphen" /></td>
<td>363</td>
<td>605</td>
<td><img src="image" alt="ascender_tilde" /></td>
<td>371</td>
<td>613</td>
<td><img src="image" alt="czech_vowel" /></td>
</tr>
<tr>
<td>356</td>
<td>592</td>
<td><img src="image" alt="Ellipsis" /></td>
<td>364</td>
<td>606</td>
<td><img src="image" alt="ascender_tilde" /></td>
<td>372</td>
<td>614</td>
<td><img src="image" alt="czech_vowel" /></td>
</tr>
<tr>
<td>357</td>
<td>593</td>
<td><img src="image" alt="Square" /></td>
<td>365</td>
<td>607</td>
<td><img src="image" alt="acute_tilde" /></td>
<td>373</td>
<td>619</td>
<td><img src="image" alt="czech_vowel" /></td>
</tr>
<tr>
<td>358</td>
<td>594</td>
<td><img src="image" alt="Checker" /></td>
<td>366</td>
<td>608</td>
<td><img src="image" alt="tilde_y" /></td>
<td>374</td>
<td>620</td>
<td><img src="image" alt="hungarian_vowel" /></td>
</tr>
<tr>
<td>IDX</td>
<td>ID</td>
<td>Char</td>
<td>IDX</td>
<td>ID</td>
<td>Char</td>
<td>IDX</td>
<td>ID</td>
<td>Char</td>
</tr>
<tr>
<td>-----</td>
<td>-----</td>
<td>------</td>
<td>-----</td>
<td>-----</td>
<td>------</td>
<td>-----</td>
<td>-----</td>
<td>------</td>
</tr>
<tr>
<td>375</td>
<td>621</td>
<td>Ć</td>
<td>383</td>
<td>631</td>
<td>Ė</td>
<td>391</td>
<td>643</td>
<td>Ł</td>
</tr>
<tr>
<td>376</td>
<td>622</td>
<td>Ć</td>
<td>384</td>
<td>634</td>
<td>Ė</td>
<td>392</td>
<td>644</td>
<td>Ł</td>
</tr>
<tr>
<td>377</td>
<td>623</td>
<td>Ć</td>
<td>385</td>
<td>637</td>
<td>Ė</td>
<td>393</td>
<td>645</td>
<td>Ł</td>
</tr>
<tr>
<td>378</td>
<td>624</td>
<td>Ć</td>
<td>386</td>
<td>638</td>
<td>Ė</td>
<td>394</td>
<td>646</td>
<td>Ł</td>
</tr>
<tr>
<td>379</td>
<td>625</td>
<td>Ć</td>
<td>387</td>
<td>639</td>
<td>Ė</td>
<td>395</td>
<td>647</td>
<td>Ł</td>
</tr>
<tr>
<td>380</td>
<td>628</td>
<td>Ć</td>
<td>388</td>
<td>640</td>
<td>Ė</td>
<td>396</td>
<td>648</td>
<td>Ł</td>
</tr>
<tr>
<td>381</td>
<td>629</td>
<td>Ć</td>
<td>389</td>
<td>641</td>
<td>Ė</td>
<td>397</td>
<td>649</td>
<td>Ł</td>
</tr>
<tr>
<td>382</td>
<td>630</td>
<td>Ć</td>
<td>390</td>
<td>642</td>
<td>Ė</td>
<td>398</td>
<td>650</td>
<td>Ł</td>
</tr>
</tbody>
</table>

**THE ATARI COMPENDIUM**
<table>
<thead>
<tr>
<th>IDX</th>
<th>ID</th>
<th>Char</th>
<th>IDX</th>
<th>ID</th>
<th>Char</th>
<th>IDX</th>
<th>ID</th>
<th>Char</th>
</tr>
</thead>
<tbody>
<tr>
<td>399</td>
<td>651</td>
<td>Ł</td>
<td>407</td>
<td>662</td>
<td>ų</td>
<td>415</td>
<td>671</td>
<td>Ş</td>
</tr>
<tr>
<td>400</td>
<td>653</td>
<td>Łń</td>
<td>408</td>
<td>663</td>
<td>ųń</td>
<td>416</td>
<td>674</td>
<td>Şń</td>
</tr>
<tr>
<td>401</td>
<td>654</td>
<td>Ąń</td>
<td>409</td>
<td>664</td>
<td>źń</td>
<td>417</td>
<td>675</td>
<td>Şń́</td>
</tr>
<tr>
<td>402</td>
<td>655</td>
<td>Ąń́</td>
<td>410</td>
<td>665</td>
<td>źń́</td>
<td>418</td>
<td>676</td>
<td>Şń́́</td>
</tr>
<tr>
<td>403</td>
<td>656</td>
<td>Ńń</td>
<td>411</td>
<td>666</td>
<td>źń̄</td>
<td>419</td>
<td>677</td>
<td>Şń̄</td>
</tr>
<tr>
<td>404</td>
<td>657</td>
<td>Ńń́</td>
<td>412</td>
<td>667</td>
<td>źń̄</td>
<td>420</td>
<td>678</td>
<td>Şń̄́</td>
</tr>
<tr>
<td>405</td>
<td>660</td>
<td>Ńř</td>
<td>413</td>
<td>669</td>
<td>źř̄</td>
<td>421</td>
<td>679</td>
<td>Şř̄</td>
</tr>
<tr>
<td>406</td>
<td>661</td>
<td>Ńř́</td>
<td>414</td>
<td>670</td>
<td>źř́</td>
<td>422</td>
<td>680</td>
<td>Şř́</td>
</tr>
</tbody>
</table>

**The Atari Compendium**
<table>
<thead>
<tr>
<th>IDX</th>
<th>ID</th>
<th>Char</th>
<th>IDX</th>
<th>ID</th>
<th>Char</th>
<th>IDX</th>
<th>ID</th>
<th>Char</th>
</tr>
</thead>
<tbody>
<tr>
<td>423</td>
<td>681</td>
<td>Ü</td>
<td>431</td>
<td>1223</td>
<td>/</td>
<td>439</td>
<td>1377</td>
<td>..</td>
</tr>
<tr>
<td>424</td>
<td>682</td>
<td>Ū</td>
<td>432</td>
<td>1364</td>
<td>'</td>
<td>440</td>
<td>1380</td>
<td>~</td>
</tr>
<tr>
<td>425</td>
<td>683</td>
<td>Ŭ</td>
<td>433</td>
<td>1365</td>
<td>‘</td>
<td>441</td>
<td>1381</td>
<td>~</td>
</tr>
<tr>
<td>426</td>
<td>684</td>
<td>Ũ</td>
<td>434</td>
<td>1368</td>
<td>`</td>
<td>442</td>
<td>1384</td>
<td>`</td>
</tr>
<tr>
<td>427</td>
<td>685</td>
<td>Ū</td>
<td>435</td>
<td>1369</td>
<td>`</td>
<td>443</td>
<td>1385</td>
<td>`</td>
</tr>
<tr>
<td>428</td>
<td>693</td>
<td>φ</td>
<td>436</td>
<td>1372</td>
<td>^</td>
<td>444</td>
<td>1388</td>
<td>_</td>
</tr>
<tr>
<td>429</td>
<td>695</td>
<td>α</td>
<td>437</td>
<td>1373</td>
<td>_</td>
<td>445</td>
<td>1392</td>
<td>_</td>
</tr>
<tr>
<td>430</td>
<td>797</td>
<td>l</td>
<td>438</td>
<td>1376</td>
<td>_</td>
<td>446</td>
<td>1393</td>
<td>_</td>
</tr>
<tr>
<td>IDX</td>
<td>ID</td>
<td>Char</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>-----</td>
<td>-----</td>
<td>------</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>447</td>
<td>1396</td>
<td>🎨</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>448</td>
<td>1397</td>
<td>🎨</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>449</td>
<td>1400</td>
<td>🎨</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>450</td>
<td>1401</td>
<td>🎨</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>451</td>
<td>1661</td>
<td>🎨</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>452</td>
<td>1667</td>
<td>🎨</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>453</td>
<td>1743</td>
<td>🎨</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>454</td>
<td>1744</td>
<td>🎨</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>455</td>
<td>1747</td>
<td>&quot;</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>456</td>
<td>1748</td>
<td>&quot;</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>457</td>
<td>1751</td>
<td>,</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>458</td>
<td>1752</td>
<td>,</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>459</td>
<td>1753</td>
<td>,</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>460</td>
<td>1756</td>
<td>-</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>461</td>
<td>1761</td>
<td>.</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>462</td>
<td>1766</td>
<td>,</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>463</td>
<td>1771</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>464</td>
<td>1776</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>465</td>
<td>1996</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>466</td>
<td>2022</td>
<td>1</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>467</td>
<td>2028</td>
<td>2</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>468</td>
<td>2034</td>
<td>3</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>469</td>
<td>2040</td>
<td>4</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>470</td>
<td>2046</td>
<td>5</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

The Atari Compendium
<table>
<thead>
<tr>
<th>IDX</th>
<th>ID</th>
<th>Char</th>
<th>IDX</th>
<th>ID</th>
<th>Char</th>
<th>IDX</th>
<th>ID</th>
<th>Char</th>
</tr>
</thead>
<tbody>
<tr>
<td>471</td>
<td>2052</td>
<td>6</td>
<td>479</td>
<td>2984</td>
<td>ग</td>
<td>487</td>
<td>4472</td>
<td>=</td>
</tr>
<tr>
<td>472</td>
<td>2058</td>
<td>7</td>
<td>480</td>
<td>2990</td>
<td>ग</td>
<td>488</td>
<td>4488</td>
<td>,</td>
</tr>
<tr>
<td>473</td>
<td>2064</td>
<td>8</td>
<td>481</td>
<td>3396</td>
<td>ग</td>
<td>489</td>
<td>4489</td>
<td>'</td>
</tr>
<tr>
<td>474</td>
<td>2070</td>
<td>9</td>
<td>482</td>
<td>3580</td>
<td>ग</td>
<td>490</td>
<td>4490</td>
<td>,</td>
</tr>
<tr>
<td>475</td>
<td>2076</td>
<td>0</td>
<td>483</td>
<td>3586</td>
<td>ग</td>
<td>491</td>
<td>4524</td>
<td>Fr</td>
</tr>
<tr>
<td>476</td>
<td>2647</td>
<td>ā</td>
<td>484</td>
<td>3704</td>
<td>ग</td>
<td>492</td>
<td>4736</td>
<td>'</td>
</tr>
<tr>
<td>477</td>
<td>2653</td>
<td>Ṣ</td>
<td>485</td>
<td>3738</td>
<td>ग</td>
<td>493</td>
<td>4744</td>
<td>~</td>
</tr>
<tr>
<td>478</td>
<td>2776</td>
<td>D’</td>
<td>486</td>
<td>3744</td>
<td>ग</td>
<td>494</td>
<td>4903</td>
<td>L</td>
</tr>
<tr>
<td>IDX</td>
<td>ID</td>
<td>Char</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>-----</td>
<td>-----</td>
<td>------</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>495</td>
<td>5042</td>
<td>⬜</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>496</td>
<td>5085</td>
<td>⬜</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>497</td>
<td>5147</td>
<td>⬜</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>498</td>
<td>5196</td>
<td>⬜</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>499</td>
<td>5243</td>
<td>⬜</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>500</td>
<td>5244</td>
<td>⬜</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>501</td>
<td>5249</td>
<td>⬜</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>502</td>
<td>5262</td>
<td>⬜</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>503</td>
<td>5371</td>
<td>☼</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>504</td>
<td>5372</td>
<td>☼</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>505</td>
<td>5403</td>
<td>☼</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>506</td>
<td>5408</td>
<td>☼</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>507</td>
<td>5410</td>
<td>☼</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>508</td>
<td>5418</td>
<td>☼</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>509</td>
<td>5421</td>
<td>☼</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>510</td>
<td>5422</td>
<td>☼</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>511</td>
<td>5423</td>
<td>☼</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>512</td>
<td>5424</td>
<td>☼</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>513</td>
<td>5427</td>
<td>☼</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>514</td>
<td>5428</td>
<td>☼</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>515</td>
<td>5429</td>
<td>☼</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>516</td>
<td>5430</td>
<td>☼</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>517</td>
<td>5431</td>
<td>☼</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>518</td>
<td>5432</td>
<td>☼</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

**THE ATARI COMPENDIUM**
<table>
<thead>
<tr>
<th>IDX</th>
<th>ID</th>
<th>Char</th>
</tr>
</thead>
<tbody>
<tr>
<td>543</td>
<td>5523</td>
<td>n</td>
</tr>
<tr>
<td>544</td>
<td>5524</td>
<td>o</td>
</tr>
<tr>
<td>545</td>
<td>5527</td>
<td>r</td>
</tr>
<tr>
<td>546</td>
<td>5528</td>
<td>s</td>
</tr>
<tr>
<td>547</td>
<td>5529</td>
<td>t</td>
</tr>
<tr>
<td>548</td>
<td>5536</td>
<td></td>
</tr>
<tr>
<td>549</td>
<td>5537</td>
<td></td>
</tr>
<tr>
<td>550</td>
<td>5538</td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>IDX</th>
<th>ID</th>
<th>Char</th>
</tr>
</thead>
<tbody>
<tr>
<td>551</td>
<td>5539</td>
<td></td>
</tr>
<tr>
<td>552</td>
<td>5540</td>
<td></td>
</tr>
<tr>
<td>553</td>
<td>5541</td>
<td></td>
</tr>
<tr>
<td>554</td>
<td>5542</td>
<td></td>
</tr>
<tr>
<td>555</td>
<td>5543</td>
<td></td>
</tr>
<tr>
<td>556</td>
<td>5544</td>
<td></td>
</tr>
<tr>
<td>557</td>
<td>5545</td>
<td></td>
</tr>
<tr>
<td>558</td>
<td>5548</td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>IDX</th>
<th>ID</th>
</tr>
</thead>
<tbody>
<tr>
<td>559</td>
<td>5554</td>
</tr>
<tr>
<td>560</td>
<td>5594</td>
</tr>
<tr>
<td>561</td>
<td>5595</td>
</tr>
<tr>
<td>562</td>
<td>5596</td>
</tr>
<tr>
<td>563</td>
<td>6458</td>
</tr>
</tbody>
</table>
THE DRAG & DROP PROTOCOL
Overview

The drag and drop protocol provides a simple method of data transmission between applications that support it. Because this protocol relies on the use of named pipes, the use of the drag and drop protocol is only possible under MultiTOS.

A drag and drop operation involves the user selecting a piece of program data (or perhaps several pieces) in the ‘originator’ application and dragging that piece of data with the mouse to the window of a ‘recipient’ application. This appendix will detail the drag and drop protocol from the perspective of the originator and the recipient.

You should note that during a drag and drop operation, neither application should lock the screen with wind_update().

The Originator

When the user selects an object or group of objects, drags the mouse (and objects), and releases the mouse button outside one of your application window’s work areas, the operation is a candidate for a drag and drop operation.

When this action is initiated by the user, your application should call wind_find() to determine the window handle of the window at the drop location. From the window handle you can use wind_get() to determine the owner’s application identifier which will be needed to send an AES message to the application.

At this point you should use Psignal() to cause SIGPIPE (13) signals to be ignored and create a pipe named DRAGDROP.xx where ‘xx’ is a unique two character combination. The pipe created should have its ‘hidden’ attribute set. This causes reads to return EOF when the other end of the pipe is closed. To ensure your value is unique, try using the ASCII representation of your own application ID. If the Fcreate() fails, try a new combination until you find one that is available.

Now use appl_write() to send an AES message to the application whose window was targeted (the recipient) as follows:

<table>
<thead>
<tr>
<th>WORD</th>
<th>Contents</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>AP_DRAGDROP (63)</td>
</tr>
<tr>
<td>1</td>
<td>Originator’s application id.</td>
</tr>
<tr>
<td>2</td>
<td>0</td>
</tr>
<tr>
<td>3</td>
<td>Window handle of the target.</td>
</tr>
<tr>
<td>4</td>
<td>Mouse X position at time of drop.</td>
</tr>
<tr>
<td>5</td>
<td>Mouse Y position at time of drop.</td>
</tr>
<tr>
<td>6</td>
<td>Keyboard shift status at time of drop.</td>
</tr>
<tr>
<td>7</td>
<td>2 character pipe ID packed into a WORD (this is the file extension of the created pipe).</td>
</tr>
</tbody>
</table>
The originator application should now use `fselect()` to wait for either a write to the pipe or a timeout (3 to 4 seconds should be sufficient). If the call times out then the drag and drop operation failed and the pipe should be closed, otherwise, read one byte from the pipe which should be either `DD_OK` (0) or `DD_NAK` (1).

**DD_OK** means that the recipient wishes to continue the exchange. **DD_NAK** means that the user dropped the data on a window not prepared to accept data and that the pipe should be closed and the drag and drop operation aborted.

On receipt of a **DD_OK**, the originator should then read an additional 32 **BYTES** from the pipe. These 32 **BYTES** consist of eight 4 **BYTE** data type values that the recipient understands in order of preference. This list is not necessarily complete and the originator should not abort simply because it can’t handle any of the listed data types. If less than eight data types are listed by the recipient the 32 bytes will be padded with zeros.

Data type values are four-byte ASCII values that represent data that might be exchanged. When these values are prefixed with a period, they represent data in a format that might be stored in a disk file. Examples of these are ‘.IMG’, ‘.TXT’, and ‘.GEM’. Some data types such as ‘ARGS’ or ‘PATH’ are not prefixed with a period because they represent special data.

The desktop sends an ‘ARGS’ drag and drop message to an application window when the user drags a group of file icons to an application window. The ‘ARGS’ data consists of a standard command line with the names of each file. ‘ARGS’ data should be translated for non-**TOS** file systems. Characters within single quotes should be interpreted as a single filename. Two single quotes in a row should be interpreted as a single quote.

After the originator has consulted the 32 byte list or preferred file types, it should construct its own structure consisting of the following data:

1. The type of data the originator has decided to send (4 ASCII bytes), ex: ‘.IMG’.
2. The length of data in bytes (**LONG**).
3. The data’s name in ASCII format terminated by a **NULL** (this is a variable length field but should be brief as it will be used to label an icon which represents the data chunk), ex: “ASCII Text”.
4. The filename the data is associated with in ASCII format terminated by a **NULL** (again, a variable length field), ex: “SAMPLE.TXT”.

The originator should now write a **WORD** to the pipe signifying the length of the header and then the header itself. After doing so, the recipient will write a one byte reply indicating a return code from the following list:
The Recipient – H.5

<table>
<thead>
<tr>
<th>Name</th>
<th>Value</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>DD_OK</td>
<td>0</td>
<td>Ready to receive data. After receiving this message you should <strong>fwrite()</strong> the actual data to the pipe and then <strong>fclose()</strong> it to complete the operation.</td>
</tr>
<tr>
<td>DD_NAK</td>
<td>1</td>
<td>Abort the drag and drop. After receiving this message, close the pipe and abort the operation.</td>
</tr>
<tr>
<td>DD_EXT</td>
<td>2</td>
<td>The recipient cannot accept the format the data is in. You may either construct a new header and send it as before or close the pipe to abort the operation.</td>
</tr>
<tr>
<td>DD_LEN</td>
<td>3</td>
<td>The recipient cannot handle so much data. Either use a format which would cause less data to be sent or close the pipe to abort.</td>
</tr>
<tr>
<td>DD_TRASH</td>
<td>4</td>
<td>The data has been dropped on a trashcan. The pipe should be <strong>fclose()’d</strong> and the data should be deleted from the originator application.</td>
</tr>
<tr>
<td>DD_PRINTER</td>
<td>5</td>
<td>The data has been dropped on a printer. The pipe should be <strong>fclose()’d</strong> and the data should be printed.</td>
</tr>
<tr>
<td>DD_CLIPBOARD</td>
<td>6</td>
<td>The data has been dropped on a clipboard. The pipe should be <strong>fclose()’d</strong> and the exchange should be treated like a ‘Copy’ clipboard operation.</td>
</tr>
</tbody>
</table>

The one exception to the above procedure involves the ‘PATH’ data type. If the recipient agrees to the ‘PATH’ data type by sending a **DD_OK**, the originator should **read** a path string (terminated by a **NULL** byte). The path string should be the complete pathname represented by the target window, ex: “C:\WORDPRO\FILES\”. The size of the data, as specified in the header, specifies the maximum size of the string the recipient should write.

The Recipient

The drag and drop protocol begins for the recipient upon receipt of the **AP_DRAGDROP** message. When this message is received, the recipient should immediately open the pipe ‘U:\PIPE\DRAGDROP.xx’, where ‘xx’ is the two-byte ASCII identifier given in **WORD 7** of the message, and write a **DD_OK** (0) to the pipe.

Next, as the recipient, you should construct a 32 byte structure consisting of eight 4 byte data names your application can receive. If your application recognizes less than eight types of data pad the 32 bytes with zeros. After this structure is constructed, write it to the pipe.

Now you should read a **WORD** from the pipe which will indicate the size of the message header which should be read immediately after. The message header consists of a four byte ASCII data type, a **LONG** indicating the size of the data, a **NULL** terminated string of variable size which identifies the data (or simply **NULL** if none), and a **NULL** terminated filename (or **NULL** if none).

After decoding the message header you should respond with one of the one-byte response codes as listed in the previous table. If the recipient cannot process the data type sent, it should send **DD_EXT** and wait for reception of another header (preceded again by a **WORD** headed size). If
the originator cannot supply any more data types you will receive a 0 byte return from the
Fread() call and you should Fclose() the pipe and abort.

If the data type is acceptable, respond with DD_OK, read the number of data bytes as indicated
in the header to receive the actual data, and then close the pipe.

A special case arises if the header specifies ‘PATH’ as a data type. In this case you should send
a DD_OK message (if appropriate) and write the pathname associated with the target window
(you can write as many bytes as is specified in the message header data length).
– APPENDIX I –

THE PROGRAMMABLE SOUND GENERATOR
Controlling the PSG

Creating sound effects and music is possible with either of two system calls. **Dosound()** processes commands in a supplied buffer during interrupt processing (50 times per second). It is best suited, therefore, at playing musical passages while program flow continues. **Giaccess()** provides register-level control over the PSG resulting in a higher level of flexibility and constant updating by the application. This makes **Giaccess()** more suited for short sound effects.

The function definitions of **Dosound()** and **Giaccess()** both reference the register numbers of the PSG. It should be noted that registers 14 and 15 actually control periperals connected to Port A and Port B of the PSG. The PSG’s registers are assigned as follows:

<table>
<thead>
<tr>
<th>Name</th>
<th>register</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>PSG_APITCHLOW</td>
<td>0</td>
<td>Set the pitch of the PSG’s channel A to the value in registers 0 and 1.</td>
</tr>
<tr>
<td>PSG_BPITCHHIGH</td>
<td>1</td>
<td>REGISTER 0 CONTAINS THE LOWER 8 BITS OF THE FREQUENCY AND THE LOWER 4 BITS OF REGISTER 1 CONTAIN THE UPPER 4 BITS OF THE FREQUENCY’S 12-BIT VALUE.</td>
</tr>
<tr>
<td>PSG_BPITCHLOW</td>
<td>2</td>
<td>Set the pitch of the PSG’s channel B to the value in registers 0 and 1.</td>
</tr>
<tr>
<td>PSG_BPITCHHIGH</td>
<td>3</td>
<td>REGISTER 0 CONTAINS THE LOWER 8 BITS OF THE FREQUENCY AND THE LOWER 4 BITS OF REGISTER 1 CONTAIN THE UPPER 4 BITS OF THE FREQUENCY’S 12-BIT VALUE.</td>
</tr>
<tr>
<td>PSG_CPITCHLOW</td>
<td>2</td>
<td>Set the pitch of the PSG’s channel C to the value in registers 0 and 1.</td>
</tr>
<tr>
<td>PSG_CPITCHHIGH</td>
<td>3</td>
<td>REGISTER 0 CONTAINS THE LOWER 8 BITS OF THE FREQUENCY AND THE LOWER 4 BITS OF REGISTER 1 CONTAIN THE UPPER 4 BITS OF THE FREQUENCY’S 12-BIT VALUE.</td>
</tr>
<tr>
<td>PSG_NOISEPITCH</td>
<td>6</td>
<td>The lower five bits of this register set the pitch of white noise. The lower the value, the higher the pitch.</td>
</tr>
<tr>
<td>PSG_MODE</td>
<td>7</td>
<td>This register contains an eight bit map which determines various aspects of sound generation. Setting each bit on causes the following actions:</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Name</th>
<th>Bit Mask</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>PSG_ENABLEA</td>
<td>0x01</td>
<td>Chnl A tone enable</td>
</tr>
<tr>
<td>PSG_ENABLEB</td>
<td>0x02</td>
<td>Chnl B tone enable</td>
</tr>
<tr>
<td>PSG_ENABLEC</td>
<td>0x04</td>
<td>Chnl C tone enable</td>
</tr>
<tr>
<td>PSG_NOISEA</td>
<td>0x08</td>
<td>Chnl A white noise on</td>
</tr>
<tr>
<td>PSG_NOISEB</td>
<td>0x10</td>
<td>Chnl B white noise on</td>
</tr>
<tr>
<td>PSG_NOISEC</td>
<td>0x20</td>
<td>Chnl C white noise on</td>
</tr>
<tr>
<td>PSG_PRTAOUT</td>
<td>0x40</td>
<td>Port A: 0 = input 1 = output</td>
</tr>
<tr>
<td>PSG_PRTBOUT</td>
<td>0x80</td>
<td>Port B: 0 = input 1 = output</td>
</tr>
<tr>
<td>PSG_AVOLUME</td>
<td>8</td>
<td>This register controls the volume of channel A. Values from 0-15 are absolute volumes with 0 being the softest and 15 being the loudest. Setting bit 4 causes the PSG to ignore the volume setting and to use the envelope setting in register 13.</td>
</tr>
<tr>
<td>PSG_BVOLUME</td>
<td>9</td>
<td>This register controls the volume of channel B. Values from 0-15 are absolute volumes with 0 being the softest and 15 being the loudest. Setting bit 4 causes the PSG to ignore the volume setting and to use the envelope setting in register 13.</td>
</tr>
</tbody>
</table>
I.4 – The Programmable Sound Generator

<table>
<thead>
<tr>
<th>Decoder</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>PSG_CVOLUME</td>
<td>10</td>
</tr>
<tr>
<td>PSG_FREQLOW</td>
<td>11</td>
</tr>
<tr>
<td>PSG_FREQHIGH</td>
<td>12</td>
</tr>
<tr>
<td>PSG_ENVELOPE</td>
<td>13</td>
</tr>
<tr>
<td>PSG_PORTA</td>
<td>14</td>
</tr>
<tr>
<td>PSG_PORTB</td>
<td>15</td>
</tr>
</tbody>
</table>

This register controls the volume of channel C. Values from 0-15 are absolute volumes with 0 being the softest and 15 being the loudest. Setting bit 4 causes the PSG to ignore the volume setting and to use the envelope setting in register 13.

Register 11 contains the low byte and register 12 contains the high byte of the frequency of the waveform specified in register 13. This value may range from 0 to 65535.

The lower four bits of the register contain a value which defines the envelope waveform of the PSG. The best definition of values is obtained through experimentation.

This register accesses Port A of the Yamaha PSG. It is recommended that the functions `Onqibit()` and `Offqibit()` be used to access this register.

This register accesses Port B of the Yamaha PSG. This register is currently assigned to the data in/out line of the Centronics Parallel port.

The following table lists the twelve-bit value required to produce the desired musical tones with the PSG’s tone generators A, B, and C. The upper nibble of the value is placed into the ‘coarse-tuning’ register and the lower BYTE is placed into the ‘fine-tuning’ register. In addition, because the PSG must approximate musical frequencies according to an equal-tempered scale, the ideal and actual frequencies are also listed.

<table>
<thead>
<tr>
<th>Note</th>
<th>Ideal Frequency</th>
<th>Actual Frequency</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>C1</td>
<td>32.703</td>
<td>32.698</td>
<td>0xD5D</td>
</tr>
<tr>
<td>C#1</td>
<td>34.648</td>
<td>34.653</td>
<td>0xC9C</td>
</tr>
<tr>
<td>D1</td>
<td>36.708</td>
<td>36.712</td>
<td>0xBE7</td>
</tr>
<tr>
<td>D#1</td>
<td>38.891</td>
<td>38.895</td>
<td>0xB3C</td>
</tr>
<tr>
<td>E1</td>
<td>41.203</td>
<td>41.201</td>
<td>0xA9B</td>
</tr>
<tr>
<td>F1</td>
<td>43.654</td>
<td>43.662</td>
<td>0xA02</td>
</tr>
<tr>
<td>F#1</td>
<td>46.249</td>
<td>46.243</td>
<td>0x973</td>
</tr>
<tr>
<td>G1</td>
<td>48.999</td>
<td>48.997</td>
<td>0x8EB</td>
</tr>
<tr>
<td>G#1</td>
<td>51.913</td>
<td>51.908</td>
<td>0x86B</td>
</tr>
<tr>
<td>A1</td>
<td>55.000</td>
<td>54.995</td>
<td>0x7F2</td>
</tr>
<tr>
<td>A#1</td>
<td>58.270</td>
<td>58.261</td>
<td>0x7B0</td>
</tr>
<tr>
<td>B1</td>
<td>61.735</td>
<td>61.733</td>
<td>0x714</td>
</tr>
<tr>
<td>C2</td>
<td>65.406</td>
<td>65.416</td>
<td>0x6AE</td>
</tr>
<tr>
<td>C#2</td>
<td>69.296</td>
<td>69.307</td>
<td>0x64E</td>
</tr>
<tr>
<td>D2</td>
<td>73.416</td>
<td>73.399</td>
<td>0x5F4</td>
</tr>
<tr>
<td>D#2</td>
<td>77.782</td>
<td>77.789</td>
<td>0x59E</td>
</tr>
<tr>
<td>E2</td>
<td>82.406</td>
<td>82.432</td>
<td>0x54D</td>
</tr>
<tr>
<td>F2</td>
<td>87.308</td>
<td>87.323</td>
<td>0x501</td>
</tr>
<tr>
<td>F#2</td>
<td>92.498</td>
<td>92.523</td>
<td>0x4B9</td>
</tr>
<tr>
<td>G2</td>
<td>97.998</td>
<td>98.037</td>
<td>0x475</td>
</tr>
<tr>
<td>G#2</td>
<td>103.826</td>
<td>103.863</td>
<td>0x357</td>
</tr>
<tr>
<td>A2</td>
<td>110.000</td>
<td>109.991</td>
<td>0x3F9</td>
</tr>
<tr>
<td>A#2</td>
<td>116.540</td>
<td>116.522</td>
<td>0x3C0</td>
</tr>
<tr>
<td>B2</td>
<td>123.470</td>
<td>123.467</td>
<td>0x38A</td>
</tr>
<tr>
<td>C3</td>
<td>130.812</td>
<td>130.831</td>
<td>0x357</td>
</tr>
<tr>
<td>Note</td>
<td>Ideal Frequency</td>
<td>Actual Frequency</td>
<td>Value</td>
</tr>
<tr>
<td>------</td>
<td>-----------------</td>
<td>------------------</td>
<td>-------</td>
</tr>
<tr>
<td>D5</td>
<td>587.328</td>
<td>588.741</td>
<td>0xBE</td>
</tr>
<tr>
<td>D#5</td>
<td>622.256</td>
<td>621.449</td>
<td>0xB4</td>
</tr>
<tr>
<td>E5</td>
<td>659.248</td>
<td>658.005</td>
<td>0xAA</td>
</tr>
<tr>
<td>F5</td>
<td>698.464</td>
<td>699.130</td>
<td>0xA0</td>
</tr>
<tr>
<td>F#5</td>
<td>739.984</td>
<td>740.800</td>
<td>0x97</td>
</tr>
<tr>
<td>G5</td>
<td>783.984</td>
<td>782.243</td>
<td>0x8F</td>
</tr>
<tr>
<td>G#5</td>
<td>830.608</td>
<td>828.598</td>
<td>0x87</td>
</tr>
<tr>
<td>A5</td>
<td>880.000</td>
<td>880.794</td>
<td>0x7F</td>
</tr>
<tr>
<td>A#5</td>
<td>932.320</td>
<td>932.173</td>
<td>0x78</td>
</tr>
<tr>
<td>B5</td>
<td>987.760</td>
<td>989.918</td>
<td>0x71</td>
</tr>
<tr>
<td>C6</td>
<td>1046.496</td>
<td>1045.428</td>
<td>0x6B</td>
</tr>
<tr>
<td>C#6</td>
<td>1108.736</td>
<td>1107.532</td>
<td>0x65</td>
</tr>
<tr>
<td>D6</td>
<td>1174.656</td>
<td>1177.482</td>
<td>0x5F</td>
</tr>
<tr>
<td>D#6</td>
<td>1244.512</td>
<td>1242.898</td>
<td>0x5A</td>
</tr>
<tr>
<td>E6</td>
<td>1318.496</td>
<td>1316.009</td>
<td>0x55</td>
</tr>
<tr>
<td>F6</td>
<td>1396.928</td>
<td>1398.260</td>
<td>0x50</td>
</tr>
<tr>
<td>F#6</td>
<td>1479.968</td>
<td>1471.852</td>
<td>0x4C</td>
</tr>
<tr>
<td>G6</td>
<td>1567.968</td>
<td>1575.504</td>
<td>0x47</td>
</tr>
<tr>
<td>G#6</td>
<td>1661.216</td>
<td>1669.564</td>
<td>0x43</td>
</tr>
<tr>
<td>A6</td>
<td>1760.000</td>
<td>1747.825</td>
<td>0x40</td>
</tr>
<tr>
<td>A#6</td>
<td>1864.640</td>
<td>1864.346</td>
<td>0x3C</td>
</tr>
<tr>
<td>B6</td>
<td>1975.520</td>
<td>1962.470</td>
<td>0x39</td>
</tr>
<tr>
<td>C7</td>
<td>2092.992</td>
<td>2110.581</td>
<td>0x35</td>
</tr>
<tr>
<td>C#7</td>
<td>2217.472</td>
<td>2237.216</td>
<td>0x32</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Note</th>
<th>Ideal Frequency</th>
<th>Actual Frequency</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>D7</td>
<td>2349.312</td>
<td>2330.433</td>
<td>0x30</td>
</tr>
<tr>
<td>D#7</td>
<td>2489.024</td>
<td>2485.795</td>
<td>0x2D</td>
</tr>
<tr>
<td>E7</td>
<td>2636.992</td>
<td>2663.352</td>
<td>0x2A</td>
</tr>
<tr>
<td>F7</td>
<td>2793.856</td>
<td>2796.520</td>
<td>0x28</td>
</tr>
<tr>
<td>F#7</td>
<td>2959.936</td>
<td>2943.705</td>
<td>0x26</td>
</tr>
<tr>
<td>G7</td>
<td>3135.936</td>
<td>3107.244</td>
<td>0x24</td>
</tr>
<tr>
<td>G#7</td>
<td>3322.432</td>
<td>3290.023</td>
<td>0x22</td>
</tr>
<tr>
<td>A7</td>
<td>3520.000</td>
<td>3495.649</td>
<td>0x20</td>
</tr>
<tr>
<td>A#7</td>
<td>3729.280</td>
<td>3728.693</td>
<td>0xC1</td>
</tr>
<tr>
<td>B7</td>
<td>3951.040</td>
<td>3995.028</td>
<td>0xC1</td>
</tr>
<tr>
<td>C8</td>
<td>4185.984</td>
<td>4142.992</td>
<td>0xC1</td>
</tr>
<tr>
<td>C#8</td>
<td>4434.944</td>
<td>4474.431</td>
<td>0xC1</td>
</tr>
<tr>
<td>D8</td>
<td>4698.624</td>
<td>4660.866</td>
<td>0xC1</td>
</tr>
<tr>
<td>D#8</td>
<td>4978.048</td>
<td>5084.581</td>
<td>0xC1</td>
</tr>
<tr>
<td>E8</td>
<td>5273.984</td>
<td>5326.704</td>
<td>0xC1</td>
</tr>
<tr>
<td>F8</td>
<td>5587.712</td>
<td>5593.039</td>
<td>0xC1</td>
</tr>
<tr>
<td>F#8</td>
<td>5919.872</td>
<td>5887.410</td>
<td>0xC1</td>
</tr>
<tr>
<td>G8</td>
<td>6271.872</td>
<td>6214.488</td>
<td>0xC1</td>
</tr>
<tr>
<td>G#8</td>
<td>6644.864</td>
<td>6580.046</td>
<td>0xC1</td>
</tr>
<tr>
<td>A8</td>
<td>7040.000</td>
<td>6991.299</td>
<td>0xC1</td>
</tr>
<tr>
<td>A#8</td>
<td>7458.560</td>
<td>7457.560</td>
<td>0xC1</td>
</tr>
<tr>
<td>B8</td>
<td>7902.080</td>
<td>7990.056</td>
<td>0xC1</td>
</tr>
</tbody>
</table>
Sound Envelopes

An envelope may be applied to sounds generated by the PSG. Registers 11 and 12 specify the frequency of this envelope and the low four bits of register 13 specifies the envelope shape as follows (an ‘x’ digit means either 0 or 1):

<table>
<thead>
<tr>
<th>Value</th>
<th>Waveform Shape</th>
</tr>
</thead>
<tbody>
<tr>
<td>%00xx</td>
<td></td>
</tr>
<tr>
<td>%01xx</td>
<td></td>
</tr>
<tr>
<td>%1000</td>
<td></td>
</tr>
<tr>
<td>%1001</td>
<td></td>
</tr>
<tr>
<td>%1010</td>
<td></td>
</tr>
<tr>
<td>%1011</td>
<td></td>
</tr>
<tr>
<td>%1100</td>
<td></td>
</tr>
<tr>
<td>%1101</td>
<td></td>
</tr>
<tr>
<td>%1110</td>
<td></td>
</tr>
<tr>
<td>%1111</td>
<td></td>
</tr>
</tbody>
</table>
Bibliography

Atari GEMDOS Reference Manual
Atari Corp., 1196 Borregas Ave., Sunnyvale, CA 94086
Atari Corp. (1986)

Atari MetaDOS Developers Manual
Atari Corp., 1196 Borregas Ave., Sunnyvale, CA 94086
Atari Corp. (1990)

Atari MultiTOS User Interface Guidelines
Atari Corp., 1196 Borregas Ave., Sunnyvale, CA 94086
Atari Corp. (1992)

ISBN 3-88745-888-5, SYBEX-Verlag GmbH, Postfach 30 09 61, 4000 Düsseldorf 30, Germany

Atari SFP004 Floating Point Coprocessor Developer Kit
Atari Corp., 1196 Borregas Ave., Sunnyvale, CA 94086
Atari Corp. (1988)

Atari ST Engineering Hardware Specifications
Atari Corp., 1196 Borregas Ave., Sunnyvale, CA 94086
Atari Corp. (1985)

Atari ST GEM Programmer’s Reference  Norbert Szczepanski and Bernd Gunther (1985)

Atari ST/STe/MSTe/TT/Falcon030 Hardware Register Listing v6.0  Dan Hollis (1993)
Dan Hollis c/o ViewTouch Corp., 344 NE Terry Ln., Grants Pass, OR 97526

Atari TT030 Hardware Reference Manual
Atari Corp., 1196 Borregas Ave., Sunnyvale, CA 94086
Atari Corp. (1990)

ISBN 0-87455-093-9, COMPUTE! Books, P.O. Box 5406, Greensboro, NC 27403

ISBN 0-87455-114-5, COMPUTE! Books, P.O. Box 5406, Greensboro, NC 27403

ISBN 0-87455-149-8, COMPUTE! Books, P.O. Box 5406, Greensboro, NC 27403

Devpac DSP User Manual
Hisoft, The Old School Rd., Greenfield, Bedford MK45 5DE, United Kingdom
Hisoft (1993)

Motorola Literature Distribution, P.O. Box 20912, Phoenix, AZ 85036

Falcon030 Hardware Reference Guide
Atari Corp., 1196 Borregas Ave., Sunnyvale, CA 94086
Atari Corp. (1992)

GEMDOS Extended Argument Specification
Atari Corp. (1986)
Bibliography

Atari Corp., 1196 Borregas Ave., Sunnyvale, CA 94086

**GEM Programmer’s Guide, Volume 1: VDI**  
Digital Research, Inc., 60 Garden Ct., P.O. Box DRI, Monterey, CA 93942

Digital Research, Inc., 60 Garden Ct., P.O. Box DRI, Monterey, CA 93942

**A Hitchhiker’s Guide to the BIOS**  
Atari Corp. (1985)
Atari Corp., 1196 Borregas Ave., Sunnyvale, CA 94086

**Indispensable PC Hardware Book, The**  
Hans-Peter Messmer (1994)

Hisoft (1991)
Hisoft, The Old School Rd., Greenfield, Bedford MK45 5DE, United Kingdom

**Lattice C 5.5, Addendum: Libraries**  
Hisoft (1991)
Hisoft, The Old School Rd., Greenfield, Bedford MK45 5DE, United Kingdom

**MC68000 Family Programmer’s Reference Manual**  
Motorola, Inc. (1989)
Motorola Literature Distribution, P.O. Box 20912, Phoenix, AZ 85036

**MC68030 Enhanced 32-Bit Microprocessor User’s Manual**  
Motorola, Inc. (1990)

**MC68881/MC68882 Floating-Point Coprocessor User’s Manual**  
Motorola, Inc. (1989)

**MK68901 Multi-Function Peripheral Data Sheet**  
United Technologies Mostek (1982)
United Technologies Mostek, 1215 W. Crosby Rd., Carrolton, TX 75006

**MiNT/MultiTOS Release Notes**  
Atari Corp. (1992)
Atari Corp., 1196 Borregas Ave., Sunnyvale, CA 94086

**Modern Atari System Software**  
Hisoft (1993)
Hisoft, The Old School Rd., Greenfield, Bedford MK45 5DE, United Kingdom

**Pexec Cookbook, Third Edition**  
Atari Corp. (1991)
Atari Corp., 1196 Borregas Ave., Sunnyvale, CA 94086

**A Programmer’s Guide to FSMGDOS**  
Atari Corp. (1991)
Atari Corp., 1196 Borregas Ave., Sunnyvale, CA 94086

**Rainbow TOS Release Notes**  
Atari Corp. (1989)
Atari Corp., 1196 Borregas Ave., Sunnyvale, CA 94086

THE ATARI COMpendium
Bibliography

ST Disk Drives: Inside and Out  Uwe Braun, Stefan Dittrich, and Axel Schramm (1986)

STe Developer Addendum  Atari Corp. (1990)
Atari Corp., 1196 Borregas Ave., Sunnyvale, CA 94086

Atari Corp., 1196 Borregas Ave., Sunnyvale, CA 94086

Atari Corp., 1196 Borregas Ave., Sunnyvale, CA 94086

Atari Corp., 1196 Borregas Ave., Sunnyvale, CA 94086
Index

1040ST, 1.3
1040STe, 1.4
260ST, 1.3
520ST, 1.3
56001, see DSP
68000, 1.3, 5.3
68030, 1.5, 5.3
6850, 5.10
68881/2, 1.4-1.5, 5.4-5.6
  _AKP cookie, 3.13, 4.13
  _CPU cookie, 3.11
  _FDC cookie, 3.11, 4.64
  _FLK cookie, 3.12, 2.7, 2.82
  _FRB cookie, 3.12
  _FPU cookie, 3.11, 5.4
  _IDT cookie, 3.13
  _MCH cookie, 3.12
  _mediach() function, 3.15
  _NET cookie, 3.12
  _SND cookie, 3.12, 4.6
  _SWI cookie, 3.12
  _vblqueue, 3.19, B.9
  _VDO cookie, 3.11, 4.3

A
  about menu, 11.15, 6.26
  access permissions, see MiNT
    access permissions
  AC_CLOSE message, 6.7, 6.68
  AC_OPEN message, 6.7, 6.68
  ACIA, B.43-B.44
  ACSI, 4.15
  ADC, 4.6
  address error, 3.35, B.4
  AES, 6.1
    alerts, 6.25, 6.77, 11.10
    application identifier, 6.4,
      6.47, 6.53
    application services
      library, 6.45
    applications, 6.4, 11.24
    clipping rectangles, 6.32
    desk accessories, 6.7
desktop window, 6.31
dialogs, 6.24, 6.81, 11.8
  drop-down list boxes, 6.28,
    6.108, 11.19
  environment string, 6.9,
    6.139
  event dispatcher, 6.9
  event library, 6.59
  event loop, 6.4
  file selector library, 6.85,
    11.12
  form library, 6.75
  function calling procedure,
    6.37
  graphics library, 6.89
  hierarchical menus, 6.27,
    6.103, 11.20
  language, 6.49, 11.23
  menu buffer, 6.28, 6.154
  menus, 6.25, 11.15
  menu library, 6.101
  message events, 6.11, 6.64
  message types, 6.9
  mouse button events, 6.12,
    6.61
  objects, 6.13
  object library, 6.113
  popup menus, 6.28, 6.108,
    11.18
  rectangle list, 6.32
  resource library, 6.125
  scrap library, 6.135
  shell buffer, 6.35, 6.140-
    6.141
  shell library, 6.137
  timer events, 6.12, 6.73
  user-defined messages, 6.12, 6.58
  VDI workstation, 6.33, 6.92
  window toolbars, 6.33,
    11.14
  windows, 6.29, 11.4
  window library, 6.147
  AES_BIOS device, 2.17
  AES_MT device, 2.17
  AESPB structure, 6.37
  AHDI, 3.5, 4.15, B.10
  alerts, see AES alerts
  alertware, 11.3
  alt-help screen dump, 4.91, B.12
  alternative RAM, see memory
  types
  advanced keyboard processor,
    see _AKP cookie
  appl_exit(), 6.47
  appl_find(), 6.47
  appl_getinfo(), 6.48
  appl_init(), 6.4, 6.7, 6.53
  appl_read(), 6.12, 6.54
  appl_search(), 6.55
  appl_tplay(), 6.56
  appl_trecord(), 6.57
  appl_write(), 6.12, 6.58
  APPLBLK structure, 6.23
  application cartridges, 3.3, 5.7
  application services library, see
    AES application services
    library
  application software, 11.24
  AS68, 1.9
  ASCII, E.1
  assembly language, 1.9
  ASSIGN.SYS file, 7.12
  ARGs data type, H.4
  Atari Extended Argument
    Specification, see
    GEMDOS ARGV
  Atari GEM, see GEM
  attenuation, see sound
    attenuation
  attributes, see GEMDOS file
    attributes
  auto-vector interrupts, B.4
  aux: file, see serial device

B
  bad sector list, 4.16
  basepage, 2.11
  BASIC, 1.9
  Bconin(), 3.27
  Bconmap(), 3.14, 4.17, 4.23
  Bconout(), 3.28
  Bconstat(), 3.28
  Bcostat(), 3.29
  bezier curves, see GDOS bezier
    curves
Index

BGM partition, 4.16
BIOS, 3.1
calling from an interrupt, 3.22
devices, 3.14
errors, D.3
function calling procedure, 3.22
parameter block, 3.30
vectors, 3.18
Bioskeys(), 4.13, 4.24
BITBLK structure, 6.21
bitmaps, see VDI raster forms
Bitstream international character set, G.7
Blitmode(), 4.25
BLiTTER chip, 4.25, 7.9
BOOLEAN, see Data Types
boot sectors, 4.14
break codes, 5.11
BPB, see BIOS parameter block
BSS segment, 2.9
Buffoper(), 4.25
Buffptr(), 4.8, 4.26
bus error, B.4
BYTE, see Data Types
C
C, 1.9
C++, 1.9
caches, 5.3
CACR register, 5.3
camera drivers, see VDI camera drivers
cartridges, 5.7
Cauxin(), 2.34, 2.39
Cauxis(), 2.34, 2.39
Cauxos(), 2.34, 2.40
Cauxout(), 2.34, 2.41
Cconin(), 2.34, 2.41
Cconis(), 2.34, 2.42
Cconos(), 2.34, 2.43
Cconout(), 2.34, 2.43
Cconrs(), 2.34, 2.44
Cconws(), 2.34, 2.45
CD-ROM drives, 2.3, 2.23, 4.12
CHAR, see Data Types
CICON structure, 6.22
CICONBLK structure, 6.22
client, see MiNT pipes
clipboard, see AES scrap library
clipping, see VDI clipping clock, see real time clock
Cnein(), 2.34, 2.46
cold boot, 3.3
colors
bit layout, 4.5, 5.25
mapping, 4.5
proper use of, 11.23
setting, 4.4
using, see VDI using colors
window, 6.160
command line, see GEMDOS command line
con: file, see console device
console device, 2.8, 3.14
context, see MiNT process context
control panel, see XCONTROL
control panel extensions, see XCONTROL CPX’s
controls, user-defined, 6.23, 11.10
conventions, 1.10
cookie jar, 3.8
COOKIE structure, 3.8
determining hardware presence, 3.8
placing a cookie, 3.9
searching for a cookie, 3.8
system cookies, 3.11
coordinate systems, see VDI coordinate systems
coprocessor exceptions, 5.4
coprocessor mode, 5.4
country code, 3.6, 10.6
CPM 68k, 2.3
Cprnos(), 2.34, 2.46
Cprnout(), 2.34, 2.47
CPX, see XCONTROL
cpx_button(), 10.19
cpx_call(), 10.5-10.6, 10.19
cpx_close(), 10.5, 10.20
cpx_draw(), 10.20
cpx_hook(), 10.21
cpx_init(), 10.4, 10.6-10.7, 10.21
cpx_key(), 10.23
cpx_m1(), 10.23
cpx_m2(), 10.24
CPX_Save(), 10.4, 10.29
cpx_timer(), 10.24
cpx_wmove(), 10.25
CPXHEAD structure, 10.3
CPXINFO structure, 10.4
Crawcin(), 2.34, 2.48
Crawio(), 2.34, 2.49
critical error handler, see GEMDOS vectors
crys_if(), 6.39
Cursconf(), 4.13, 4.27
CX-40 joystick, 5.12
D
DAC, 4.6
data cache, see caches
DATA segment, 2.9
data types, 1.11
Dbmsg(), 4.19, 4.28
Dclosedir(), 2.16, 2.50
Dcntl(), 2.16-2.18, 2.50
Dcreate(), 2.4, 2.53
Ddelete(), 2.4, 2.54
debugging, 2.31
debugging keys, see MultiTOS debugging keys
debugging levels, 2.33
deferred vertical blank handlers, 3.19
desk menu, 6.26, 11.15
DESKCICN.RSC file, 9.5
DESKCOPY environment variable, see desktop extensibility
DESKFMT environment variable, see desktop extensibility
DESKICON.RSC file, 9.5
desktop window, see AES
desktop window
desktop, 9.1
drag and drop usage, 9.3
extendibility, 9.3
messages, 9.3
replacing, 9.3
TOS application launching, 9.4
DESKTOP.INF file, 9.4
Devconnect(), 4.7, 4.29
device-specific format, see VDI
device-specific format
device independence, 11.22
Dfree(), 2.4, 2.54
Dgetcwd(), 2.16, 2.56
Dgetdrv(), 2.5, 2.56
Dgetpath(), 2.5, 2.57
diagnostic cartriges, 5.7
diagonals, see AES dialogs
dialogware, 11.3
Digital Research, Inc., 1.3
disk transfer address, see GEMDOS
Dgetleton(), 2.16, 2.57
DMA sound system, see sound
DMAread(), 4.15, 4.31
DMAwrite(), 4.15, 4.32
Dopendir(), 2.16, 2.58
Dpathconf(), 2.16, 2.61
Dreaddir(), 2.16, 2.61
Drewinddir(), 2.16, 2.62
drop-down list boxes, see AES
drop-down list boxes
Drvmap(), 3.30
Dsetdrv(), 2.5, 2.62
Dsetpath(), 2.5, 2.63
DSP, 4.8
connection matrix, 4.7
controller registers, B.36
debugging, 4.11
general-purpose pins, 4.11
ISR register, 4.11
memory map, 4.9
programs, 4.10
sending data, 4.11
state, 4.11
subroutines, 4.9
word size, 4.9
Dsp_Available(), 4.10, 4.34
Dsp_BlkBytes(), 4.11, 4.35
Dsp_BlkHandshake(), 4.11, 4.35
Dsp_BlkUnpacked(), 4.11, 4.36
Dsp_BlkWords(), 4.11, 4.37
Dsp_DoBlock(), 4.11, 4.38
Dsp_ExecuteBoot(), 4.11, 4.39
Dsp_ExecuteProg(), 4.11, 4.39
Dsp_FlushSubroutines(), 4.40
Dsp_GetProgAbility(), 4.11, 4.40
Dsp_GetWordSize(), 4.41
Dsp_Hf0(), 4.11, 4.41
Dsp_Hf1(), 4.11, 4.42
Dsp_Hf2(), 4.11, 4.43
Dsp_Hf3(), 4.11, 4.43
Dsp_Hstat(), 4.11, 4.44
Dsp_InqrSubrAbility(), 4.10, 4.44
Dsp_InStream(), 4.11, 4.45
Dsp_IOSStream(), 4.11, 4.46
Dsp_LoadProg(), 4.11, 4.47
Dsp_LoadSubroutine(), 4.48
Dsp_Lock(), 4.9, 4.48
Dsp_LodToBinary(), 4.11, 4.49
Dsp_MultBlocks(), 4.11, 4.50
Dsp_OutStream(), 4.11, 4.51
Dsp_RemoveInterrupts(), 4.11, 4.51
Dsp_RequestUniqueAbility(), 4.10, 4.52
Dsp_Reserve(), 4.10, 4.53
Dsp_RunSubroutine(), 4.53
Dsp_SetVectors(), 4.11, 4.54
Dsp_TriggerHC(), 4.11, 4.55
Dsp_Unlock(), 4.9, 4.55
Dsptristate(), 4.8, 4.56
DTA, see GEMDOS DTA
dual-state menu items, 11.17
E
edit menu, 11.17
EgetPalette(), 4.5, 4.56
EgetShift(), 4.4, 4.57
enhanced joystick, 5.8
entertainment software, 11.25
Epson printer, 4.96
error codes, D.1
EsetBank(), 4.5, 4.58
EsetColor(), 4.5, 4.59
EsetGray(), 4.4, 4.60
EsetPalette(), 4.5, 4.60
EsetShift(), 4.4, 4.61
EsetSmear(), 4.4, 4.62
EOF, 2.39-2.41
evt_button(), 6.12, 6.61
evt_dclick(), 6.9, 6.62
evt_keybd(), 6.12, 6.63
evt_mesag(), 6.11, 6.64
evt_mouse(), 6.12, 6.70
evt_multi(), 6.10, 6.71
evt_timer(), 6.12, 6.73
EVNTREC structure, 6.57
exception vectors, B.4
expansion area, B.46
EXTEND.SYS file, 7.15
extended partition, see XGM
extension (file), 2.4
F
Falcon030, 1.6
FALSE, see Data Types
FAT, see file allocation table
Fattrib(), 2.6, 2.64
Fchmod(), 2.15, 2.65
Fchown(), 2.15, 2.66
Fclose(), 2.66
Fcntl(), 2.15, 2.67
Fcreate(), 2.6, 2.74
Fdatetime(), 2.7, 2.75
Index

delete(), 2.7, 2.76  
dup(), 2.8, 2.76  
Force(), 2.8, 2.77  
getchar(), 2.15, 2.79  
getdta(), 2.6, 2.79  
file allocation table, 4.14  
file menu, 11.16  
file selector library, see AES file selector library  
file systems, see MiNT loadable file systems  
filenames, see GEMDOS filenames  
fine scrolling, 5.26  
instat(), 2.15, 2.80  
fix31, see Data Types  
link(), 2.15, 2.81  
floating-point coprocessor, 5.4  
floating-point support, see _FPU cookie  
flock system variable, B.8  
lock(), 2.7, 2.82  
floppy drives, 4.15  
Flopfmt(), 4.15, 4.63  
Floprate(), 4.15, 4.65  
Floprd(), 4.15, 4.66  
Flopwr(), 4.15, 4.67  
.FNT file format, C.7  
    character offset table, C.9  
    data, C.8  
    header, C.7  
    horizontal offset table, C.9  
midpipe(), 2.16, 2.83  
folders, see GEMDOS directories  
Font Scaling Module, see FSMGDOS  
FONTGDOS, see GDOS FONTGDOS  
fonts  
in AES objects, 6.20  
    bitmap, see VDI fonts  
    file format, C.7  
    outline, see VDI fonts  
    system, 6.36, 6.48  
open(), 2.84  
form_alert(), 6.25, 6.77  
form_button(), 6.25, 6.78  
form_center(), 6.79  
form_dial(), 6.80  
form_do(), 6.24, 6.81  
form_error(), 6.25, 6.82  
form_keybd(), 6.25, 6.83  
Forth, 1.9  
Foutstat(), 2.15, 2.85  
Fpipe(), 2.27, 2.86  
Putchar(), 2.15, 2.86  
Read(), 2.7, 2.87  
Readlink(), 2.15, 2.88  
Rename(), 2.4, 2.89  
seek(), 2.7, 2.89  
fsel_exinput(), 6.34, 6.87  
fsel_input(), 6.34, 6.88  
Fselect(), 2.15, 2.90  
setdta(), 2.6, 2.91  
first(), 2.5, 2.92  
SMC cookie, 3.13  
FSMGDOS, 7.13  
snext(), 2.5, 2.93  
fsymlink(), 2.15, 2.94  
Write(), 2.6, 2.95  
G  
gadgets, see AES windows  
gain, see sound setting gain  
game controllers, 5.8  
GDOS, 7.11  
    bezier curves, 7.13  
    caching, 7.15  
    camera drivers, 7.17  
    device drivers, 7.16  
    error support, 7.13  
fix31 data type, 7.14  
    font naming convention, 7.12  
    fonts, 7.13  
FSMGDOS, 7.12-7.13  
function calling procedure, see VDI function calling procedure  
kerning, 7.15  
memory driver, 7.18  
matafiles, 7.17  
original, 7.12  
plotter drivers, 7.16  
printer drivers, 7.16  
special effects, 7.15  
SpeedoGDOS, 7.12, 7.14  
Speedo character indexes, 7.15  
tablet drivers, 7.17  
user-defined printer buffer, 7.17  
version, 7.11  
GDP, see VDI GDP’s  
.GEM file format, C.3  
GEM, 1.7  
    partition type, 4.16  
    user interface guidelines, 11.1  
GEM/3, 7.13  
GEM.CNF file, 6.36  
gemdos(), 2.35  
GEMDOS, 2.1  
    ARGV, 2.12  
    application startup, 2.11  
    character functions, 2.34  
    command line, 2.11  
    date functions, 2.35  
    default directory, 2.5  
    default drive, 2.5  
    deleting files, 2.7  
    directories, 2.4  
    drive identifiers, 2.3  
DTA, 2.6  
    environment string, 2.12  
    errors, D.3  
    executable file format, 2.9  
    file attributes, 2.6  
    file handles, 2.7  
    file locking, 2.7  
    file position pointer, 2.7  
    file time/date stamp, 2.7  
    filenames, 2.4  

THE ATARI COMPRENDIUM
<table>
<thead>
<tr>
<th>Index</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>function calling procedure,</strong> 2.35</td>
</tr>
<tr>
<td><strong>path,</strong> 2.5</td>
</tr>
<tr>
<td><strong>processes,</strong> 2.9</td>
</tr>
<tr>
<td><strong>record locking,</strong> 2.7</td>
</tr>
<tr>
<td><strong>redirection,</strong> 2.8</td>
</tr>
<tr>
<td><strong>root directory,</strong> 2.4</td>
</tr>
<tr>
<td><strong>time functions,</strong> 2.35</td>
</tr>
<tr>
<td><strong>the TOS file system,</strong> 2.3</td>
</tr>
<tr>
<td><strong>vectors,</strong> 2.13</td>
</tr>
<tr>
<td><strong>version,</strong> 2.3</td>
</tr>
<tr>
<td><strong>volume label,</strong> 2.6</td>
</tr>
<tr>
<td><strong>GEMFILE.GEM,</strong> 7.17</td>
</tr>
<tr>
<td><strong>generalized device primitives,</strong> see <strong>VDI GDP’s</strong></td>
</tr>
<tr>
<td><strong>GENLOCK,</strong> 4.6</td>
</tr>
<tr>
<td><strong>Get_Buffer()</strong>, 10.29</td>
</tr>
<tr>
<td><strong>Getbpb()</strong>, 3.15, 3.30</td>
</tr>
<tr>
<td><strong>getcookie()</strong>, 10.30</td>
</tr>
<tr>
<td><strong>GetFirstRect()</strong>, 10.30</td>
</tr>
<tr>
<td><strong>Getmpb()</strong>, 3.31</td>
</tr>
<tr>
<td><strong>GetNextRect()</strong>, 10.31</td>
</tr>
<tr>
<td><strong>Getrez()</strong>, 4.4, 4.68</td>
</tr>
<tr>
<td><strong>Gettime()</strong>, 4.18, 4.69</td>
</tr>
<tr>
<td><strong>Giaccess()</strong>, 4.70, 1.3</td>
</tr>
<tr>
<td><strong>Gpio()</strong>, 4.8, 4.72</td>
</tr>
<tr>
<td><strong>graf_dragbox()</strong>, 6.34, 6.91</td>
</tr>
<tr>
<td><strong>graf_drawbox()</strong>, 6.34, 6.92</td>
</tr>
<tr>
<td><strong>graf_handle()</strong>, 6.34, 6.92, 7.3</td>
</tr>
<tr>
<td><strong>graf_mkstate()</strong>, 6.34, 6.93</td>
</tr>
<tr>
<td><strong>graf_mouse()</strong>, 6.34, 6.94</td>
</tr>
<tr>
<td><strong>graf_movebox()</strong>, 6.34, 6.96</td>
</tr>
<tr>
<td><strong>graf_rubberbox()</strong>, 6.34, 6.97</td>
</tr>
<tr>
<td><strong>graf_shrinkbox()</strong>, 6.34, 6.98</td>
</tr>
<tr>
<td><strong>graf_slidebox()</strong>, 6.34, 6.99</td>
</tr>
<tr>
<td><strong>graf_watchbox()</strong>, 6.34, 6.100</td>
</tr>
<tr>
<td><strong>graphics library,</strong> see <strong>AES graphics library</strong></td>
</tr>
<tr>
<td><strong>grayscale mode,</strong> 4.4</td>
</tr>
<tr>
<td><strong>GRECT structure,</strong> 7.7</td>
</tr>
<tr>
<td><strong>H</strong></td>
</tr>
<tr>
<td><strong>handles,</strong> see <strong>GEMDOS file handles or VDI</strong></td>
</tr>
<tr>
<td><strong>workstation handles</strong></td>
</tr>
<tr>
<td><strong>hierarchical menus,</strong> 6.27, 11.20</td>
</tr>
<tr>
<td><strong>I</strong></td>
</tr>
<tr>
<td><strong>icon,</strong> 6.21</td>
</tr>
<tr>
<td><strong>color,</strong> 6.22</td>
</tr>
<tr>
<td><strong>ICONBLK structure,</strong> 6.21</td>
</tr>
<tr>
<td><strong>iconification,</strong> 6.30, 6.156, 11.7</td>
</tr>
<tr>
<td><strong>IKBD,</strong> 5.10</td>
</tr>
<tr>
<td><strong>commands,</strong> 5.14</td>
</tr>
<tr>
<td><strong>scan codes,</strong> F.1</td>
</tr>
<tr>
<td><strong>Ikbdwsv(),</strong> 4.14, 4.72</td>
</tr>
<tr>
<td><strong>Imagen,</strong> see <strong>QMS/Imagen</strong></td>
</tr>
<tr>
<td><strong>.IMG file format,</strong> C.5</td>
</tr>
<tr>
<td><strong>extra palette information,</strong> C.5</td>
</tr>
<tr>
<td><strong>header,</strong> C.5</td>
</tr>
<tr>
<td><strong>image compression,</strong> C.6</td>
</tr>
<tr>
<td><strong>image data format,</strong> C.6</td>
</tr>
<tr>
<td><strong>Initmous(),</strong> 4.12, 4.73</td>
</tr>
<tr>
<td><strong>instruction cache,</strong> 5.3</td>
</tr>
<tr>
<td><strong>interrupt priority level,</strong> 5.3</td>
</tr>
<tr>
<td><strong>IOREC structure,</strong> 4.75</td>
</tr>
<tr>
<td><strong>Iorec(),</strong> 4.17, 4.75</td>
</tr>
<tr>
<td><strong>J</strong></td>
</tr>
<tr>
<td><strong>Jdisint(),</strong> 4.18, 4.76</td>
</tr>
<tr>
<td><strong>Jenabint(),</strong> 4.18, 4.76</td>
</tr>
<tr>
<td><strong>joysticks,</strong> 5.8, 5.12</td>
</tr>
<tr>
<td><strong>K</strong></td>
</tr>
<tr>
<td><strong>Kbdvbase(),</strong> 4.77</td>
</tr>
<tr>
<td><strong>KBDVECS,</strong> 4.77</td>
</tr>
<tr>
<td><strong>Kbrate(),</strong> 4.13, 4.78</td>
</tr>
<tr>
<td><strong>Kbshift(),</strong> 3.7, 3.32</td>
</tr>
<tr>
<td><strong>kerning,</strong> see <strong>GDOS kerning</strong></td>
</tr>
<tr>
<td><strong>keyboard,</strong> 5.11</td>
</tr>
<tr>
<td><strong>keyboard equivalents,</strong> 11.20</td>
</tr>
<tr>
<td><strong>keyboard tables,</strong> 4.12, 7.15, E.1, F.1</td>
</tr>
<tr>
<td><strong>Keytbl(),</strong> 4.12, 4.78</td>
</tr>
<tr>
<td><strong>KEYTBL.TBL file,</strong> 4.13</td>
</tr>
<tr>
<td><strong>L</strong></td>
</tr>
<tr>
<td><strong>LAN connector,</strong> 4.17</td>
</tr>
<tr>
<td><strong>Lattice C,</strong> 1.9</td>
</tr>
<tr>
<td><strong>light gun,</strong> 5.10</td>
</tr>
<tr>
<td><strong>Line-A,</strong> 8.1</td>
</tr>
<tr>
<td><strong>arbitrary line function,</strong> 8.12</td>
</tr>
<tr>
<td><strong>bitblt function,</strong> 8.15</td>
</tr>
<tr>
<td><strong>copy raster function,</strong> 8.21</td>
</tr>
<tr>
<td><strong>draw sprite function,</strong> 8.20</td>
</tr>
<tr>
<td><strong>filled polygon function,</strong> 8.14</td>
</tr>
<tr>
<td><strong>filled rectangle function,</strong> 8.13</td>
</tr>
<tr>
<td><strong>font headers,</strong> 8.7</td>
</tr>
<tr>
<td><strong>function calling procedure,</strong> 8.8</td>
</tr>
<tr>
<td><strong>get pixel function,</strong> 8.12</td>
</tr>
<tr>
<td><strong>hide mouse function,</strong> 8.19</td>
</tr>
<tr>
<td><strong>horizontal line function,</strong> 8.13</td>
</tr>
<tr>
<td><strong>initialize function,</strong> 8.11</td>
</tr>
<tr>
<td><strong>plot pixel function,</strong> 8.11</td>
</tr>
<tr>
<td><strong>seed fill function,</strong> 8.22</td>
</tr>
<tr>
<td><strong>show mouse function,</strong> 8.18</td>
</tr>
<tr>
<td><strong>textblt function,</strong> 8.16</td>
</tr>
<tr>
<td><strong>transform mouse function,</strong> 8.19</td>
</tr>
<tr>
<td><strong>undraw sprite function,</strong> 8.20</td>
</tr>
<tr>
<td><strong>variable table,</strong> 8.3</td>
</tr>
<tr>
<td><strong>links,</strong> see <strong>MiNT links</strong></td>
</tr>
<tr>
<td><strong>list boxes,</strong> see <strong>AES drop-down list boxes</strong></td>
</tr>
<tr>
<td><strong>Localtalk,</strong> see <strong>LAN connector</strong></td>
</tr>
<tr>
<td><strong>Locksnd(),</strong> 4.6, 4.79</td>
</tr>
<tr>
<td><strong>Logbase(),</strong> 4.3, 4.80</td>
</tr>
<tr>
<td><strong>logical screen,</strong> 4.3</td>
</tr>
<tr>
<td><strong>M</strong></td>
</tr>
<tr>
<td><strong>magneto-optical drives,</strong> 2.3</td>
</tr>
<tr>
<td><strong>Maddalt(),</strong> 2.97</td>
</tr>
<tr>
<td><strong>make codes,</strong> 5.11</td>
</tr>
<tr>
<td><strong>Malloc(),</strong> 2.8, 2.98</td>
</tr>
<tr>
<td><strong>MAPTAB structure,</strong> 4.24</td>
</tr>
<tr>
<td><strong>matrix,</strong> see <strong>sound connection matrix</strong></td>
</tr>
<tr>
<td><strong>media change,</strong> 3.15</td>
</tr>
<tr>
<td><strong>Mediach(),</strong> 3.15, 3.33</td>
</tr>
<tr>
<td><strong>Mega ST,</strong> 1.4</td>
</tr>
<tr>
<td><strong>Mega STe,</strong> 1.4</td>
</tr>
<tr>
<td><strong>memory driver,</strong> see <strong>GDOS memory driver</strong></td>
</tr>
<tr>
<td><strong>memory initialization,</strong> 3.3</td>
</tr>
<tr>
<td><strong>memory management unit,</strong> B.5</td>
</tr>
<tr>
<td><strong>memory map,</strong> B.1</td>
</tr>
<tr>
<td><strong>memory protection,</strong> 2.14</td>
</tr>
</tbody>
</table>
Index

memory types, 2.8
memory usage parameter block, 3.31
MEMORY.SYS, see GDOS memory driver
MENU structure, 6.103
menu buffer, see AES menu buffer
menu_attach(), 6.27, 6.103
menu_bar(), 6.27, 6.105
menu_ichck(), 6.27, 6.106
menu_ienable(), 6.27, 6.106
menu_istart(), 6.27, 6.107
menu_popup(), 6.28, 6.108
menu_register(), 6.4, 6.7, 6.109
menu_settings(), 6.28, 6.110
menu_text(), 6.27, 6.111
menus, see AES menus
messages, see AES messages
META.SYS driver, see GDOS metafiles
METADOS, 4.12
metafiles
creating, see GDOS metafiles
header, C.3
records, C.4
sub-opcodes, C.4
METAINFO structure, 4.80
Metainit(), 4.12, 4.80
MFDB structure, 7.119
MFORM structure, 6.95
MFsave(), 10.31
MFP, B.5
configuration, 4.17
interrupts, 4.18
ST port registers, B.37
TT port registers, B.41
vectors, B.5
Mfpint(), 4.18, 4.81
Mfree(), 2.99
MICROWIRE interface, 5.22
MIDI, 3.14, 5.10
Midisz(), 4.19, 4.82
MiNT, 2.14
access permissions, 2.14
cookie, 3.13
debugging, 2.31
default directory, 2.16
DEV directory, 2.17
directory enumeration, 2.16
exit codes, 2.14
file attributes, 2.15
file ownership, 2.15
file status, 2.15
file system extensions, 2.15
function calling procedure, see GEMDOS function calling procedure
hard links, 2.15
interprocess communication, 2.27
links, 2.15
loadable devices, 2.17
loadable file systems, 2.23
messages, 2.31
MINT.CNF file, 2.33
PIPE directory, 2.27
pipes, 2.27
PROC directory, 2.16
process attributes, 2.17
process context, 2.32
process identifier, 2.14
process priority, 2.14
processes, 2.14
pseudo-drives, 2.16
resources, 2.14
semaphores, 2.31
shared memory, 2.30
SHM directory, 2.30
signals, 2.28
symbolic links, 2.15
threads, 2.14
timeslice, 2.14
tracing, 2.31
user-defined longword, 2.14
MN_SET structure, 6.110
MODICOLORWORD structure, 6.18
objects, 6.13
colorword, 6.18
flags, 6.16
fonts, 6.20
ob_spec, 6.18
states, 6.17
structure, 6.15
types, 6.15
Offgibit(), 4.17, 4.84
Onegibit(), 4.17, 4.84
ORU’s, G.3

MIDI, 3.14

MPB, see memory usage parameter block
Mshrink(), 2.11, 2.99
MS-DOS, 2.3
multi-function peripheral port, see MFP
MultiTOS, 2.3
debugging keys, 2.32
Mxalloc(), 2.8, 2.100

N
NDC, see VDI coordinate systems
NEWDESK.INF file, 9.4
non-maskable interrupt, 5.3
non-volatile RAM, see NVMaccess()
normalized device coordinates, see VDI coordinate systems
NULL device, 2.17
NVMaccess(), 4.18, 4.83

O
objc_add(), 6.14, 6.115
objc_change(), 6.17, 6.115
objc_delete(), 6.14, 6.116
objc_draw(), 6.117
objc_edit(), 6.25, 6.118
objc_find(), 6.14, 6.119
objc_offset(), 6.14, 6.120
objc_order(), 6.14, 6.121
objc_sysvar(), 6.122
OBJC_COLORWORD structure, 6.18
objects, 6.13
colorword, 6.18
flags, 6.16
fonts, 6.20
ob_spec, 6.18
states, 6.17
structure, 6.15
types, 6.15
Offgibit(), 4.17, 4.84
Onegibit(), 4.17, 4.84
ORU’s, G.3
OS, 1.6
overlay mode, see VsetMask()

P
p_cookies, see cookie jar
p_kbshift, 3.7
p_root, 3.5
p_run, 3.7
paddles, 5.9
page flipping, 4.3
palette, see VDI palette based devices
palette registers, 4.4
PARMBLK structure, 6.23
partition information block, 4.16
Pascal, 1.9
PATH environment variable, 6.9
Pause(), 2.101
Pdomain(), 2.3, 2.102
peripheral mode, 5.4
Pexec(), 2.9, 2.103
Pfork(), 2.14, 2.105
Physbase(), 4.4, 4.85
physical screen, 4.3
Pgetegid(), 2.14, 2.106
Pgeteuid(), 2.14, 2.106
Pgetgid(), 2.14, 2.107
Pgetpgrp(), 2.14, 2.107
Pgetpgrp(), 2.14, 2.107
Pgetpgrp(), 2.14, 2.108
Pgetuid(), 2.14, 2.108
Pkill(), 2.109
plotter drivers, see VDI plotter drivers
Pmsg(), 2.31, 2.109
Pnice(), 2.14, 2.111
Popup(), 10.32
popup menus, 6.28, 11.18
Prenice(), 2.14, 2.111
prescaler, 4.7
PRGFLAGS, 2.9-2.10
printer, 4.18
printer device, 2.8, 2.17
printer drivers, see VDI printer drivers
prn: file, see printer device
process terminate handler, see GEMDOS vectors
processor cache control, 5.3
MegaSTe, B.34
processor state save area, B.7
progress indicators, 11.12
Protbt(), 4.15, 4.86
prt_cnt(), B.12
PRTBLK structure, 4.87
Prtblk(), 4.18, 4.87
Prusage(), 2.14, 2.112
Psemaphore(), 2.31, 2.113
Psetgid(), 2.14, 2.114
Psetlimit(), 2.14, 2.114
Psetpgprp(), 2.14, 2.115
Psetuid(), 2.14, 2.116
pseudo-drive, 2.16
PSG, I.1
Psigaction(), 2.28, 2.116
Psigblock(), 2.28, 2.118
Psig.signal(), 2.28, 2.118
Psigpause(), 2.28, 2.119
Psigpending(), 2.28, 2.120
Psigreturn(), 2.28, 2.120
Psigsetmask(), 2.28, 2.121
Pterm(), 2.9, 2.11, 2.121
PtermO(), 2.11, 2.122
Ptermres(), 2.11, 2.123
PTRACEFLOW, 2.31
PTRACEEGO, 2.31
PTRACEFLAGS, 2.31
PTRACESTEP, 2.31
Pumask(), 2.16, 2.123
Puntaes(), 3.7, 4.19, 4.88
Pusrval(), 2.14, 2.124
Pvfork(), 2.14, 2.124
Pwait(), 2.14, 2.125
Pwait3(), 2.14, 2.126
Pwaitpid(), 2.14, 2.127
Q
QMS/Imagen, 7.13
raster coordinates, see VDI coordinate systems
raster forms, see VDI raster forms
RC, see VDI coordinate systems
RCS, see resource construction set
real-time clock, B.31
rectangle list, see AES rectangle list
rectangles, see VDI rectangles
reset vector, see BIOS vectors
resolutions, see screen
resource construction set, 6.13
resources, 6.13
file format, see .RSC file format
usage, see AES resource library
ROOT definition, 6.14
.RSC file format, C.9
CICONBLK extension, C.9
Cextension array, C.11
free strings and images, C.11
header, C.9
object trees, C.10
AES 3.30 resource format, C.11
Rsconf(), 4.17, 4.89
rsh_fix(), 10.33
rsh_obfix(), 10.34
rsrc_free(), 6.127
rsnc_gaddr(), 6.13, 6.127
rsnc_load(), 6.7, 6.13, 6.128
rsnc_obfix(), 6.13, 6.129
rsnc_rcfix(), 6.13, 6.130
rsnc_saddr(), 6.13, 6.130
Rwabs(), 3.34
S
Salert(), 2.28, 2.128
SBUF PTR, 4.26
scan codes, F.1
SCC, 4.17
DMA registers, B.33
ports, B.33
vectors, B.6

scr_dump, B.14

scrap library, see AES scrap library

Scrdmp(), 4.18, 4.91

screen
determining the size, 4.4
memory, 4.3, 5.25
registers, B.19
resolution, 4.4, 5.24
resolution change, 6.144

scrp_read(), 6.34, 6.135

scrp_write(), 6.34, 6.136

SCSI, 4.15

semaphores, see MiNT semaphores

serial device, 2.8
serial number, 4.14
serial port, 4.16
mapping, 4.17
server, see MiNT pipes

Set_Event_Mask(), 10.34

Setbuffer(), 4.7, 4.92

Setcolor(), 4.4, 4.93

Setexec(), 3.20, 3.35

Setinterrupts(), 4.8, 4.93

Setmode(), 4.7, 4.94

Setmontracks(), 4.8, 4.95

Setpalette(), 4.3, 4.97

Setprt(), 4.18, 4.96

Setscreen(), 4.3, 4.97

Settime(), 4.18, 4.98

Settracks(), 4.8, 4.99

shadow image, B.46

shel_envrn(), 6.9, 6.139

shel_find(), 6.36, 6.139

shel_get(), 6.35, 6.140

shel_put(), 6.35, 6.141

shel_read(), 6.36, 6.141

shel_write(), 2.13, 6.9, 6.36, 6.142

shell buffer, see AES shell buffer

shell, see AES shell library

shift keys, 3.7, 3.32

signals, see MiNT signals

g督导, 6.4, 6.7

SL_Arrow(), 10.35

SL_dragx(), 10.36

SL_dragy(), 10.36

SL_size(), 10.37

SL_x(), 10.37

SL_y(), 10.38

slider bar, 6.30

SLM804, 7.16

SMALLER gadget, 6.30

smear mode, 4.4

Sndstatus(), 4.8, 4.99

sound

attenuation, 4.8

adjusting gain, 4.8

configuring levels, 4.8

connection matrix, 4.7

determining status, 4.8

envelopes, I.6

Falcon030 sound system, 4.6

FM, I.3

handshaking, 4.7

interrupts, 4.8

playing, I.1

proper use of, 11.24

recording, 4.8

registers, B.25-B.26

selecting tracks, 4.8

setting frequency, 4.7

STe/TT digital sound, 5.28

Soundcmd(), 4.7, 4.100

SpeedoGDOS, 7.14

ciphertext set, G.7

font header, G.3

Ssbrk(), 4.19, 4.102

ST, 1.3

ST Book, 1.5

ST RAM, see memory types

Stacy, 1.3

stack allocation, 6.5

standard format, 7.9

standard RAM, see memory types

submenu, see hierarchical menus

Super(), 2.128

supervisor mode, 2.128, 4.12, 4.103

Supexec(), 4.12, 4.103

Sversion(), 2.3, 2.129

Syield(), 2.130

symbol table, 2.10

_sysbase, 3.4

Sysconf(), 2.130

system boot variables, B.4

system font, 6.36, 6.48

system bell vector, see BIOS vectors

system control unit, B.34

system keyclick vector, see BIOS vectors

system RAM, B.16

system startup, 3.3

system vectors, B.7

T

tablet drivers, see VDI tablet drivers

Talarm(), 2.131

TEDINFO structure, 6.19

terminal device, 2.17

TEXT segment, 2.9

Tgetdate(), 2.35, 2.132

Tgettime(), 2.35, 2.132

threads, see MiNT threads

three-dimensional objects, 6.16-6.17

Tickcal(), 3.36

timer, see AES timer events

timer tick vector, see GEMDOS vectors

toolbars, 6.33, 11.14

toolboxes, 11.13

TOS, 1.3

configuration bits, 3.6

file system, 2.3

header, 3.4

OSHEADER structure, 3.5

TOSRUN pipe, 9.4

TPA, see transient program area

tracing, see MiNT tracing
transient program area, 2.11
TRAP exception vectors, B.4
TRUE, see Data Types
true-color, see VDI true-color devices
toolbars, see AES window toolbars
Tsetdate(), 2.35, 2.133
Tsettime(), 2.35, 2.133
TT RAM, see memory types
TT030, 1.5
TTY, see terminal device
typesetting, 1.10
UBYTE, see Data Types
UCHAR, see Data Types
ULONG, see Data Types
UNIX, 2.3
Unlocksnd(), 4.6, 4.103
user interface, 11.1
user mode, 4.12
UWORD, see Data Types
v_alpha_text(), 7.23
v_arc(), 7.24
v_bar(), 7.25
v_bez(), 7.13, 7.26
v_bez_fill(), 7.13, 7.27
v_bez_off(), 7.13, 7.28
v_bez_on(), 7.13, 7.29
v_bez_qual(), 7.30
v_bit_image(), 7.31
v_cellarray(), 7.32
v_circle(), 7.33
v_clear_disp_list(), 7.34
v_clrwk(), 7.34
v_clsvwk(), 7.35
data types
v_clswk(), 7.35
v_contourfill(), 7.36
v_curdown(), 7.37
v_curhome(), 7.37
v_curlft(), 7.38
v_curleft(), 7.38
v_curtxt(), 7.38
v_curup(), 7.40
v_dspcur(), 7.40
v_eel(), 7.41
v_eeos(), 7.42
v_ellarc(), 7.42
v_ellipse(), 7.43
v_ellipse(), 7.44
v_enter_cur(), 7.45
v_exit_cur(), 7.46
v_fillarea(), 7.46
v_flushcache(), 7.47
v_fontinit(), 7.48
v_form_adv(), 7.48
v_ftext(), 7.49
v_ftext16(), 7.16, 7.50
v_ftext_offset(), 7.51
v_ftext_offset16(), 7.16, 7.52
v_get_pixel(), 4.5, 7.55
v_getbitmap_info(), 7.12, 7.14, 7.53
v_getoutline(), 7.12, 7.54
v_gtext(), 7.56
v_hardcopy(), 7.57
v_hide_c(), 7.57
v_justified(), 7.58
v_killoutline(), 7.12, 7.59
v_loadcache(), 7.59
v_meta_extents(), 7.60
v_opnvwk(), 7.3, 7.61
v_opnvwk(), 7.5, 7.65
v_opnwk(), 7.3, 7.66
v_opnwk(), 7.5, 7.67
v_output_window(), 7.68
v_pcount(), 7.69
v_pieslice(), 7.70
v_pie_slice(), 7.70
v_pline(), 7.71
v_pmarker(), 7.72
v_rbox(), 7.72
v_rfbox(), 7.73
v_rmcur(), 7.74
v_rvoff(), 7.75
v_rvoff(), 7.75
v_savecache(), 7.76
v_set_app_buff(), 7.77
v_show_c(), 7.77
v_updwk(), 7.16, 7.78
v_write_meta(), 7.79
validation string, 6.19
VDI, 7.1
circling, 7.3, 7.125
color mapping, 7.9
device IDs, 7.4
device-specific format, 7.10
devices, see GDOS device
drawing, see GDOS device
drawing functions, 7.8
function availability, 7.8
function calling procedure, 7.18
function reference, 7.21
GDOS, see GDOS
GDOS parameters, 7.6
monochrome devices, 7.9
raster forms, 7.9
rectangles, 7.7
rendering graphics, 7.6
palette-based devices, 7.9
parameter block, 7.18
physical workstations, 7.3
standard format, 7.10
toolbars, see AES window toolbars
true-color devices, 7.9
using color, 7.8
vector handling, 7.10
virtual workstations, 7.4
workstations, 7.3
workstation handles, 7.3
write modes, 7.8, 7.162
VDI_Workstation structure, 7.65
vertical blank
handlers, 3.19
interrupt, 3.19
vex_butv(), 7.10, 7.80
vex_curv(), 7.10, 7.81
vex_motv(), 7.10, 7.82
vex_timv(), 7.10, 7.83
VgetMonitor(), 4.4, 4.104
VgetRGB(), 4.6, 4.104
VgetSize(), 4.4, 4.105
video control, 4.3
video registers, B.19
video mode, see screen
vm_coords(), 7.17, 7.83
vm_filename(), 7.17, 7.84
vm_pagesize(), 7.17, 7.85
VOID, see Data Types
VOIDP, see Data Types
VOIDPP, see Data Types
volume label, see GEMDOS
volume label
vq_cellarray(), 7.86
vq_chcells(), 7.87
vq_color(), 7.88
vq_curaddress(), 7.89
vq_extnd(), 7.8, 7.89
vq_gdos(), 7.11, 7.92
vq_key_s(), 7.93
vq_mouse(), 7.93
vq_scan(), 7.94
vq_tabstatus(), 7.95
vq_tdimensions(), 7.96
vqf_attributes(), 7.96
vqm_attributes(), 7.97
vqp_error(), 7.100
vqp_films(), 7.101
vqp_state(), 7.101
vqt_advance(), 7.102
vqt_advance32(), 7.14, 7.103
vqt_attributes(), 7.104
vqt_cachesize(), 7.15, 7.105
vqt_devinfo(), 7.106
vqt_extent(), 7.107
vqt_f_extent(), 7.108
vqt_f_extent16(), 7.109
vqt_formheader(), 7.12, 7.110
vqt_fontinfo(), 7.111
vqt_get_table(), 7.12, 7.15, 7.112
vqt_name(), 7.16, 7.113
vqt_pairkern(), 7.12, 7.15, 7.114
vqt_trackkern(), 7.12, 7.15, 7.115
vqt_width(), 7.115
vr_recfl(), 7.117
vr_trnfm(), 4.5, 7.9, 7.117
vro_cpyfm(), 7.8-7.9, 7.117
vrq_choice(), 7.121
vrq_locator(), 7.121
vrq_string(), 7.122
vrq_valuator(), 7.123
vrt_cpyfm(), 7.9, 7.124
vs_clip(), 7.125
vs_color(), 7.126
vs_curaddress(), 7.126
vs_palette(), 7.127
vsc_form(), 7.128
VsetMask(), 4.6, 4.106
VsetMode(), 4.4, 4.107
VsetRGB(), 4.6, 4.108
VsetScreen(), 4.108
VsetSync(), 4.6, 4.109
vsf_color(), 7.129
vsf_interior(), 7.129
vsf_style(), 7.131
vsf_udpat(), 7.132
vsin_mode(), 7.133
vsl_color(), 7.134
vsl_ends(), 7.134
vsl_type(), 7.135
vsm_height(), 7.136
vsm_locator(), 7.137
vsm_type(), 7.138
vsm_valuator(), 7.138
vsm_choice(), 7.139
vsm_color(), 7.139
vsm_height(), 7.139
vsm_locator(), 7.140
vsm_string(), 7.141
vsm_type(), 7.142
vsmvaluator(), 7.143
vsp_message(), 7.144
vsp_save(), 7.145
vsp_state(), 7.145
vst_alignment(), 7.146
vst_arbpt(), 7.147
vst_arbpt32(), 7.14, 7.148
vst_charmap(), 7.15, 7.149
vst_color(), 7.150
vst_effects(), 7.150
vst_error(), 7.13, 7.151
vst_font(), 7.152
vst_height(), 7.153
vst_kern(), 7.12, 7.15, 7.154
vst_load_fonts(), 7.13, 7.155
vst_point(), 7.155
vst_rotation(), 7.156
vst_scratch(), 7.15, 7.157
vst_setsize(), 7.158
vst_setsize32(), 7.14, 7.159
vst_skew(), 7.160
vst_unload_fonts(), 7.161
vswr_mode(), 7.8, 7.162
Vsync(), 4.110
VT-52 emulator, 3.14
vt_alignment(), 7.163
vt_axis(), 7.164
vt_origin(), 7.164
vt_resolution(), 7.165
W
warm boot, 3.3
WavePlay(), 4.110
wildcards, 2.5
wind_calc(), 6.33, 6.149
wind_close(), 6.31, 6.150
wind_create(), 6.29, 6.150
wind_delete(), 6.31, 6.152
wind_find(), 6.31, 6.152
wind_get(), 6.31, 6.153
wind_new(), 6.157
wind_open(), 6.31, 6.158
wind_set(), 6.31, 6.158
wind_update(), 6.32, 6.161
windows, see AES windows
WORD, see Data Types
workstations, see VDI workstations
WORM drives, 2.3
write modes, see VDI write modes
WYSIWYG, 7.14
X
XBIOS, 4.1
calling from an interrupt, 4.20
function calling procedure, 4.19

Xbtimer(), 4.113

XCPB structure, 10.5

XCONTROL, 10.1
  boot-only CPX’s, 10.6
  callback functions, 10.17
  cpx flavors, 10.6
  CPX types, 10.6
  event CPX’s, 10.9
  executable format, 10.3
  file formats, 10.12
  file naming, 10.12
  form CPX’s, 10.6
  function calling procedure, 10.13
  function reference, 10.15
  parameter block, 10.5
  resident CPX’s, 10.7
  set-only CPX’s, 10.7
  stack space, 10.13
  utility functions, 10.27

Xform_do(), 10.38

XGen_Alert(), 10.39

XGM partition, 4.16