The Hitchhiker's Guide to GEOS

v2020

A Potpourri of Technical Programming Notes
(provided "as is" without support)

April 1988

Heavily Revised for Digital Medium 2020

This work is in an Alpha stage.
The Goal of this Document is to provide a one stop resource for GEOS programming information.

1. Convert to Fully Indexed Digital Form:
   The Hitchhikers Guide to GEOS
   by Berkely Softworks 1988
   Note: all Apple Information will be removed from this conversion. If I get geoAssembler ported into the Apple GEOS, there will be another document made from this one with all the apple information in it. Until then the lack of development tools for Apple lead to an early death of GEOS in that environment and its inclusion here is of no value to a CBM GEOS developer.

2. Combine additional information from other sources including.
   A. Geos Programmer's Reference Guide by
      by Alexander Donald Boyce 1986
      Revised by Bo Zimmerman 1997

   B. The Official GEOS Programmers Reference Guide
      by Berkeley Softworks 1987

   C. Information now available from the Dissembled GEOS Kernal

   D. Information Obtained from my Disassembly of GEOS Applications.

3. Include API Information for Wheels 4.4
   Note: Thanks to "THE" email chain collected by Bo Zimmerman, *there is some original author source for documentation. In addition more information will be extracted from the dissembled sources of both the Wheels kernal and of wheels applications.*
   Major find by
4. Include API Information for MP3+
   Note: MP3 is still being actively developed and it is open source so this should not be a problem.
5. Add Tutorials for at least the following.
   a. creating Auto-Exec applications. With all of the special restrictions outlined.
   b. creating Desk Accessories. With all of the special restrictions outlined.
   c. creating VLIR applications. With fully functioning Module Management outlined.
6. Include geoProgrammer Manual content so it can benefit with hotlinks into the GEOS API and examples.
7. Add any and all other relevant information from others sources including from my own experience developing for GEOS. With appropriate credits given for all source Documents

Volunteers are welcome to assist and credits will be given...
My goal is too add a minimum of 1 page a day until this document it is completed.

Paul B Murdaugh
Writer of Dual Top and the Landmark Series for GEOS
paulbmurdaugh@gmail.com

Preface:
Why this document.

I want to see GEOS not only continue to exists, but to have a newly growing user base. This is only possible with new development. New development needs new programmers that have been attracted into the system.

After some time getting back into GEOS programming myself I quickly decided the bugs and limitations of geoProgrammer that I used to deal with were no longer acceptable. I did not find concept to be a solution since it has many of the same limitations and problems as geoProgrammer with only a small set of bugs fixed. It also forced you into only developing while running wheels and into using is "little mini desktop" to work in. None of that worked for me and I assumed it could be an issue for others. While I do appreciate the good parts of concept it was simply not for me.

Next I tried CC65. CC65 is great achievement imho. Wrote a couple of small applications using it. Then discovered it does not do Desk Accessories and the author has posted a comment about not seeing any point in fixing that. Combine that with me not really enjoying developing outside of GEOS and not being able to use geoDebugger and it was pretty easy to say goodbye to CC65 as my development platform. While being a great piece of work, CC65 was just not for me.

That left me with patching problems in geoProgrammer... Which grew into a complete upgrade with major portions of the geoProgrammer programs being rewritten. During this process I of course needed reference material to work with as I was reverse engineering and writing new code. What I found was lots of OCR scanned documents and some good web resources.

I was constantly going between these different sources. HHG2G was the most complete and by far the hardest to search as the OCR is pretty rough. There is a really nice web resource for data but has no API at all. Plus I have no ability to make notes or additions into someone else's web site. GEOS Programmer’s Reference Guide was fairly easy to search as it did not have OCR issues but it was not complete by HHG2G standards and had other issues I wanted corrected.

My first effort was just turning the Geos Programmer’s Reference Guide to text so I could make changes and notes as needed. Intended only for my own private use. I would add a new section everything I had to reference something I didn't already have converted in. As this started to grow it started morphing into the HHG2G as that was where all the real meat from Berkeley themselves is.

When it became obvious to me the end result was going to be a full digital conversion of HHG2G it had to be something that got shared so new programmers could just jump right in and not have any of the starting hurdles I had to deal with. Then came prereleases so feedback from the community could help to shape the final form. With the end goal be a complete but living document covering all aspects of GEOS development. From this document it would be very quick base for translation into other languages. Conversion to Web form for a WIKI type structure , brought into a database for easy report generation on the fly for any type of content. The data in these old OCR scans has been set free.

So it is my hope that with a new set of development tools provided in geoProgrammer'2.0 that works inside of any version of GEOS from 1.3 up to Wheels 4.4 / MP3 combined with documentation that is indexed and cross referenced will make it a more enjoyable for programmers just discovering GEOS for the first time.

Sources

	Hitchhikers Guide to GEOS 1988	Base Source Document
1	Geos Programmer's Reference Guide 1986/1997	Secondary documentation source. notes with a superscript 1 (^{1}) are from this document.
2	The Official GEOS™ Programmer's Reference Guide	Adding Content that is not already covered in HHG2G and GPRG
3	Paul B Murdaugh (author). Personal experience, Information learned from the rewrite / upgrades of geoProgrammer applications combined with discoveries from reverse engineering Berkeley applications.	Additional Content /changes made by me get a (^{3}). Example note[3]: Also any content related to geoProgrammer' 2.0 is my original content.
4	Scott Hutter. Data Mining	Found great extinct resource on additional Wheels documentation from 2002ish era. All of Kernal Group 0 was documented from his findings.
	Additional Sources to be added as used	

Contributors

1	Bo Zimmerman	Geos Programmer's Reference Guide 1997 wheels documentation collection via Email with Wheels Author
2	Scott Hutter	Data Mining Proof Reading
3	Facebook group GEOS - Wheels - GeoWorks - MegaPatch - gateWay	General Feedback and a place for me distrubute this document
	Additional Contributors to be added as needed	

Introduction

In 1986, Berkeley Softworks pioneered GEOS — the Graphic Environment Operating System — for the Commodore 64. GEOS offered the power of an icon/windowing operating system, once thought only possible on the likes of Apple's Macintosh, to one of the world's lowest priced microcomputers. The computing community quickly recognized this innovation as significant: the Software Publisher's Association (SPA) gave GEOS a Technical Achievement Award and Commodore Business Machines endorsed it as the official operating system for the Commodore 64. Some industry critics even said it brought the Commodore 64 out of obsolescence. Since that time, GEOS has been ported to the Commodore 128 and, most recently, to the Apple II family of computers. Boasting an installed base approaching one-million units, GEOS not only promises to be around for some time, but to grow into the operating system for low-end computers

Why Develop GEOS Applications

GEOS provides an environment for programmers and software companies to quickly and efficiently develop sophisticated applications. GEOS insulates the programmer from the frustrating details and dirty work usually associated with application development. By using the GEOS facilities for disk file handling, screen graphics, menus, icons, dialog boxes, printer and input device support, the application can concentrate on doing what it does best, applying itself to the task at hand, using the GEOS system resources, routines, and user-interface facilities to both speed program development and build better programs.

Consistent User-interface

A very large portion of GEOS is devoted to supporting the user-interface. The GEOS interface has proven popular with thousands of users, and an application that takes advantage of this will likely be well received because the users will already be familiar with the basic program operation. Once a user has learned to operate geoWrite, for example, it is a smooth transition to another application such as geoCalc.

Large Installed Base and Portability

GEOS is currently available for three machines: the Commodore 64, the Commodore 128, and the Apple II. There are hundreds of thousands of owners who use GEOS on these machines and there is a correspondingly large demand for follow-on products. With careful programming, an application can be developed to run under all available system configurations with only minor changes. Berkeley Softworks plans to port GEOS to other 6502-based microcomputers, thereby further increasing the user base. As the popularity of GEOS grows, so does the market for your product.

Application Integration

GEOS offers a flexible cut and paste facility for text and graphic images. These photo scraps and text scraps allow applications to share data: a word processor can use graphics from a paint program and a graph and charting application can use data from a spreadsheet. The scrap format is standard and allows applications from different manufacturers to exchange data. Berkeley Softworks is currently developing a second-generation scrap facility for object-oriented graphics such as those used in desktop publishing and CAD programs.

Input and Output Technology

GEOS supports the concept of a device driver. A device driver is a small program which co-resides with the GEOS Kernal and communicates with I/O devices. Device drivers translates data and parameters from a generalized format that GEOS understands into a format relevant to the specific device. GEOS has input drivers for mice, joysticks, light pens, and other input devices, printer drivers for text and graphic output devices (including laser printers), and disk drivers for storage devices such as floppy disk drives, hard disks, and RAM expansion units (RAMdisks). As new devices become available, it is merely necessary to write a driver to support it.

What Exactly is GEOS?

First and foremost, GEOS is an operating system: a unified means for an application to interact with peripherals and system resources. GEOS is also an environment — specifically, a graphics based user-interface environment offering a standard library of routines and visual-based controls, such as menus and icons. And finally, GEOS is a programmer's toolbox, providing routines for double-precision integer math, random-number generation, and memory manipulation..

> **NOTE**: GEOS as a general term can represent full range of concepts — an operating system, a user environment, the deskTop, a group of integrated applications — but in this book it usually refers specifically to the GEOS Kernal, the resident portion of the operating system with which the application deals with.

GEOS As an Operating System

College textbook writers are forever coming up with splendid new metaphors to describe operating systems. But as the coach of a baseball team or the governor of California, an operating system has the same basic function: it is the manager of a computer, providing facilities for controlling the system while isolating the application from the underlying hardware. An operating system allows the application to function in higher-level abstract terms such as "load a file into memory11 rather than "let a bit rotate into the serial I/O shift register and send an acknowledge signal." The operating system will handle the laborious tasks of reading disk files, moving the mouse pointer, and printing to the printer.

GEOS provides the following basic operating system functions:

- Complete management of system initialization, multiple RAM banks, interrupt processing, keyboard/joystick/mouse input, as well as an application environment that supports dynamic overlays for programs larger than available memory, desk accessories, and the ability to launch other applications.

- A sophisticated disk file system that supports multiple drives, fast disk I/O, and RAM disks.

- Time-based processes, allowing a limited form of multitasking within an application.

- Printer output support, offering a unified way to deal with a wide variety of printers.

GEOS As a Graphic and User-Interface Environment

Interactive graphic interfaces have become the norm for modern day productivity. GEOS provides a services for placing lines, rectangles, and images on the screen, as well as handling menus, icons, and dialog boxes. Using the GEOS graphic elements make applications look better and easier to use.

GEOS provides the following graphic and user-interface functions:

- Multi-level dynamic menus which can be placed anywhere on the screen. GEOS automatically handles the user's interaction with the menus without permanently disrupting the display.
- Icons — graphic pictures the user can click on to perform some function.
- Complete dialog box library offering a standard set of dialog boxes (such as the file selector) ready for use. The application may also define its own custom dialog boxes.
- A library of graphic primitives for drawing points, lines, patterned rectangles, and pasting photo scraps from programs like geoPaint.
- Sprite support. (Sprites are small graphic images which overlay the display screen and can be moved easily. The mouse pointer, for example, is a sprite.)
- A secondary screen buffer for undo operations.

GEOS As a Programmer's Toolbox

.GEOS also contains a large library of general support routines for math operations, string manipulations, and other functions. This relieves the application programmer of the task of writing and debugging common routines ("re-inventing the wheel" as it were).

GEOS provides the following support routines:

- Double-precision (two-byte) math: shifting, signed and unsigned multiplication and division, random number generation, etc.
- Copy and compare string operations.
- Memory functions for initializing, filling, clearing, and moving.
- Miscellaneous routines for performing cyclic redundancy checks (CRC), initialization, error handling, and machine-specific functions.

Development System Recommendations

There are many ways to develop GEOS applications. Berkeley Softworks, for example, uses a UNIX™ based 6502 cross assembler and proprietary in-circuit emulators to design, test, and debug GEOS applications. Most developer's, however, will find this method too costly or impractical and will opt to develop directly on the target machines. Anticipating this, Berkeley Softworks has developed geoProgrammer, an assembler, linker, debugger package designed specifically for building GEOS applications.

geoProgrammer

geoProgrammer is a sophisticated set of assembly language development tools designed specifically for building GEOS applications. geoProgrammer is a scaled-down version of the UNIX™ based development environment Berkeley Softworks actually uses to develop GEOS programs. In fact, nearly all the functionality of our microPORT™ system has been preserved in the conversion to the GEOS environment. All sample source code, equates, and examples in this book are designed for uses with geoProgrammer.

The geoProgrammer development system consists of three major components:

geoAssembler, the workhorse of the system, takes 6502 assembly language source code and creates linkable object files.

- Reads source text from geoWrite documents; automatically converts graphic and icon images into binary data.
- Recognizes standard MOS Technology 6502 assembly language mnemonics and addressing modes.
- Allows over 1,000 symbol, label, and equate definitions, each up to 20 characters long.
- Full 16-bit expression evaluator allows any combination of arithmetic and logical operations.
- Supports local labels, as targets for branch instructions.
- Extensive macro facility with nested invocation and multiple arguments.
- Conditional assembly, memory segmentation, and space allocation directives.
- Generates relocatable object files with external definitions, encouraging modular programming.

geoLinker takes object files created with geoAssembler and links: them together, resolving all cross-references and generating a runable GEOS application file.

- Accepts a link command file created with geoWrite.

- Creates all GEOS applications types (sequential, desk accessory, and VLIR), allowing a customized header block and file icon. geoLinker will also create standard Commodore applications which do not require GEOS to run. Resolves external definitions and cross-references; supports complex expression evaluation at link-time.

- Allows over 1,700 unique, externally referenced symbols.

- Supports VLIR overlay modules.

geoDebugger allows you to interactively track-down and eliminate bugs and errors in your GEOS applications.

- Resides with your application and maintains two independent displays: a graphics screen for your application and a text screen for debugging.
- Automatically takes advantage of a RAM-expansion unit, allowing you to debug applications which use all of available program space.
- Complete set of memory examination and modification commands, including memory dump, fill, move, compare, and find.
- Symbolic assembly and disassembly.
- Supports up to eight conditional breakpoints.
- Single-step, subroutine step, loop, next, and execute commands.
- RESTORE key stops program execution and enters the debugger at any time.
- Contains a full-featured macro programming language to automate multiple keystrokes and customize the debugger command set.

Commodore 64

GEOS was first implemented on the Commodore 64, and currently there are more GEOS applications for this system than the Apple II or the Commodore 128. The following is recommended for developing under this environment:

- Commodore 64 or 64c computer.
- Commodore 1351 mouse.
- At least one 1541 or 1571 disk drive.
- Commodore 1764 or 1751 RAM-expansion unit.
- GEOS supported printer.
- The basic GEOS operating system (GEOS 64), version 1.3 or later which includes geoWrite and geoPaint.
- geoProgrammer for the Commodore 64.

Commodore 128

The Commodore 128 may be the ideal environment for prototyping and developing GEOS applications because it can be used to create programs which run under GEOS 64 (in 64 emulation mode) and GEOS 128. The 128 sports a larger memory capacity, and geoProgrammer takes advantage of this extra space for symbol and macro tables. The following is recommended for developing under this environment:

Commodore 128 computer.
- Commodore 1351 mouse.
- At least one 1541 or 1571 disk drive.
- Commodore 1764 or 1751 RAM-expansion unit.
- GEOS supported printer.
- The basic GEOS operating system (GEOS 64), version 1.3 or later which includes geoWrite and geoPaint.
- The basic GEOS 128 operating system, version 1.3 or later which includes geoWrite 128 and geoPaint 128.
- geoProgrammer for the Commodore 128.

Table of Contents

Name	Addr	Description	Category	Page
AllocateBlock	9048	Mark a disk block as in-use.	**disk mid-level**	19
AppendRecord	C289	Insert a new VLIR record after the current record.	**disk VLIR**	69
BBMult	C160	Byte by byte (single-precision) unsigned multiply.	**math**	125
Bell	N/A	1000 Hz Bell sound.	**utility**	177
BitmapClip	C2AA	Display a compacted bitmap, clipping to a sub-window.	**graphics**	88
BitmapUp	C142	Display a compacted bitmap without clipping.	**graphics**	90
BldGDirEntry	C1F3	Build a GEOS directory entry in memory.	**disk mid-level**	20
BlkAlloc	C1FC	Allocate sectors for a file.	**disk mid-level**	21
BlockProcess	C10C	Block process from running. Does not freeze timer.	**process**	163
Bmult	C163	Byte by word unsigned multiply.	**math**	126
BootGeos	C000	Reboot GEOS. Requires only 128 bytes at $c000.	**internal**	117
CalcBlksFree	C1DB	Calculate total number of free disk blocks.	**disk mid-level**	22
GetScanLine	C13C	Calculate scanline address.	**graphics**	94
CallRoutine	C1D8	pseudo-subroutine call. $0000 aborts call.	**utility**	178
ChangeDiskDevice	C2BC	Change disk drive device number.	**disk very low-level**	5
ChkDkGEOS	C1DE	Check if a disk is GEOS format.	**disk mid-level**	23
ClearRam	C178	Clear memory to $00.	**memory**	137
CloseRecordFile	C277	Close/Save currently open VLIR file.	**disk VLIR**	70
CmpFString	C26E	Compare two fixed-length strings.	**memory**	138
CmpString	C26B	Compare two null-terminated strings.	**memory**	139
CopyFString	C268	Copy a fixed-length string.	**memory**	140
CopyString	C265	Copy a null-terminated string.	**memory**	141
CRC	C20E	Cyclic Redundancy Check calculation.	**utility**	179
Dabs	C16F	Double-precision signed absolute value.	**memory**	127
DeleteFile	C238	Delete file.	**disk high-level**	54
DeleteRecord	C283	Delete current VLIR record.	**disk VLIR**	71
Ddec	C175	Double-precision unsigned decrement.	**math**	128
Ddiv	C169	Double-precision unsigned division.	**math**	129
DisablSprite	C1D5	Disable sprite.	**sprite**	172
DMult	C166	Double-precision unsigned multiply.	**math**	131
Dnegate	C172	Double-precision signed negation.	**math**	132
DoBOp	C2EC	(128) Back-RAM memory primitive	**memory**	142
DoDlgBox	C256	Display and begin interaction w/dialog box.	**dialog box**	2
DoIcons	C15A	Display and begin interaction with icons.	**icon/menu**	103
DoInlineReturn	C2A4	Return from inline subroutine.	**utility**	180
DoMenu	C151	Display and begin interaction with menus.	**icon/menu**	104
DoPreviousMenu	C190	Retract sub-menu and reactivate menus up one level.	**icon/menu**	106
DoRAMOp	C2D4	RAM-expansion unit access primitive.	**memory**	149

Name	Addr	Description	Category	Page
DoneWithIO	C25F	Restore system after serial I/O.	**disk very low-level**	6
DrawLine	C130	Draw, clear, or recover line between two endpoints.	**graphics**	91
DrawPoint	C133	Draw, clear, or recover a single screen point.	**graphics**	92
DrawSprite	C1C6	Define sprite image.	**sprite**	173
DSDiv	C16C	Double-precision signed division.	**math**	133
DShiftLeft	C15D	Double-precision left shift (zeros shifted in).	**math**	134
DShiftRight	C262	Double-precision right shift (zeros shifted in).	**math**	135
EnableProcess	C109	Make a process runnable immediately.	**process**	166
EnablSprite	C1D2	Enable sprite.	**sprite**	174
EnterDeskTop	C22C	Leave application and return to GEOS deskTop.	**disk high-level**	55
EnterTurbo	C214	Activate disk turbo on current drive.	**disk very low-level**	7
ExitTurbo	C232	Deactivate disk turbo on current drive.	**disk very low-level**	8
FastDelFile	C244	Quick file delete (requires full track/sector list).	**disk mid-level**	24
FetchRAM	C2CB	Transfer data from RAM-expansion unit.	**memory**	150
FillRam	C17B	Fill memory with a particular byte.	**memory**	143
FindBAMBit	C2AD	Get allocation status of particular disk block.	**disk mid-level**	25
FindFile	C20B	Search for a particular file.	**disk high-level**	56
FindFTypes	C23B	Find all files of a particular GEOS type.	**disk high-level**	57
FirstInit	C271	Initialize GEOS variables.	**internal**	118
FollowChain	C205	Follow chain of sectors, building track/sector table.	**disk mid-level**	26
FrameRectangle	C127	Draw an outline in a pattern	**graphics**	93
FreeBlock	C2B9	Mark a disk block as not-in-use in **BAM**.	**disk mid-level**	27
FreeFile	C226	Free all blocks associated with a file.	**disk mid-level**	28
FreezeProcess	C112	Pause a process countdown timer.	**process**	164
Get1stDirEntry	9030	Get first directory entry.	**disk mid-level**	29
GetBlock	C1E4	Read single disk block into memory.	**disk low-level**	16
GetCharWidth	C1C9	Calculate width of char without style attributes.	**text**	184
GetDirHead	C247	Read directory header into memory.	**disk mid-level**	31
GetFile	C208	Load GEOS file.	**disk high-level**	59
GetFHdrInfo	C229	Read a GEOS file header into **fileHeader**.	**disk mid-level**	32
GetFreeDirBlk	C1F6	Find an empty directory slot.	**disk mid-level**	33
GetNextChar	C2A7	Get next character from keyboard queue.	**text**	185
GetNxtDirEntry	9033	Get directory entry other than first.	**disk mid-level**	30
GetOffPageTrSc	9036	Get track and sector of off-page directory.	**disk mid-level**	35
GetPtrCurDkNm	C298	Return pointer to current disk name.	**disk mid-level**	61
GetRandom	C187	Calculate new random number.	**utility**	181
GetRealSize	C1B1	Calculate actual character size with attributes.	**text**	186
GetSerialNumber	C196	Return GEOS serial number.	**internal**	119
GetString	C1BA	Get string input from user.	**text**	187

Name	Addr	Description	Category	Page
GotoFirstMenu	C1BD	Retract all sub-menus and reactivate at main level.	**icon/menu**	107
i_BitmapUp	C1AB	Inline **BitmapUp**.	**graphics**	90
i_FillRam	C1B4	Inline **FillRam**.	**memory**	143
i_FrameRectangle	C1A2	Draw a solid outline with inline data	**graphics**	93
i_MoveData	C1B7	Inline **MoveData**.	**memory**	146
i_PutString	C1AE	Inline **PutString**.	**text**	195
i_Rectangle	C19F	Inline **Rectangle**.	**graphics**	97
InterruptMain	C100	Main interrupt level processing.	**internal**	120
InitForIO	C25C	Prepare system for serial I/O.	**disk very low-level**	9
InitMouse	FE80	Initialize input device.	**input driver**	112
InitProcesses	C103	Initialize processes.	**process**	165
InitRam	C181	Initialize memory areas from table.	**memory**	144
InitTextPrompt	C1C0	Initialize text prompt.	**text**	189
InsertRecord	C286	Insert new VLIR record in front of current record.	**disk VLIR**	72
IsMseInRegion	C2B3	Check if mouse is within a screen region.	**mouse/sprite**	155
LdApplic	C21D	Load GEOS application.	**disk mid-level**	36
LdDeskAcc	C217	Load GEOS desk accessory.	**disk mid-level**	38
LdFile	C211	Load GEOS data file.	**disk mid-level**	40
LoadCharSet	C1CC	Load and activate a new font	**text**	190
MainLoop	C1C3	GEOS MainLoop processing.	**internal**	121
MouseOff	C18D	Disable mouse pointer and GEOS mouse tracking.	**mouse/sprite**	156
MouseUp	C18A	Enable mouse pointer and GEOS mouse tracking.	**mouse/sprite**	157
MoveBData	C2E3	128 BackRAM memory move routine.	**memory**	145
MoveData	C17E	Intelligent memory block move.	**memory**	146
NewDisk	C1E1	Initialize a drive.	**disk mid-level**	41
NextRecord	C27A	Make next VLIR the current record.	**disk VLIR**	73
NormalizeX	C2E0	Normalize C128 X-coordinates for 40/80 modes.	**graphics**	95
NxtBlkAlloc	C24D	Version of **BlkAlloc** that starts at a specific block.	**disk mid-level**	42
OpenDisk	C2A1	Open disk in current drive.	**disk high-level**	61
OpenRecordFile	C274	Open VLIR file on current disk.	**disk VLIR**	74
Panic	C2C2	System-error dialog box.	**internal**	122
PointRecord	C280	Make specific VLIR record the current record.	**disk VLIR**	75
PosSprite	C1CF	Position sprite.	**sprite**	175
PreviousRecord	C27D	Make previous VLIR record the current record.	**disk VLIR**	76
PromptOff	C29E	Turn off text prompt.	**text/keyboard**	191
PromptOn	C29B	Turn on text prompt.	**text/keyboard**	192
PutBlock	C1E7	Write single disk block from memory.	**disk low-level**	17
PutChar	C145	Display a single character to screen.	**text**	197
PutDecimal	C184	Format and display an unsigned double-precision nbr.	**text**	194

Name	Addr	Description	Category	Page
PutDirHead	C24A	Write directory header to disk.	**disk mid-level**	44
PutString	C148	Print string of characters to screen.	**text**	195
ReadBlock	C21A	Get disk block primitive.	**disk mid-level**	11
ReadByte	C2B6	Read a File 1 byte at a time.	**disk mid-level**	45
ReadFile	C1FF	Read chained list of blocks into memory.	**disk mid-level**	46
ReadLink	904B	Read track/sector link.	**disk mid-level**	12
ReadRecord	C28C	Read current VLIR record into memory.	**disk VLIR**	77
RecoverAllMenus	C157	Recover all menus from background buffer.	**icon/menu**	108
RecoverMenu	C154	Recover single menu from background buffer.	**icon/menu**	109
Rectangle	C124	Draw a filled rectangle.	**graphics**	97
ReDoMenu	C193	Reactivate menus at the current level.	**icon/menu**	110
RenameFile	C259	Rename GEOS disk file.	**disk mid-level**	63
ResetHandle	C003	internal Bootstrap entry point	**internal**	123
RestartProcess	C106	Unblock, unfreeze, and restart process.	**process**	167
RstrAppl	C23E	Leave desk accessory and return to calling application.	**disk mid-level**	64
SaveFile	C1ED	Save Memory to create a GEOS file.	**disk high-level**	65
SetDevice	C2B0	Establish communication with a new serial device.	**disk high-level**	66
SetGDirEntry	C1F0	Create and save a new GEOS directory entry.	**disk mid-level**	48
SetGEOSDisk	C1EA	Convert normal CBM disk into GEOS format disk.	**disk high-level**	67
SetMouse	FE89	Reset input device scanning circuitry.	**input driver**	113
SetMsePic	C2DA	Set and preshift new soft-sprite mouse picture.	**mouse/sprite**	158
SetNewMode	C2DD	Change GEOS 128 graphics mode (40/80 switch).	**graphics**	98
SetPattern	C139	Set current fill pattern.	**graphics**	99
SetGDirEntry	C1F0	Create and save a new GEOS directory entry.	**disk mid-level**	49
Sleep	C199	Put current subroutine to sleep for a specified time.	**process**	168
SlowMouse	FE83	Reset mouse velocity variables.	**input driver**	114
SmallPutChar	C202	Fast character print routine.	**text**	196
StartASCII	7912	Begin ASCII mode printing.	**print driver**	160
StartAppl	C22F	Warmstart GEOS and start application in memory.	**disk mid-level**	51
StashRAM	C2C8	Transfer memory to RAM-expansion unit.	**memory**	151
SwapBData	C2E6	128 memory swap between front/back ram.	**memory**	147
SwapRAM	C2CE	RAM-expansion unit memory swap.	**memory**	152
RstrFrmDialog	C2BF	Exits from a dialog box.	**dialog box**	3
TempHideMouse	C2D7	Hide soft-sprites before direct screen access.	**mouse/sprite**	159
TestPoint	C13F	Test status of single screen point (on or off?).	**graphics**	100
ToBasic	C241	Pass Control to Commodore BASIC.	**utility**	182
UnblockProcess	C10F	Unblock a blocked process, allowing it to run again.	**process**	168
UnfreezeProcess	C115	Unpause a frozen process timer.	**process**	170
UpdateMouse	FE86	Update mouse variables from input device.	**input driver**	115

Name	Addr	Description	Category	Page
UpdateRecordFile	C295	Update currently open VLIR file without closing	**disk VLIR**	78
UseSystemFont	C14B	Use default system font (BSW 9).	**text**	196
VerifyBData	C2E9	128 BackRAM verify.	**memory**	148
VerifyRAM	C2D1	RAM-expansion unit verify.	**memory**	153
VerWriteBlock	C223	Disk block verify primitive.	**disk very low-level**	13
WriteBlock	C220	Write disk block primitive.	**disk very low-level**	14
WriteFile	C1F9	Write chained list of blocks to disk.	**disk mid-level**	52
WriteRecord	C28F	Write current VLIR record to disk.	**disk VLIR**	79

Table of Contents

GEOS Kernal by Category

Table of Contents

Table of Contents

Table of Contents

categories

Table of Contents

categories

Table of Contents

categories

Chapter 1 GEOS Kernal

dialog box

DoDlgBox:	(C64,C128)	C256

Function: Initializes, displays, and begins interaction with a dialog box.

Parameters: **r0** DIALOG — pointer to dialog box definition (word).
r5-r10 can be used to send parameters to a dialog box.

When using **DBGetFileS**
r5 BUFFER Ptr to buffer to store returned filename.
r7L FILETPE GEOS file type to search for (byte). (NULL for all)
r10 PERMNAME GEOS file type to search for (byte). (NULL for all)

Wheels: When using **DBGetFileS and bit 7 of r7L is set.**
r5 FILTER Ptr to Filter Procedure. Called once for every file before adding to the list of files.
r7L FILETPE GEOS file type to search for (byte). (NULL for all)
r10 PERMNAME GEOS file type to search for (byte). (NULL for all)

Returns: **r0L** return code: typically the number of the system icon clicked on to exit.
*Note: returns when dialog box exits through **RstrFrmDialog**.*

Destroys: **n/a**

Description: **DoDlgBox** saves off the current state of the system, places GEOS in a near warm start state, displays the dialog box according to the definition table (whose address is passed in **r0**), and begins tracking the user's interaction with the dialog box. When the dialog box finishes, the original system state is restored, and control is returned to the application.

Simple dialog boxes will typically contain a few lines of text and one or two system icons (such as **OK** and **CANCEL**). When the user clicks on one of these icons, the GEOS system icon routine exits the dialog box with an internal call to **RstrFrmDialog**, passing the number of the system icon selected in **sysDBData**. **RstrFrmDialog** restores the system state and copies **sysDBData** to **r0L**.

More complex dialog boxes will have application-defined icons and routines that get called. These routines, themselves, can choose to load a value into **sysDBData** and call **RstrFrmDialog**.

Note: Part of the system context save within **DoDlgBox** saves the current stack pointer. Dialog boxes cannot be nested. **DoDlgBox** is not reentrant. That is, a dialog box should never call **DoDlgBox**.

Structure: **DIALOG**

Example:

See also: **RstrFrmDialog**

RstrFrmDialog:	(C64,C128)	C2BF

Function: Exits from a dialog box, restoring the system to the state prior to the call to **DoDlgBox**.

Parameters: none.

Returns: Returns to point where **DoDlgBox** was called. System context is restored. **r0L** contains **sysDBData** return value.

Destroys: assume a, x, y, **r0H-r15**

Description: **RstrFrmDialog** allows a custom dialog box routine to exit from the a dialog box. **RstrFrmDialog** is typically called internally by the GEOS system icon dialog box routines. However, it may be called by any dialog box routine to force an immediate exit.

RstrFrmDialog first restores the GEOS system state (context restore) and then calls indirectly through **recoverVector** to remove the dialog box rectangle from the screen. The routine in **recoverVector** is called with the **r2-r4** loaded for a call to **RecoverRectangle**. By default **recoverVector** points to **RecoverRectangle**, which will automatically recover the foreground screen from the background buffer. However, if the application is using background buffer for data, it will need to intercept the recover by placing the address of its own recover routine in recoverVector. If there is no shadow on the dialog box, then **recoverVector** is only called through once with **r2-r4** holding the coordinates of the dialog box rectangle. However, if the dialog box has a shadow, then **recoverVector** will be called through two times: first for the patterned shadow rectangle and second for the dialog box rectangle. The application may want to special-case these two recovers when recovering.

Note: **RstrFrmDialog** restores the sp register to value it contained at the call to **DoDlgBox** just before returning. This allows **RstrFrmDialog** to be called with an arbitrary amount of data on top of the stack (as would be the case if called from within a subroutine). GEOS will restore the stack pointer properly.

Structure: **DIALOG**

Example

disk very low-level

ChangeDiskDevice: (C64, C128) C2BC

Function: Instruct a drive to change its serial device number.

Parameters: a NEWDEVNUM — new device number to give current drive (byte).
curDrive drive whose device number will change.

Uses: **curDrive** drive whose device number will change.

Returns: **x** error ($00 = no error).

Alters: **curDrive** NEWDEVNUM
curDevice NEWDEVNUM

Destroys: **a,y**

Description: **ChangeDiskDevice** requests the turbo software to change the serial device number of the current drive. Most applications have no need to call this routine, as it is in the realm of low-level disk utilities. **ChangeDiskDevice** is used primarily by the deskTop and Configure programs to add, rearrange, and remove drives.

Be aware that changing the device number merely instructs the turbo software in the drive to monitor a different serial bus address. Many internal GEOS variables and disk drivers expect the original device number to remain unchanged.

Note: If **ChangeDiskDevice** is used on a RAMdisk, **curDrive** and **curDevice** both change. However, because of the nature of the RAMdisk driver, the RAMdisk does not respond as this new device.

Example:

See also: **SetDevice**

DoneWithIO: (C64, C128) C25F

Function: Restore system after I/O across the serial bus.

Parameters: none.

Returns: nothing.

Destroys: a,y

Description: **DoneWithIO** restores the state of the system after a call to **InitForIO.** It restores the interrupt status, turns sprite DMA back on, returns the 128 to its original clock speed, and switches out the ROM and I/O banks if appropriate (only on C64).

Disk and printer routines access the serial bus between calls to **InitForIO** and **DoneWithIO.**

Example: **MyPutBlock**

See also: **InitForIO**

EnterTurbo: **(C64, C128)** C214

Function: Activate disk drive turbo mode

Parameters: none.

Uses: **curDrive** currently active disk drive.
curType vl.3+: checks disk type because not all use turbo software.

Returns: **x** error ($00 = no error).

Destroys: **a,y**

Description: **EnterTurbo** activates the turbo software in the current drive. If the turbo software has not yet been downloaded to the drive, **EnterTurbo** will download it. The turbo software allows GEOS to perform high-speed serial disk access.

EnterTurbo treats different drive types appropriately. A RAMdisk, for example, does not use turbo code so **EnterTurbo** will not attempt to download the turbo software.

The very-low level Commodore GEOS read/write routines, such as **ReadBlock**, **WriteBlock**, **VerWriteBlock**, and **ReadLink**, expect the turbo software to be active. Call **EnterTurbo** before calling one of these routines.

Example: **MyPutBlock**

See also: **WriteBlock**, **ExitTurbo**, **PurgeTurbo**.

ExitTurbo: (C64, C128) **C232**

Function: Deactivate disk drive turbo mode.
Parameters: none.

Uses: **curDrive** currently active disk drive.

Returns: **x** error ($00 = no error).

Destroys: **a,y**

Description: **ExitTurbo** deactivates the turbo software in the current drive so that the serial bus may access another device. **SetDevice** automatically calls this before changing devices.

Note: If the turbo software has not been downloaded or is already inactive, **ExitTurbo** will do nothing.

Example:

See also: **EnterTurbo, PurgeTurbo.**

InitForIO: (C64, C128) C25C

Function: Prepare for I/O across the serial bus

Parameters: none.

Returns: nothing.

Destroys: a,y

Description: **InitForIO** prepares the system to perform I/O across the Commodore serial bus. It disables interrupts, turns sprite DMA off, slows the 128 down to lMhz, switches in the ROM and I/O banks if necessary, and performs anything other initialization needed for fast serial transfer.

Call **InitForIO** before directly accessing the serial port (e.g., in a printer driver) or before using **ReadBlock**, **WriteBlock**, VerWriteBIock, or **ReadLink.** To restore the system to its previous state, call **DoneWithIO.**

Example: **MyPutBlock**

See also: **DoneWithIO, SetDevice**

PurgeTurbo: (C64, C128) C235

Function: Completely deactivate and remove disk drive turbo code from current drive, returning to standard Commodore DOS mode.

Parameters: **none**

Uses: **curDrive** currently active disk drive.

Returns: **x** error ($00 = no error).

Destroys: **a,y**

Description: **PurgeTurbo** deactivates and removes the turbo software from the current drive, returning control of the device to the disk drive's internal ROM software. This allows access to normal Commodore DOS routines. An application may want to access the Commodore DOS to perform disk functions not offered by the GEOS Kernal such as formatting.

Example:

See also: **EnterTurbo, ExitTurbo.**

ReadBlock: (C64, C128) **C12A**

Function: Very low-level read block from disk.

Parameters:
r1L **TRACK—valid track number (byte),**
r1H **SECTOR—valid sector on track (byte).**
r4 **BUFFER — address of buffer of BLOCKSIZE bytes to read block into (word).**

Uses:
curDrive currently active disk drive.
curType GEOS 64 v1.3 and later for detecting REU shadowing.

Returns: **x** error ($00 = no error).

Destroys: **a,y**

Description: **ReadBlock** reads the block at the specified TRACK and SECTOR into BUFFER. If the disk is shadowed, **ReadBlock** will read from the shadow memory. **ReadBlock** is a pared down version of GetBlock. It expects the application to have already called **EnterTurbo** and **InitForIO.** By removing this overhead from GetBlock, multiple sector reads can be accomplished without the redundant initialization. This is exactly what happens in many of the higher-level disk routines that read multiple blocks at once, such as **ReadFile.**

ReadBlock is useful for multiple-sector disk operations where speed is an issue and the standard GEOS routines don't offer a decent solution. **ReadBlock** can function as the foundation of specialized, high-speed disk routines.

Example: **MyGetBlock**

See also: **GetBlock, WriteBlock, VerWriteBlock.**

ReadLink: (C64, C128) 904B

Function: Read link (first two bytes) from a disk block

Parameters: **r1L** **TRACK** — track number (byte),
r1H **SECTOR** — sector on track (byte).
r4 **BUFFER** — address of buffer of at least **BLOCKSIZE** bytes, usually points to **diskBlkBuf** (word).

Uses: **curDrive** currently active disk drive.

Returns: **x** error ($00 = no error).

Destroys: **a,y**

Description: **ReadLink** returns the track/sector link from a disk block as the first two bytes in BUFFER. The remainder of BUFFER (BLOCKSIZE-2 bytes) may or may not be altered.

ReadLink is useful for following a multiple-sector chain in order to build a track/sector table. It mainly of use on 1581 disk drives, which walk through a chain significantly faster when only the links are read. Routines such as **DeleteFile** and **FollowChain** will automatically take advantage of this capability of 1581 drives.

Note: Disk drives that do not offer any speed increase through **ReadLink** will simply perform a **ReadBlock**.

Note: Does not work in 1541 Drivers. Use **ReadBlock** instead.

Example:

See also: **ReadBlock, FollowChain**

VerWriteBlock: (C64, C128) C223

Function: Very low-level verify block on disk.

Parameters: **r1L** TRACK — track number (byte).
r1H SECTOR — valid sector on track (byte).
r4 BUFFER — address of buffer of BLOCKSIZE bytes that contains data that should be on this sector (word).

Uses: **curDrive** currently active disk drive.
curType GEOS 64 v1.3 and later for detecting REU shadowing.

Returns: **x** error ($00 = no error).

Destroys: **a,y**

Description: **VerWriteBlock** verifies the validity of a recently written block. If the block does not verify, the block is rewritten by calling **WriteBlock**. **VerWriteBlock** is a low-level disk routine and expects the application to have already called **EnterTurbo** and **InitForIO**.

VerWriteBlock can be used to accelerate the verifies that accompany multiple sector writes by first writing all the sectors and then verifying them. This is often faster than verifying a sector immediately after writing it because when writing sequential sectors, the GEOS turbo code will catch the sector interleave. If a sector is written and then immediately verified, the turbo code will need to wait for the disk to make one complete revolution before the newly-written sector will again pass under the read/write head. By writing all the sectors first and catching the interleave, then verifying all the sectors (again, catching the interleave), the dead time when the turbo code is Waiting for the disk to spin around is minimized. Many of the higher-level disk routines that write multiple blocks do just this.

VerWriteBlock is useful for multiple-sector disk operations where speed is an issue and the standard GEOS routines don't offer a decent solution. **VerWriteBlock** can function as the foundation of specialized, high-speed disk routines.

VerWriteBlock does not always do a byte-by-byte compare with the data in BUFFER. Some devices, such as the Commodore 1541, can do a cyclic redundancy check on the data in the block, and this internal checksum is sufficient evidence of a good write. Other devices, such as RAM-expansion units, have built-in byte-by-byte verifies.

Example: **MyPutBlock**

See also: **WriteBlock, PutBlock**

WriteBlock: (C64, C128) C220

Function: Very low-level write block to disk.

Parameters: **r1L** TRACK — valid track number (byte).
r1H SECTOR—valid sector on track (byte).
r4 BUFFER — address of buffer of BLOCKSIZE bytes that contains data to write out (word).

Uses: **curDrive** currently active disk drive.
curType GEOS 64 v1.3 and later for detecting REU shadowing.

Returns: **x** error ($00 = no error).

Destroys: **a,y**

Description: **WriteBlock** writes the block at BUFFER to the specified TRACK and SECTOR. If the disk is shadowed, **WriteBlock** will also write the data to the shadow memory. **WriteBlock** is pared down version of **PutBlock.** It expects the application to have already called **EnterTurbo** and **InitForIO**, and it does not verify the data after writing it.

WriteBlock can be used to accelerate multiple-sector writes and their accompanying verifies by writing all the sectors first and then verifying them. This is often faster than verifying a sector immediately after writing it because when writing sequential sectors, the GEOS turbo code will catch the sector interleave. If a sector is written and then immediately verified, the turbo code will need to wait for the disk to make one complete revolution before the newly written sector will again pass under the read/write head. By writing all the sectors first and catching the interleave, then verifying all the sectors (again, catching the interleave), the dead time when the turbo code is waiting for the disk to spin around is minimized. Many of the higher-level disk routines that write multiple blocks do just this.

WriteBlock is useful for multiple-sector disk operations where speed is an issue and the standard GEOS routines don't offer a decent solution. **WriteBlock** can function as the foundation of specialized, high-speed disk routines.

Example: **MyPutBlock**

See also: **PutBlock, ReadBlock, VerWriteBlock.**

disk low-level

disk low-level

GetBlock: (C64, C128) C1E4

Function: General purpose routine to get a block from current disk.

Parameters:
r4 BUFFER — address of buffer to place block; must be at least BLOCKSIZE bytes (word).
r1L TRACK — track number (byte).
r1H SECTOR — sector number on track (byte).

Uses:
curDrive currently active disk drive.
curType GEOS 64 v1.3 and later for detecting REU shadowing.

Returns:
x error ($00 = no error).
r1, **r4** unchanged

Destroys:
a,y
(1581 drive, r1, r4)
AAnote: *Need to confirm is this is still true*

Description: **GetBlock** reads a block from the disk into BUFFER. **GetBlock** is useful for implementing disk utility programs and new file structures.

GetBlock is a higher-level version of **ReadBlock**. It calls **InitForIO**, **EnterTurbo**, **ReadBlock**, and **DoneWithIO**. If an application needs to read many blocks at once, **ReadBlock** may offer a faster solution. If the disk is shadowed, **GetBlock** will read from the shadow memory, resulting in a faster transfer.

The Commodore 1581 driver has a bug that causes its **GetBlock** to trash r1 and **r4**.

Example:

See also: **PutBlock, WriteBlock, BlkAlloc.**

PutBlock: (C64, C128) C1E7

Function: General purpose routine to write a block to disk with verify.

Parameters: **r4** BUFFER — address of buffer to get block from;
rlL TRACK — valid track number (byte).
r1H SECTOR — valid sector on track (byte).

Uses: **curDrive** currently active disk drive.
curType GEOS 64 vl.3 and later for detecting REU shadowing.

Returns: **x** error ($00 = no error).
rl, **r4** unchanged

Destroys: **a,y**

Description: **PutBlock** writes a block from BUFFER to the disk. **PutBlock** is useful for implementing disk utility programs and new file structures.

PutBlock is a higher-level version of **WriteBlock**. It calls **InitForIO**, **EnterTurbo**, **ReadBlock**, and **DoneWithIO**. If an application needs to write many blocks at once, **WriteBlock** may offer a faster solution. If the disk is shadowed, **PutBlock** will also write the data to the shadow memory..

Note[3]: **PutBlock** does no boundary check on the buffer. If the buffer is less the then **BLOCKSIZE ($100)** bytes, **PutBlock** will write the buffer and the memory contents that are after the buffer. This normally will not cause any problems as the size of data in the data block is stored in offset 1 of the block when the block is not full.

Example:

See also: **GetBlock, WriteBlock, BlkAlloc.**

disk mid-level

AllocateBlock: (C64, C128) 9048

Function: Allocate a disk block, marking it as in use..

Parameters: **r6L** track number of block (byte).
r6H sector number of block (byte).

Uses: **curDrive** drive that disk is in.
curDirHead this buffer must contain the current directory header.
dir2Head2† (BAM for 1571 and 1581 drives only)
dir3Head3† (BAM for 1581 drive only)
;†used internally by GEOS disk routines; applications generally don't use.

Returns: x error ($00 = no error).
BAD_BAM
r6 unchanged

Alters: **curDirHead** BAM updated to reflect newly allocated blocks.
dir2Head† (BAM for 1571 and 1581 drives only)
dir3Head† (BAM for 1581 drive only)

Destroys: a,y, **r7, r8H**

Description: **AllocateBlock** allocates a single block on this disk by setting the appropriate flag in the block allocation map (BAM).

If the sector is already allocated then a BAD_BAM error is returned. **AllocateBlock** does not automatically write out the BAM. See **PutDirHead** for more information on writing out the BAM. The Commodore 1541 device drivers do not have a jump table entry for **AllocateBlock**. All other device drivers, however, do. The following subroutine will properly allocate a block on any device, including the 1541.
NewAllocateBlock

Example: **CallNewAlloc**

See also: **SetNextFree, BlkAlloc, FreeBlock.**

disk mid-level

BldGDirEntry: (C64, C128) C1F3

Function: Builds a directory entry in memory for a GEOS file using the information in a file header.

Parameters:
r2 NUMBLOCKS — number of blocks in file (word).
r6 TSTABLE — pointer to a track/sector list of unused blocks (unused but allocated in the BAM), usually a pointer to **fileTrScTab**; **BlkAlloc** can be used to build such a list (word).
r9 FILEHDR — pointer to GEOS file header (word).

Uses: **curDrive** drive that disk is in.

Returns: **r6** pointer to first non-reserved block in track/sector table (**BldGDirEntry** reserves one block for the file header and a second block for the index table if the file is a VLIR file).

Alters: **dirEntryBuf** contains newly-built directory entry.

Destroys: a,y, **r5**

Description: Given a GEOS file header, **BldGDirEntry** will build a system specific directory entry suitable for writing to an empty directory slot.

Most applications create new files by calling **SaveFile**. **SaveFile** calls **SetGDirEntry**, which calls **BldGDirEntry** as part of its normal processing.

Example: **MySetGDirEntry**

See also: **SetGDirEntry**

disk mid-level

BlkAlloc: (C64, C128) C1FC

Function: Allocate enough disk blocks to hold a specified number of bytes.

Parameters: **r2** BYTES — number of bytes to allocate space for. Commodore version can allocate up to 32,258 bytes (127 Commodore blocks).
r6 TSTABLE — pointer to buffer for building out track and sector table of allocated blocks, usually points to **fileTrScTab** (word).

Uses:
curDrive drive that disk is in.
curDirHead this buffer must contain the current directory header.
dir2Head2† (BAM for 1571 and 1581 drives only)
dir3Head3† (BAM for 1581 drive only)
interleave† desired physical sector interleave (usually 8); used by **SetNextFree.** Applications need not set this explicitly — will be set automatically by internal GEOS routines.

;†used internally by GEOS disk routines; applications generally don't use.

Returns: x error ($00 = no error).
r2 number of blocks allocated to hold BYTES amount of data.
r3L track of last allocated block.
r3H sector of last allocated block.

Alters:
curDirHead BAM updated to reflect newly allocated blocks.
dir2Head† (BAM for 1571 and 1581 drives only)
dir3Head† (BAM for 1581 drive only)

Destroys: a, y, **r4-r8**

Description: **BlkAlloc** calculates the number of blocks needed to store BYTES amount of data, taking any standard overhead into account (such as the two-byte track/sector link required in each Commodore block), then calls **CalcBlksFree** to ensure that enough free blocks exist on the disk. If there are not enough free blocks to accommodate the data, **BlkAlloc** returns an **INSUFFICIENT_SPACE** error without allocating any blocks. Otherwise, **BlkAlloc** calls **SetNextFree** to allocate the proper number of unused blocks.

BlkAlloc builds out a track and sector table in the buffer pointed to by TSTABLE. The 256 bytes at **fileTrScTab** are usually used for this purpose. When **BlkAlloc** returns, the table contains a two-byte entry for each block that was allocated: the first byte is the track and the second byte is the sector. The last entry in the table has its first byte set to $00, indicating the end of the table. The second byte of the last entry is an index to the last byte in the last block. This track/sector list can be passed directly to **WriteFile** for use in writing data to the blocks.

Note: For more information on the scheme used to allocate successive blocks, refer to **SetNextFree.**

Example: **GrabSomeBlocks**

See also: **NxtBlkAlloc, SetNextFree, GetFreeDirBlk, FreeBlock.**

CalcBlksFree: (C64, C128) C1DB

Function: Calculate total number of free blocks on disk.

Parameters: **r5** DIRHEAD — address of directory header, should always point to **curDirHead** (word).

Uses:
curDrive drive that disk is in.
dir2Head2† (BAM for 1571 and 1581 drives only)
dir3Head3† (BAM for 1581 drive only)

†used internally by GEOS disk routines; applications generally don't use.

Returns:
r4 number of free blocks.
r5 unchanged.
r3 in GEOS vl.3 and later: total number of available blocks on empty disk. This is useful because vl.3 and later support disk devices other than the 1541. GEOS versions earlier than vl.3 leave r3 unchanged.

Destroys: a, y

Description: **CalcBlksFree** calculates the number of free blocks available on the disk. An application can call **CalcBlksFree**, for example, to tell the user the amount of free space available on a particular disk. GEOS disk routines that allocate multiple blocks at once, such as **BlkAlloc**, call **CalcBlksFree** to ensure enough free space exists on the disk to prevent a surprise **ENSUFFICENT_SPACE** error, midway through the allocation. (This is why it is usually not necessary to check for sufficient space before saving a file or a VLIR record—the higher level GEOS disk routines handle this checking automatically.)

CalcBlksFree looks at the BAM in memory and counts the number of unallocated blocks. The BAM is stored in the directory header and the directory header is stored in the buffer at **curDirHead.** Calling **CalcBlksFree** requires first loading **r5** with the address of **curDirHead.**

```
LoadW r5, #curDirHead
jsr CalcBlksFree
```

When checking the total number of blocks (both allocated and free) on a particular disk device, call **CalcBlksFree** with r3 loaded with the number of blocks on a 1541 disk device. On GEOS v1.3 and above, this number is changed to reflect the actual number of blocks in the device. On previous versions of GEOS, r3 comes back unchanged.

```
N1541_BLOCKS = 664       ; total number of blocks on 1541 devices

LoadW  r3, #N1541_BL0CKS ; assume 1541 block count for vl.2 Kernal's
LoadW  r5, #curDirHead   ; point to the directory header
jsr    CalcBlksFree      ; r3 comes back with total number of blocks
                         ; on this device
```

Example: **CheckDiskSpace**

See also: **NxtBlkAlloc, SetNextFree, GetFreeDirBlk, FreeBlock.**

ChkDkGEOS: (C64, C128) C1DE

Function: Check Commodore disk for GEOS format.

Parameters: **r5** DIRHEAD — address of directory header, should always point to **curDirHead** (word).

Returns: a TRUE/FALSE matching **isGEOS.**
Z flag=0 GEOS Disk
Z flag=1 Non GEOS Disk

Alters: **isGEOS** set to **TRUE** if disk is a GEOS disk, otherwise set to **FALSE.**

Destroys: a,y

Description: **ChkDkGEOS** checks the directory header for the version string that flags it as a GEOS disk (at **OFF_GEOS_BD**). The primary difference between a GEOS disk and a standard Commodore disk is the addition of the off-page directory and the possibility of GEOS files on the disk. GEOS files have an additional file header block that holds the icon image and other information, such as the author name and permanent name string. To convert a non-GEOS disk into a GEOS disk, use **SetGEOSDisk.**

OpenDisk automatically calls **ChkDkGEOS.** As long as **OpenDisk** is used before reading a new disk, applications should have no need to call **ChkDkGEOS**

Example:

```
    jsr   GetDirHead          ; read in the directory header
    txa                       ; check status
    bne   99$                 ; exit on error
    LoadW r5,#curDirHead      ; point to directory header
    jsr   ChkDkGEOS           ; Check for GEOS disk
    beq 50$                   ; if not a GEOS disk, branch
    ; code here to handle GEOS disk
    bra 90$                   ; jump to exit
50$
    ; code here to handle non-GEOS disk
90$
    clc                       ; Success Exit
    rts
99$
    sec
    rts                       ; error exit
```

See also: **SetGEOSDisk**

disk mid-level

FastDelFile: (C64, C128) **C244**

Function: Special Commodore version of **DeleteFile** that quickly deletes a sequential file when the track/sector table is available.

Parameters: **r0** FILENAME — pointer to null-terminated file name (word).
r3 TSTABLE — pointer to track and sector table of file, usually points to **fileTrScTab** (word).

Uses:

curDrive	
curType	GEOS 64 v 1.3 and later for detecting REU shadowing.
curDirHead	BAM updated to reflect newly freed blocks.
dir2Head2†	(BAM for 1571 and 1581 drives only)
dir3Head3†	(BAM for 1581 drive only)

†used internally by GEOS disk routines; applications generally don't use.

Returns: x error ($00 = no error).

Destroys: a,y, **r0, r9**

Description: **FastDelFile** quickly deletes a sequential file by taking advantage of an already existing track/sector table. It first removes the directory entry determined by FILENAME and calls FreeBlock for each block in a track/sector table at TSTABLE. The track/sector table is in the standard format, such as that returned from **ReadFile**, where every two-byte entry constitutes a track and sector. A track number of $00 terminates the table.

FastDelFile is fast because it does not need to follow the chain of sectors to delete the individual blocks. It can do most of the deletion by manipulating the BAM in memory then writing it out with a call to **PutDirHead** when done.

FastDelFile will not properly delete VLIR files without considerable work on the application's part. Because there is no easy way to build a track/sector table that contains all the blocks in all the records of a VLIR file, it is best to use **DeleteFile** or **FreeFile** for deleting VLIR files or **DeleteRecord** for deleting a single record.

FastDelFile calls **GetDirHead** before freeing any blocks. This will overwrite any BAM and directory header in memory.

Note: **FastDelFile** can be used to remove a directory entry without actually freeing any blocks in the file by passing a dummy track/sector table, where the first byte (track number) is $00 signifying the end of the table: See Example **DeleteDirEntry**

Examples: **DeleteDirEntry, ReadAndDelete**

See also: **FreeFile, DeleteFile**

disk mid-level

FindBAMBit: (C64, C128) **C2AD**

Function: Get disk block allocation status.

Parameters: **r6L** TRACK —track number of block (byte).
r6H SECTOR — sector number of block (byte).

Uses:
curDrive
curDirHead BAM updated to reflect newly freed blocks.
dir2Head (BAM for 1571 and 1581 drives only)
dir3Head (BAM for 1581 drive only)

Returns: st z flag reflects allocation status (1 = free; 0 = allocated).
r6 unchanged

1541 drives only:
x offset from **curDirHead** for BAM byte.
r8H mask for isolating BAM bit.
a BAM byte masked with r8H.
r7H offset from **curDirHead** of byte that holds free blocks on track total.

Destroys: non-1541 drives:
a, y, **r7H**, **r8H**.

1541 drives:
y (a, **r7H**, and **r8H** all contain useful values).

Description: **FindBAMBit** accesses the BAM of the current disk "in **curDirHead**) and returns the allocation status of a particular block. If the BAM bit is zero, then the block is in-use; if the BAM bit is one, then the block is free. **FindBAMBit** returns with the z flag set to reflect the status of the BAM so that a subsequent bne or beq branch instructions can test the status of a block after calling **FindBAMBit**.

```
bne  BlockIsFree       ;branch if block is free
  - or -
beq  BlockInUse        ;branch if block is in-use
```

Note: **FindBAMBit** will return the allocation status of a block on any disk device, even those with large or multiple BAMs (such as the 1571 and 1581 disk drives). Only the 1541 driver, however, will return useful information in a, y, **r7H**, and **r8H**. For an example of using these extra 1541 return values, refer to **AllocateBlock**.

Examples:

```
LoadB   r6L,#TRACK    ; get track and sector number
LoadB   r6H,#SECTOR
jsr     FindBAMBit    ; get allocation status
beq     BlocklnUse    ; branch if already in use
```

See also: **AllocateBlock, FreeBlock, GetDirHead , PutDirHead**

FollowChain: (C64, C128) C205

Function: Follow a chain of Commodore disk blocks, building out a track/sector table.

Parameters:
r1L START_TRACK — track number of starting block (byte).
r1H START_SEC — sector number of starting block (byte).
r3 TSTABLE — pointer to buffer for building out track and sector table of chain, usually points to **fileTrScTab** (word).

Uses:
curDrive
curType GEOS 64 v 1.3 and later for detecting REU shadowing.

Returns: x error ($00 = no error).
r3 unchanged
track/sector built-out in buffer pointed to by TSTABLE.

Alters: **diskBlkBuf** **used for temporary block storage.**

Destroys: a, y, **r1**, **r4**

Description: **FollowChain** constructs a track/sector table for a list of chained blocks on the disk. It starts with the block passed in START_TR and START_SC and follows the links until it encounters the last block in the chain. Each block (including the first block at START_TR, START_SC) becomes a part of the track/sector table.

Commodore disk blocks are linked together with track/sector pointers. The first two bytes of each block represent a track/sector pointer to the next block in the chain. Each sequential file and VLIR record on the disk is actually a chained list of blocks. **FollowChain** follows these track/sector links, adding each to the list at TSTABLE until it encounters a track pointer of $00, which terminates the chain. **FollowChain** adds this last track pointer ($00) and its corresponding sector pointer (which is actually an index to the last valid byte in the block) to the track/sector table and returns to the caller.

FollowChain builds a standard track/sector table compatible with routines such as **WriteFile** and **FastDelFile.**

Examples:

```
LoadB   r1L,#START_TR       ; start track
LoadB   r6H,#START_SC       ; and sector
LoadW   r3,#fileTricTab     ; buffer for table
jsr     FollowChain         ; get allocation status
txa                         ; set status flags
bne                         ; branch if error
```

See also: **FastDelFile, WriteFile, ReadLink**

disk mid-level

FreeBlock: (C64, C128) C2B9

Function: Free an allocated disk block.

Parameters: **r6** track number of block to free (byte).
r6H sector number of block to free (byte).

Uses: **curDrive**
curDirHead must contain the current directory header.
dir2Head (BAM for 1571 and 1581 drives only)
dir3Head (BAM for 1581 drive only)

Returns: **x** error ($00 = no error).
BAD_BAM if block already free.
r6L, r6H unchanged.

Alters: **curDirHead** BAM updated to reflect newly allocated block.
dir2Head (BAM for 1571 and 1581 drives only)
dir3Head (BAM for 1581 drive only)

Destroys: **a,y,r7,r8H**

Description: **FreeBlock** tries to free (deallocate) the block number passed in r6. If the block is already free, then **FreeBlock** returns a **BAD_BAM** error.

Note: **FreeBlock** was not added to the Commodore GEOS jump table until vl.3, but it can be accessed directly under GEOS vl.2. The following routine will check the GEOS version number and act correctly under GEOS vl.2 and later. See Example **MyFreeBlock**

Example: **MyFreeBlock**

See also: **FreeFile, AllocateBlock**

FreeFile: (C64, C128) **C226**

Function: Free all the blocks in a GEOS file (sequential or VLIR) without deleting the directory entry. The GEOS file header and any index blocks are also deleted.

Parameters: **r9** DIRENTRY — pointer to directory entry of file being freed, usually points to **dirEntryBuf** (Apple GEOS: must be in main memory.) (word).

Uses: **curDrive**
curType GEOS 64 v 1.3 and later for detecting REU shadowing.

Returns: **x** error ($00 = no error).
r6L, **r6H** unchanged.

Alters:

diskBlkBuf	used for temporary block storage.
curDirHead	BAM updated to reflect newly allocated block.
dir2Head†	(BAM for 1571 and 1581 drives only)
dir3Head†	(BAM for 1581 drive only)
fileHeader	temporary storage of the index table when deleting a VLIR file.

†used internally by GEOS disk routines; applications generally don't use.

Destroys: **a,y,r0-r9**

Description: Given a valid directory entry, **FreeFile** will delete (free) all blocks associated with the file. The GEOS file header and any index blocks associated with the file are also be freed. The directory entry on the disk, however, is left intact

The directory entry is a standard GEOS data structure returned by routines such as **FindFile**, **Get1stDirEntry** and **GetNxtDirEntry**. **FreeFile** is called automatically by **DeleteFile**.

FreeFile tries to free (deallocate) the block number passed in **r6**. If the block is already free, then **FreeBlock** returns a **BAD_BAM** error.

FreeFile calls **GetDirHead** to get the current directory header and BAM into memory. It then checks at **OFF_GHDR_PTR** in the directory entry for a GEOS file header block, which it then frees.

If the file is a sequential file, **FreeFile** walks the chain pointed at by the **OFF_DE_TR_SC** track/sector pointer in the directory header and frees all the blocks in the chain. **FreeFile** then calls **PutDirHead** to write out the new BAM.

When using **Get1stDirEntry** and **GetNxtDirEntry**, do not pass **FreeFile** a pointer into **diskBlkBuf**. Copy the full directory entry (**DIRENTRY_SIZE** bytes) from **diskBlkBuf** to another buffer (such as **dirEntryBuf**) and pass **FreeFile** the pointer to that buffer. Otherwise when **FreeFile** uses **diskBlkBuf** it will corrupt the directory entry.

Because **FreeFile** deletes a block at a time as it follows the chains, it is capable of deleting files with chains larger than 127 blocks, which is the standard GEOS limit imposed by the size of TrScTable.

See also: **DeleteFile**, **FreeDir**, **FreeBlock**.

Get1stDirEntry: (C64, C128) 9030

Function: Loads in the first directory block of the current directory and returns a pointer to the first directory entry within this block.

Parameters: none.

Uses: **curDrive**
curType GEOS 64 v 1.3 and later for detecting REU shadowing.

Returns: **x** error ($00 = no error).
r5 pointer to first directory entry within **diskBlkBuf**.

Alters: **diskBlkBuf** directory block.

Destroys: **a,y,r1,r4**

Description: **Get1stDirEntry** reads in the first directory block of the current directory and returns with r5 pointing to the first directory entry. **Get1stDirEntry** is called by routines like **FindFTypes** and **FindFile.**

To get a pointer to subsequent directory entries, call **GetNxtDirEntry.**

Since Commodore GEOS does not support a hierarchical file system, the "current directory" is actually the entire disk.

Get1stDirEntry did not appear in the jump table until version 1.3. An application running under version 1.2 can access **Get1stDirEntry** by calling directly into the Kernal. The following subroutine will work on Commodore GEOS vl.2 and later:

```
;****************************************************
; MyGet1stDirEntry — Call instead of Get1stDirEntry
; to work on GEOS vl.2 and later
;****************************************************
;EQUATE: vl.2 entry point directly into Kernal. Must
;do a version check before calling.

o_Get1stDirEntry = $c9f7          ; exact entry point

MyGet1stDirEntry:
   lda version                    ; check version number
   cmp #$13
   bcc 10$                        ; branch < vl.3
   jmp Get1stDirEntry             ; direct call
10$
   jmp o_Get1stDirEntry           ; go through jump table
```

Example:

See also: **GetNxtDirEntry, FindFTypes.**

GetNxtDirEntry: (C64, C128) 9033

Function: Given a pointer to a directory entry returned by **Get1stDirEntry** or GetNxtDirEntry, returns a pointer to the next directory entry.

Parameters: **r5** CURDIRENTRY — pointer to current directory entry as returned from **Get1stDirEntry** or **GetNxtDirEntry**; will always be a pointer into **diskBlkBuf** (word).

Uses: **curDrive**

diskBlkBuf must be unaltered from previous call to **Get1stDirEntry** or **GetNxtDirEntry**.

curType GEOS 64 v 1.3 and later for detecting REU shadowing.

Returns: x error ($00 = no error).

r5 pointer to next directory entry within **diskBlkBuf**.

y non-zero if end of directory reached

Alters: **diskBlkBuf** directory block.

Destroys: **a,y,r1,r4**

Description: **GetNxtDirEntry** increments r5 to point to the next directory entry in **diskBlkBuf**. If **diskBlkBuf** is exceeded, the next directory block is read in and **r5** is returned with an index into this new block. Before calling **GetNxtDirEntry** for the first time, call **Get1stDirEntry**.

GetNxtDirEntry did not appear in the jump table until version 1.3. An application running under version 1.2 can access **GetNxtDirEntry** by calling directly into the Kernal. The following subroutine will work on Commodore GEOS vl.2 and later:

```
;****************************************************
; MyGetNxtDirEntry — Call instead of GetNxtDirEntry
; to work on GEOS vl.2 and later
;****************************************************
;EQUATE: vl.2 entry point directly into Kernal. Must
;do a version check before calling.

o_GetNxtDirEntry = $ca10           ; exact entry point

MyGetNxtDirEntry:
   lda version                     ; check version number
   cmp #$13
   bcc 10$                         ; branch < vl.3
   jmp GetNxtDirEntry              ; direct call
10$
   jmp o_GetNxtDirEntry            ; go through jump table
```

Example:

See also: **GetlstDirEntry**, **FindFTypes**.

disk mid-level

GetDirHead : (C64, C128) C247

Function: Read directory header from disk. GEOS also reads in the BAM

Parameters: none.

Uses: **curDrive**
curType GEOS 64 v 1.3 and later for detecting REU shadowing.

Returns: x error ($00 = no error).
r4 pointer to **curDirHead.**

Alters: **curDirHead** contains directory header
dir2Head† (BAM for 1571 and 1581 drives only)
dir3Head† (BAM for 1581 drive only)

†used internally by GEOS disk routines; applications generally don't use.

Destroys: **a,y,r1**

Description: **GetDirHead** reads the full directory header (256 bytes) into the buffer at **curDirHead.** This block also includes the BAM (block allocation map) for the entire disk.

GEOS disks, like the standard Commodore disks upon which they are based, have one directory header. The directory header occupies one full block on the disk. The Commodore directory header contains information about the disk, such as the location of the directory blocks, the disk name, and the GEOS version string (if a GEOS disk). The Commodore directory header also contains the disk BAM, which flags particular sectors as used or unused

GetDirHead calls **GetBlock** to read in the directory header block into the buffer at **curDirHead.** The directory header block contains the directory header and the disk BAM (block allocation map). Typically, applications don't call **GetDirHead** because the most up-to-date directory header is almost always in memory (at **curDirHead**), **OpenDisk** calls **GetDirHead** to get it there initially. Other GEOS routines update it in memory, some calling **PutDirHead** to bring the disk version up to date.

Because Commodore disks store the BAM information in the directory header it is important that the BAM in memory not get overwritten by an outdated BAM on the disk. An application that manipulates the BAM in memory (or calls GEOS routines that do so), must be careful to write the BAM back out (with **PutDirHead**) before calling any other routine that might overwrite the copy in memory. **GetDirHead** is called by routines such as **OpenDisk**, **SetGEOSDisk**, and **OpenRecordFile**, etc.

Example:

See also: **PutDirHead**

GetFHdrInfo: (C64, C128) C229

Function: Loads the GEOS file header for a particular directory entry.

Parameters: r9 DIRENTRY — pointer to directory entry of file, usually points to **dirEntryBuf** (Apple GEOS: must be in main memory) (word).

Uses: **curDrive**
curType GEOS 64 v 1.3 and later for detecting REU shadowing.

Returns: x error ($00 = no error).
r7 load address copied from the **O_GHST_ADDR** word of the GEOS file header.
rl track/sector copied from bytes +1 and +2 of the directory entry (*DIRENTRY*). This is the track/sector of the first data block of a sequential file (**OFF_DE_TR_SC**) or the index table block of a VLIR file (**OFF_INDEX_PTR**).

Alters: **fileHeader** contains 256-byte GEOS file header.
fileTrScTab track/sector of header added to first two bytes of this table; a subsequent call to **ReadFile** or similar routine will augment this table beginning with the third byte (**fileTrScTab**+2) so as not to disrupt this value.

Destroys: **a,y,r4**

Description: Given a valid directory entry, **GetFHdrInfo** will load the GEOS file header into the buffer at **fileHeader**.

The directory entry is a standard GEOS data structure returned by routines such as **FindFile**, **Get1stDirEntry** and **GetNxtDirEntry**. **GetFHdrInfo** is called by routines such as **LdFile** just prior to calling **ReadFile** (to load in a sequential file or record zero of a VLIR).

GetFHdrInfo gets the block number (Commodore track/sector) of the GEOS file header by looking at the **OFF_GHDR_PTR** word in the directory entry.

Example:

See also:

GetFreeDirBlk: (C64, C128) C1F6

Function: Search the current directory for an empty slot for a new directory entry. Allocates another directory block if necessary.

Parameters: **r10L** DIRPAGE — directory page to begin searching for free slot; each directory page holds eight files and corresponds to one notepad page on the GEOS deskTop. The first page is page one.

Uses:
curDrive
curType GEOS 64 v 1.3 and later for detecting REU shadowing.
curDirHead this buffer must contain the current directory header.
dir2Head2† (BAM for 1571 and 1581 drives only)
dir3Head3† (BAM for 1581 drive only)
interleave† desired physical sector interleave (usually 8); Applications need not set this explicitly — will be set automatically by internal GEOS routines. Only used when new directory block is allocated.

†used internally by GEOS disk routines; applications generally don't use.

Returns:
x error ($00 = no error).
FULL_DIRECTORY
r10L page number of empty directory slot.
r1 block (track/sector) number of directory block in **diskBlkBuf**.
y index to empty directory slot in **diskBlkBuf**.

Alters:
curDirHead contains directory header
dir2Head† (BAM for 1571 and 1581 drives only)
dir3Head† (BAM for 1581 drive only)

Destroys: **a,r0,r3,r5,r7-r8.**

Description: **GetFreeDirBlk** searches the current directory looking for an empty slot for a new directory entry. A single directory page has eight directory slots, and these eight slots correspond to the eight possible files that can be displayed on a single GEOS deskTop notepad page.

GetFreeDirBlk starts searching for an empty slot beginning with page number **DIRPAGE.** If **GetFreeDirBlk** reaches the last directory entry without finding an empty slot, it will try to allocate a new directory block. If DIRPAGE doesn't yet exist, empty pages are added to the directory structure until the requested page is reached.

$01 will most often be passed as the DIRPAGE starting page number, so that all possible directory slots will be searched, starting with the first page. If higher numbers are used, **GetFreeDirBlk** won't find empty directory slots on lower pages and extra directory blocks may be allocated needlessly.

GetFreeDirBlk is called by **SetGDirEntry** before writing out the directory entry for a new GEOS file.

Since GEOS 2.0 does not support a hierarchical file system, the "current directory" is actually the entire disk. A directory page corresponds exactly to a single sector on the directory track. There is a maximum of 18 directory sectors (pages) on a Commodore disk. If this 18th page is exceeded, **GetFreeDirBlk** will return a **FULL_DIRECTORY** error.

GetFreeDirBlk allocates blocks by calling **SetNextFree** to allocate sectors on the directory track. **SetNextFree** will special-case the directory track allocations. Refer to **SetNextFree** for more information.

GetFreeDirBlk does not automatically write out the **BAM.** See **PutDirHead** for more information on writing out the **BAM.**

Example: **MySetGDirEntry**

See also: AllocateBlock, FreeBlock, BlkAlloc

GetOffPageTrSc:	(C64, C128)	**9036**

Function: Get track and sector of off-page directory.

Parameters: none.

Uses:

curDrive drive that disk is in.
curType GEOS 64 v 1.3 and later for detecting REU shadowing.

Returns:

x error ($00 = no error).
y $ff if the disk is not a GEOS disk and therefore has no off-page directory block, otherwise $00.
r1L track of off-page directory.
r1H sector of off-page directory.
r4 pointer to **curDirHead**.

Destroys: **a,y, r5**

Description: Commodore GEOS disks have an extra directory block somewhere on the disk called the off-page directory. The GEOS deskTop uses the off-page directory block to keep track of file icons that have been dragged off of the notepad and onto the border area of the deskTop. The off-page directory holds up to eight directory entries.

GetOffPageTrSc reads the directory header into the buffer at **curDirHead** and calls **ChkDkGEOS** to ensure that the disk is a GEOS disk. If the disk is not a GEOS disk, it returns with $ff in the y register. Otherwise, **GetOffPageTrSc** copies the off-page track/sector from the **OFF_OP_TR_SC** word in the directory header to rl and returns $00 in y.

Example:

```
; Put off-page block into diskBlkBuf
        jsr     GetOffPageTrSc    ; get off-page directory block
        txa                       ; check for error
        bne     99$               ;
        tya                       ; check for GEOS disk
        tax                       ; put in x in case error
        bne     99$               ;
        LoadW   r4,#diskBlkBuf    ; get off-page block
        jsr                       ; return with error in x
99$     rts
```

See also: **PutDirHead**

disk mid-level

LdApplic: (C64, C128) C21D

Function: Load and (optionally) run a GEOS application, passing it the standard application startup flags as if was launched from the deskTop.

Parameters:
r9 DIRENTRY — pointer to directory entry of file, usually points to **dirEntryBuf** (word).
r0L LOAD_OPT:
bit 0: 0 load at address specified in file header; application will be started automatically
1 load at address in r7; application will not be started automatically.
bit 7: 0 not passing a data file.
1 r2 and r3 contain pointers to disk and data file names,
bit 6: 0 not printing data file.
1 printing data file; application should print file and exit
r7 LOAD_ADDR — optional load address, only used if bit 0 of *LOAD_OPT* is set (word).
r2 DATA_DISK — only valid if bit 7 or bit 6 of LOAD_OPT is set: pointer to name of the disk that contains the data file, usually a pointer to one of the **DrXCurDkNm** buffers (word).
r3 DATA_FILE — only valid if bit 7 of LOAD_OPT is set: pointer to name of the data file (word).

Uses:
curDrive drive that disk is in.
curType GEOS 64 v 1.3 and later for detecting REU shadowing.

Returns: *only returns if alternate load address or disk error.*
x error ($00 = no error).

Passes: usually doesn't return, but warmstarts GEOS and passes the following:
r0 as originally passed to **LdApplic**.
r2 as originally passed to **LdApplic** (use **dataDiskName**).
r3 as originally passed to **LdApplic**.(use **dataFileName**).

Alters: GEOS brought to a warmstart state.
dataDiskName contains name of data disk if bit 7 of **r0** is set.
dataFileName contains name of data file if bit 6 of **r0** is set

Destroys: **a,x,y,r0-r15**

Description: **LdApplic** is a mid-level application loading routine called by the higher level **GetFile**. Given a directory entry of a GEOS application file, **LdApplic** will attempt load it into memory and optionally run it. **LdApplic** calls **LdFile** to load the application into memory: a sequential file is loaded entirely into memory but only record zero of a VLIR file is loaded. Based on the status of bit 0 of **LOAD_OPT**, optionally runs the application by calling it through **StartAppl**.

Most applications will not call **LdApplic** directly but will go indirectly through **GetFile**.

Note: Only in extremely odd cases will an alternate load address be specified for an application. Loading an application at another location is not particularly useful because it will most likely not run at an address other than its specifiec load address. When **LdApplic** returns to the caller, it does so before calling **StartAppl** to warmstart GEOS.

Example:

See also: **GetFile, LdDeskAcc, StartAppl**

disk mid-level

LdDeskAcc: (C64, C128) C217

Function: Load and run a .GEOS desk accessory.

Parameters: **r9** DIRENTRY — pointer to directory entry of file, usually points to **dirEntryBuf** (word).
r0L RECVR_OPTS — should be set to $00 (see below for explanation) (byte).

Uses: **curDrive** drive that disk is in.
curType GEOS 64 v 1.3 and later for detecting REU shadowing.

Returns: *returns when desk accessory exits with a call to* **RstrAppl***.*
x error ($00 = no error).

Passes: warmstarts GEOS and passes the following to the desk accessory:
r10L as originally passed to **LdDeskAcc** (should be $00; see below).

Alters: nothing directly; desk accessory may alter some buffers that are not saved.

Destroys: **a,x,y,r0-r15**

Description: **LdDeskAcc** is a mid-level desk accessory loading routine called by the higher level **GetFile**. Given a directory entry of a GEOS desk accessory file, **LdDeskAcc** will attempt load it into memory and run it. When the user closes the desk accessory, control returns to the calling application.

LdDeskAcc first loads in the desk accessory's file header to get the start and ending load address. Under GEOS 64 and Apple GEOS, it will then save out the area of memory between these two addresses to a file on the current disk named "SWAP FILE". The GEOS 128 version saves this area to the 24K desk accessory swap area in back RAM. Desk accessories larger than 24K cannot be used under GEOS 128 (to date, there are none); a **BFR_OVERFLOW** error is returned.

After saving the overlay area, the dialog box and desk accessory save-variables are copied to a special area of memory, the current stack pointer is remembered, and the desk accessory is loaded and executed. When the desk accessory calls **RstrAppl** to return to the application, this whole process is reversed to return the system to a state similar to the one it was in before the desk accessory was called. The "SWAP FILE" file is deleted.

Most applications will not call **LdDeskAcc** directly, but will go indirectly through **GetFile**.

C64 : GEOS versions 1.3 and above have a GEOS file type called **TEMPORARY.** When the deskTop first opens a disk, it deletes all files of this type. The "SWAP FILE" is a **TEMPORARY** file.

Note: The RECVR_OPTS flag originally carried the following significance:

bit 7: 1 force desk accessory to save foreground screen area and restore it on return to application.
0 not necessary for desk accessory to save foreground.

bit 6: 1 force desk accessory to save color memory and restore it on return to application.
0 not necessary for desk accessory to save foreground.

Note: It was found that the extra code necessary to make desk accessories save the foreground screen and color memory provided no real benefit because this context save can just as easily be accomplished from within the application itself. The RECVR_OPTS flag is set to $00 by all Berkeley Softworks applications, and desk accessories can safely assume that this will always be the case. (In fact, future versions of GEOS may force rlOH to $00 before calling desk accessories just to enforce this standard!)

The application should always set **rl0H** to $00 and bear the burden of saving and restoring the foreground screen and the color memory. (Color memory only applicable to GEOS 64 and GEOS 128 in 40-column mode.)

Example:

See also: **GetFile, LdApplic, RstrAppl, RstrFrmDialog.**

disk mid-level

LdFile: (C64, C128) C211

Function: Given a directory entry, loads a sequential file or record zero of a VLIR record.

Parameters: **r9** DIRENTRY — pointer to directory entry of file, usually points to **dirEntryBuf** (word).

Uses: **curDrive** drive that disk is in.
curType GEOS 64 v 1.3 and later for detecting REU shadowing.

Returns: **x** error ($00 = no error).
r7 pointer to last byte read into BUFFER plus one.

Alters: **fileHeader** contains 256-byte GEOS file header. (This is a 512-byte buffer in Apple GEOS, although only 256 bytes are used in the GEOS file header for compatibility).

fileTrScTab track/sector of header in first two bytes of this table (**fileTrScTab**+0 and **fileTrScTab**+1); As the file is loaded, the track/sector pointer to each block is added to the file track/sector table starting at **fileTrScTab**+2 and **fileTrScTab**+3.

Destroys: Not Listed in source material. **LdFile** is in an Unusable state already so this is to be expected.

Description: **LdFile** is a mid-level file handling routine called by the higher level **GetFile.** Given a directory entry of a sequential file, **LdFile** will load it into memory. Given the directory entry of a VLIR file, **LdFile** will load its record zero into memory.

Most applications will not call **LdFile** directly, but will go indirectly through **GetFile.**

All versions of **LdFile** to date under Commodore GEOS are unusable because the load variables that are global under Apple GEOS (loadOpt and loadAddr) are local to the Kernal and inaccessible to applications. Fortunately this is not a problem because applications can always go through **GetFile** to achieve the same effect.

See also: **GetFile, LdApplic, LdDeskAcc.**

NewDisk: (C64, C128) C1E1

Function: Tell the turbo software that a new disk has been inserted into the drive.

Parameters: **r1L**[1] Track to position the disk drive head at.
r1H[1] Sector to position the disk drive head at.

Uses: **curDrive** drive that disk is in.
curType GEOS 64 v 1.3 and later for detecting REU shadowing.

Returns: **x** error ($00 = no error).

Destroys: **a,y, r0-r3**

Description: **NewDisk** informs the disk drive turbo software that a new disk has been inserted into the drive. It first calls **EnterTurbo** then sends an initialize command to the turbo code. If the disk is shadowed, the shadow memory is also cleared.

NewDisk gets called automatically when **OpenDisk** opens a new disk. An application that does not deal with anything but the low-level disk routines might want to call **NewDisk** instead of **OpenDisk** to avoid the unnecessary overhead associated with reading the directory header and initializing internal file-level variables.

Note: **NewDisk** has no effect on a RAMdisk. Also, some early versions of the 1541 turbo code leave the disk in the drive spinning after it is first loaded. A call to **NewDisk** during the application's initialization will stop the disk.

Note:[1] It also positions the head over a particular sector.

Calls:[2] **EnterTurbo, InitForIO, DoneWithIO**

Example:

See also: **OpenDisk, SetDevice**

disk mid-level

NxtBlkAlloc: (C64, C128) C24D

Function: Special version of **BlkAlloc** that begins allocating from a specific block on the disk.

Parameters:
r2 BYTES — number of bytes to allocate space for. Can allocate up to 32,258 bytes (127 blocks). (word)
r3L START_TR — start allocating from this track (byte).
r3H START_SC — start allocating from this sector (byte).
r6 TSTABLE — pointer to buffer for building out track and sector table of the newly allocated blocks (word). *usually a position within* **fileTrScTab**

Uses:
curDrive drive that disk is in.
curDirHead this buffer must contain the current directory header.
dir2Head2† (BAM for 1571 and 1581 drives only)
dir3Head3† (BAM for 1581 drive only)
interleave† desired physical sector interleave (usually 8); used by **SetNextFree.** Applications need not set this explicitly — will be set automatically by internal GEOS routines.
;†used internally by GEOS disk routines; applications generally don't use.

Returns:
x error ($00 = no error).
r2 number of blocks allocated to hold BYTES amount of data.
r3L track of last allocated block.
r3H sector of last allocated block.

Alters:
curDirHead BAM updated to reflect newly allocated blocks.
dir2Head† (BAM for 1571 and 1581 drives only)
dir3Head† (BAM for 1581 drive only)

Destroys: a, y, **r4-r8**

Description: **NxtBlkAlloc** begins allocating blocks from a specific block on the disk, allowing a chain of blocks to be appended to a previous chain while still maintaining the sector interleave. **NxtBlkAlloc** is essentially a special version of **BlkAlloc** that starts allocating blocks from an arbitrary block on the disk rather than from a fixed block. **NxtBlkAlloc** is otherwise identical to **BlkAlloc.**

Use **NxtBlkAlloc** for appending more blocks to a list of blocks just allocated with **BlkAlloc**, thus circumventing the 32,258-byte barrier. Point TSTABLE at the last entry in a track/sector table (the terminator bytes which we can overwrite), load the BYTES parameter with the number of bytes left, and call **NxtBlkAlloc.** The START TR and START_SC parameters in **r3L** and **r3H** will contain the correct values on return from **BlkAlloc. NxtBlkAlloc** will allocate enough additional blocks to hold BYTES amount of data, appending them in the track/sector table automatically. This combined list of track and sectors can then be passed directly to **WriteFile** too write data to the full chain of blocks.

NxtBlkAlloc does not automatically write out the BAM. See **PutDirHead** for more information on writing out the BAM. Also, the START_TR parameter should not be track number of the directory track. Refer to GetFreeDirBlk for more information on allocating blocks on the directory track.

Note: For more information on the scheme used to allocate successive blocks, refer to **SetNextFree.**

Example:

See also: **BlkAlloc, SetNextFree, AllocateBlock, FreeBlock.**

disk mid-level

PutDirHead: (C64, C128) C24A

Function: Write directory header to disk. GEOS also writes out the BAM.

Parameters: none.

Uses:
curDrive drive that disk is in.
curType GEOS 64 v 1.3 and later for detecting REU shadowing.
curDirHead this buffer must contain the current directory header.
dir2Head2† (BAM for 1571 and 1581 drives only)
dir3Head3† (BAM for 1581 drive only)
†used internally by GEOS disk routines; applications generally don't use.

Returns:
x error ($00 = no error).
r4 pointer to **curDirHead**.

Destroys: **a,y, r1**

Description: **PutDirHead** writes the directory header to disk from the buffer at **curDirHead.** GEOS writes out the full directory header block, including the BAM (block allocation map).

GEOS disks, like the standard Commodore disks upon which they are based, have one directory header. The directory header occupies one full block on the disk. The Commodore directory header contains information about the disk, such as the location of the directory blocks, the disk name, and the GEOS version string (if a GEOS disk). The Commodore directory header also contains the disk BAM, which flags particular sectors as used or unused.

PutDirHead calls **PutBlock** to write out the directory header block from the buffer at **curDirHead.** The directory header block contains the directory header and the disk BAM (block allocation map). Applications that are working with the mid- and low-level GEOS disk routines may need to call **PutDirHead** to update the BAM on the disk with the BAM in memory. Many useful, mid-level GEOS routine's, such as **BlkAlloc,** only update the BAM in memory (for speed and ease of error recovery). When a new file is written disk, GEOS allocates the blocks in the in-memory BAM, writes the blocks out using the track sector table, then, as the last operation, calls **PutDirHead** to write the new BAM to the disk. An application that uses the mid-level GEOS routines to build its own specialized disk file functions will need to keep track of the status of the BAM in memory, writing it to disk as necessary.

It is important that the BAM in memory not get overwritten by an outdated BAM on the disk. Applications that manipulate the BAM in memory (or calls GEOS routines that do so), must be careful to write out the new BAM before calling a routine that might overwrite it. Routines that call **GetDirHead** include **OpenDisk**, **SetGEOSDisk**, and **OpenRecordFile**.

GEOS VLIR routines set the global variable **fileWritten** to TRUE to signal that the VLIR file has been written to and that the BAM in memory is more recent than the BAM on the disk. **CloseRecordFile** checks this flag. If **fileWritten** is TRUE, **CloseRecordFile** calls **PutDirHead** to write out the new BAM.

Example:

See also: **GetDirHead.**

ReadByte: (C64, C128) C2B6

Function: Special version of **ReadFile** that allows reading a chained list of blocks a byte at a time.

Parameters: *on initial call only:*
r1 START_TRSC — track/sector of first data block (word).
r4 BLOCKBUF — pointer to temporary buffer of **BLOCKSIZE** bytes for use by **ReadByte**, usually a pointer to **diskBlkBuf** (word).
r5 $0000 (word).

Uses:

curDrive	drive that disk is in.
curType	GEOS 64 v 1.3 and later for detecting REU shadowing.

Returns: **a byte returned**
x error ($00 = no error).
rl, r4, r5 contain internal values that must be preserved between calls to **ReadByte**.

Destroys: **y**

Description: **ReadByte** allows a chain of blocks on the disk to be read a byte at a time. The first time **ReadByte** is called, **rl, r4,** and **r5** must contain the proper parameters. When **ReadByte** returns without an error, the a register will contain a single byte of data from the chain. To read another byte, call **ReadByte** again. Between calls to **ReadByte**, the application must preserve **rl, r4, r5**, and the data area pointed to by BLOCKBUF.

ReadByte loads a block into BLOCKBUF and returns a single byte from the buffer at each call. After returning the last byte in the buffer, **ReadByte** loads in the next block in the chain and starts again from the beginning of BLOCKBUF. This process continues until there are no more bytes in the file. A **BFR_OVERFLOW** error is then returned.

ReadByte is especially useful for displaying very large bitmaps with **BitOtherClip**

Note: Reading a chain a byte at a time involves finding the first data block and passing its track/sector to **ReadFile.** The track/sector of the first data block in a sequential file is returned in **rl** by **GetFHdrlnfo.** The first data block of a VLIR record is contained in the VLIR's index table.

Example:

See also: **OpenDisk, SetDevice**

disk mid-level

ReadFile: (C64, C128) C1FF

Function: Read a chained list of blocks into memory.

Parameters:
r7 BUFFER — pointer to buffer where data will be read into (word).
r2 BUFSIZE — size of buffer Commodore version can read up to 32,258 bytes (127 blocks) (word).
rl START_TRSC — track/sector of first data block (word).

Uses:
curDrive device number of active drive.
curType GEOS 64 v 1.3 and later for detecting REU shadowing.

Returns:
x error ($00 = no error).
r7 Pointer to last byte read into BUFFER plus one.
rl if BFR_OVERFLOW error returned, contains the track/sector of the block that, had it been copied from **diskBlkBuf** to the application's buffer space, would have exceeded the size of BUFFER. The process of copying any extra data from **diskBlkBuf** to the end of BUFFER is left to the application. The data starts at **diskBlkBuf**+2. If no error, then rl is destroyed.

r5L byte index into **fileTrScTab** of last entry (last entry = **fileTrScTab** plus value in r5).

Alters: **fileTrScTab** As the chain is followed, the track/sector pointer to each block is added to the file track/sector table. The track and sector of the first data block is added at **fileTrScTab**+2 and **fileTrScTab**+3, respectively, because the first two bytes (**fileTrScTab**+0 and **fileTrScTab**+1) are reserved for the GEOS file header track/sector.

Destroys: y, (**rl**), **r2-r4** (see above for **rl**).

Description: **ReadFile** reads a chain of blocks from the disk into memory at *BUFFER*. Although the name implies that it reads "files" into memory, it actually reads a chain of blocks and doesn't care whether this chain is a sequential file or a VLIR record — **ReadFile** merely reads blocks until it encounters the end of the chain or overflows the memory buffer.

ReadFile can be used to load VLIR records from an unopened VLIR file. geoWrite, for example, loads different fonts while another VLIR file is open by looking at all the font file index tables and remembering the index information for records that contain font data. When a VLIR document file is open, geoWrite can load a different font by passing one of these saved values in **rl** to **ReadFile**. **ReadFile** will load the font into memory without disturbing the opened VLIR file.

For reading a file when only the filename is known, use the high-level **GetFile**.

Note: The Commodore filing system links blocks together with track/sector links: each block has a two-byte track/sector forward-pointer to the next sector in the chain (or $OO/$ff to signal the end). Reading a chain involves passing the first track/sector to **ReadFile**. The first block contains a pointer to the next block, and so on. The whole chain can be followed by reading successive blocks.

ReadFile reads each 256-byte block into **diskBlkBuf** and copies the 254 data bytes (possibly less in the last block of the chain) to the BUFFER area and copies the two-byte track/sector pointer to **fileTrScTab.** This process is repeated until the last block is copied into the buffer or when there is more data in **diskBlkBuf** than there is room left in *BUFFER*.

When there is more data in **diskBlkBuf** than there is room left in *BUFFER*, **ReadFile** returns with a **BFR_OVERFLOW** error without copying any data into BUFFER. The application can copy data, starting at diskBlkBuf+2, to fill the remainder of *BUFFER* manually.

Because of the limited size of **fileTrScTab** (256 bytes), **ReadFile** cannot load more than 127 blocks of data. (256 total bytes divided by two bytes per track/sector minus two bytes for the GEOS file header equals 127.) 127 blocks can hold 127 * 254 = 32,258 bytes of data.

Example:

See also: **GetFile, WriteFile, ReadRecord.**

disk mid-level

SetGDirEntry: (C64, C128) C1F0

Function: Search for a nearby free block and allocate it.

Parameters:

r10L directory page to begin searching for free slot; each directory page holds eight files and corresponds to one notepad page on the GEOS deskTop. The first page is page one.

r2 NUMBLOCKS — number of blocks in file (word).
r6 TSTABLE — pointer to a track/sector list of unused blocks (unused but allocated in the BAM), usually a pointer to **fileTrScTab**; **BlkAlloc** can be used to build such a list (word).
r9 FILEHDR—pointer to GEOS file header (word).

Uses:

curDrive device number of active drive.
year, month, day, hours, minutes for date-stamping file.
curType GEOS 64 vl.3 and later for detecting REU shadowing
curDirHead this buffer must contain the current directory header.
dir2Head2† (BAM for 1571 and 1581 drives only)
dir3Head3† (BAM for 1581 drive only)
interleave† desired physical sector **interleave** (usually 8). applications need not set this explicitly — will be set automatically by internal GEOS routines. Only used when new directory block is allocated.

†used internally by GEOS disk routines; applications generally don't use.

Returns:

x error ($00 = no error).
r6 pointer to first non-reserved block in track/sector table (**SetGDirEntry** reserves one block for the file header and a second block for the index table if the file is a VLIR file).

Alters:

dirEntryBuf contains newly-built directory entry.
diskBlkBuf used for temporary storage of the directory block.

Destroys: a, y, **rl, r3-r5, r7-r8**.

Description: **SetGDirEntry** calls **BldGDirEntry** to build a system specific directory entry form the GEOS file header, date-stamps the directory entry, calls **GetFreeDirBlk** to find an empty directory slot, and writes the new directory entry out to disk.

Most applications will create new files by calling **SaveFile**. **SaveFile** calls **SetGDirEntry** as part of it's normal processing.

Note[3]: **Required Offsets into GEOS File Header to Set**

Offset	Constant	Size	Description
$00		word	Pointer to Filename
$44	O_GHCMDR_TYPE	byte	DOS File Type
$45	O_GHGEOS_TYPE	byte	GEOS file type
$46	O_GHSTR_TYPE	byte	GEOS file structure type (SEQ or VLIR)

Example:

See also: **GetFile, OpenRecordFile.**

SetNextFree: (C64, C128) C292

Function: Builds a system specific directory entry from a GEOS file header, date-stamps it, and writes it out to the current directory.

Parameters: **r3** block (track/sector) to begin search (word).

Uses:
curDrive device number of active drive.
curDirHead This buffer must contain the current directory header.
dir2Head2† (BAM for 1571 and 1581 drives only)
dir3Head3† (BAM for 1581 drive only)
interleave† Desired physical sector **interleave** (usually 8). applications need not set this explicitly — will be set automatically by internal GEOS routines.

†used internally by GEOS disk routines; applications generally don't use.

Returns:
x error ($00 = no error).
r3 block (Commodore track/sector) allocated.

Alters:
curDirHead BAM updated to reflect newly allocated blocks.
dir2Head† (BAM for 1571 and 1581 drives only)
dir3Head† (BAM for 1581 drive only)

Destroys: a, y, **r6-r7**, **r8H**.

Description: Given the current block as passed in r3, **SetNextFree** searches for the next free block on the disk. The "next" free block is not necessarily adjacent to the previous block because **SetNextFree** may interleave the blocks. Proper interleaving allows the drive to read and write data as fast as possible because it minimizes the time the drive spends waiting for a block to spin under the read/write head. It means, however, that sequential data blocks may not occupy adjacent blocks on the disk. As long as an application is using the standard GEOS file structures, this interleaving should not be apparent.

After determining the ideal sector from any interleave calculations, **SetNextFree** tries to allocate the block it if it is unused. If the block is used, **SetNextFree** picks another nearby sector (jumping to another track if necessary) and calls tries again. This process continues until a block is actually allocated or the end of the disk is reached, whichever comes first. If the end of the disk is reached, an **INSUFFICIENT_SPACE** error is returned.

Notice that **SetNextFree** only searches for free blocks starting with the current block and searching towards the end of the disk. It does not backup to check other areas of the disk because it assumes they have already been filled. (Actually, under Commodore GEOS, **SetNextFree** will backtrack as far back as beginning of the current track but will not go to any previous tracks.). Usually this is a safe assumption because **SetNextFree** is called by **BlkAlloc**, which always begins searching for free blocks from the beginning of the disk.

It is conceivable, however, that an application might want to implement an **AppendRecord** function (or something of that sort), which would append a block of data to an already existing VLIR record without deleting, reallocating, and then rewriting the record like **WriteRecord**.

In order to maintain any interleave from the last block in the record to the new block, the **AppendRecord** routine passes the track and sector of the last block in the record to **SetNextFree**. **SetNextFree** will start searching from this block. If a free block cannot be found, an **INSUFFICIENT_SPACE** error is returned Since **SetNextFree** only searched from the current block to the end of the disk, the possibility exists that a free block lies somewhere on a previous, still unchecked disk area. The following alternative to **SetNextFree** will circumvent this problem:

```
MySetNextFree:
 ;--- Look for a free block starting at the current block
 ;--- so that we continue the interleave if possible
   jsr   SetNextFree                 ; look for block to allocate
   cpx   #INSUFFICENT_SPACE          ; check for no blocks
   beq   10$                         ; start from beginning if none
   rts                               ; exit on any other error or
                                     ; valid block found.

 ;--- We got an insufficient space error. Start the search
 ;--- again from the beginning of the disk.
10$
   LoadB r3H,#0                      ; always sector 0
   ldx   #1                          ; assume track 1
   ldy   curDrive                    ; but special case 1581
   lda   driveType-8,y               ; because of outer/inner track
   and   #$0F                        ; searching scheme
   cmp   DRV_1581
   bne   20$                         ; branch if not 1581
   ldx   #39                         ; 1581 counts down on inner (39-1)
20$
   stx   r3L                         ; track number
   jmp   SetNextFree
```

Note: **SetNextFree** uses the value in interleave to establish the ideal next sector. A good interleave will arrange successive sectors so as to minimize the time the drive spends stepping the read/write head and waiting for the desired sector to spin around. The value in interleave is usually set by the Configure program and internally by GEOS disk routines. The application will usually not need to worry about the value in interleave.

Because Commodore disks store the directory on special tracks, **SetNextFree** will automatically skip over these special tracks unless **r3L** is started on one of these tracks, in which case **SetNextFree** assumes that this was intentional and a block on the directory track is allocated. (This is exactly how **GetFreeDirBlk** operates.) The directory blocks for various drives can be determined by the following constants:

1581	DIR_1581_TRACK	$28	(one track)
1541	DIR_TRACK	$12	(one track)
1571	DIR_TRACK DIR_TRACK+N_TRACKS	$12 $12+$23	(two tracks)

SetNextFree does not automatically write out the BAM. See **PutDirHead** for more information on writing out the BAM.

Example:

See also: **GetFile, OpenRecordFile.**

disk mid-level

StartAppl: (C64, C128) C22F

Function: Warmstart GEOS and start an application that is already loaded into memory.

Parameters: *These are all passed on to the application being started.*
r7 START_ADDR — start address of application (word).
r0L OPTIONS:
bit 7: 0 not passing a data file.
1 **r2** and **r3** contain pointers to disk and data file names,
bit 6: 0 not printing data file.
1 printing data file; application should print file and exit
r2 DATA_DISK — only valid if bit 7 or bit 6 of *OPTIONS* is set: pointer to name of the disk that contains the data file, usually a pointer to one of the **DrXCurDkNm** buffers (word).
r3 DATA_FILE — only valid if bit 7 of *OPTIONS* is set: pointer to name of the data file (word).

Returns: *never returns.*

Passes: *warmstarts GEOS and passes the following to the application at START_ADDR:*

Alters: GEOS brought to a warmstart state.
r0 as originally passed to **StartAppl**.
r2 as originally passed to **StartAppl** (use **dataDiskName**).
r3 as originally passed to **StartAppl**.(use **dataFileName**).
dataDiskName contains name of data disk if bit 7 of r0 is set.
dataFileName contains name of data file if bit 6 of r0 is set

Destroys: **n/a**

Description: **StartAppl** warmstarts GEOS and jsr's to START\ADDR as if the application had been loaded from the deskTop. **GetFile** and **LdApplic** call **StartAppl** automatically when loading an application.

StartAppl is useful for bringing an application back to its startup state. It completely warmstarts GEOS, resetting variables, initializing tables, clearing the processor stack, and executing the application's initialization code with a jsr from **MainLoop**.

Example:

See also: **LdApplic, GetFile**

disk mid-level

WriteFile: (C64, C128) C1F9

Function: Write data to a chained list of disk blocks.

Parameters: *These are all passed on to the application being started.*
r7 DATA — pointer to start of data (word).
r6 TSTABLE — pointer to a track/sector list of blocks to write data to (unused but allocated in the BAM), usually a pointer to **fileTrScTab**+2; **BlkAlloc** can be used to build such a list.

Uses: **curDrive** device number of active drive.
curType GEOS 64 v 1.3 and later for detecting REU shadowing.

Returns: x error ($00 = no error).

Destroys: **a, y, rl-r2,r4,r6-r7.**

Description: **WriteFile** writes data from memory to disk. The disk blocks are verified, and any blocks that don't verify are rewritten.

Although the name "**WriteFile**" implies that it writes "files," it actually writes a chain of blocks and doesn't care if this chain is an entire sequential file or merely a VLIR record.

Note: **WriteFile** uses the track/sector table at TSTABLE as a list of linked blocks that comprise the chain. The end of the chain is marked with a track/sector pointer of $OO,$FF. **WriteFile** copies the next 254 bytes from the data area to **diskBlkBuf**+2, looks two-bytes ahead in the TSTABLE for the pointer to the next track/sector, and copies those two-bytes to **diskBlkBuf**+0 and **diskBlkBuf**+1. **WriteFile** then writes this block to disk. This is repeated until the end of the chain is reached

WriteFile does not flush the BAM (it does not alter it either — it assumes the blocks in the track/sector table have already been allocated). See **BlkAlloc**, **SetNextFree**, and **AllocateBlock** for information on allocating blocks. See **PutDirHead** for more information on writing out the BAM.

Example:

See also: **SaveFile, WriteRecord, ReadFile.**

disk high-level

disk high-level

DeleteFile: (C64, C128) C238

Function: Delete a GEOS file by deleting the its directory entry and freeing all its blocks. Works on both sequential and VLIR files.

Parameters: **r0** FILENAME — pointer to null-terminated name of file to delete

Uses:

curDrive	
curType	GEOS 64 v 1.3 and later for detecting REU shadowing.

Returns: x error ($00 = no error).

Alters:

diskBlkBuf	used for temporary block storage
dirEntryBuf	deleted directory entry.
fileHeader	temporary storage of index table when deleting a VLIR file.
Written to Disk:	
curDirHead	BAM updated to reflect newly freed blocks.
dir2Head	(BAM for 1571 and 1581 drives only)
dir3Head	(BAM for 1581 drive only)

Destroys: **a, y, r0-r9.**

Description: Given a null-terminated filename, **DeleteFile** will remove it from the current directory by deleting its directory entry and calling **FreeFile** to free all the blocks in the file.

DeleteFile first calls **FindFile** to get the directory entry and ensure the file does in fact exist. If the file specified with FILENAME is not found, a **FILE_NOT_FOUND** error is returned.

The directory entry is deleted by setting its **OFF_CFILE_TYPE** byte to $00.

Example:

EnterDeskTop: (C64, C128) C22C

Function: Standard application exit to GEOS deskTop.

Parameters: none.

Returns: *never returns to application.*

Description: **EnterDeskTop** takes no parameters and looks for a copy of the file DESK TOP on each drive. Later versions of GEOS are only compatible with the correspondingly later revision of the deskTop and will check the version number in the permanent name string of the DESK TOP file to ensure that it is in fact a newer version. If after all drives are searched no valid copy of the deskTop is found, **EnterDeskTop** will prompt the user to insert a disk with a copy of the deskTop on it.

Example:

See also: RstrAppl, GetFile.

FindFile: (C64, C128) C20B

Function: Search for a particular file in the current directory.

Parameters: r6 **FILENAME** — pointer to null-terminated name of file of a maximum of 16 bytes (not counting null terminator). (word).

Uses: **curDrive**
curType GEOS 64 vl.3 and later for detecting REU shadowing
$886E[1] Flag byte

Returns: **x** error ($00 = no error).
r1 track/sector of directory block containing entry;
r5 Pointer to directory entry within diskBIkBuf.

Alters: **dirEntryBuf** directory entry of file if found.
diskBlkBuf contains directory block where FILENAME found.

Destroys: **a,y,r4,r6**

Description: Given a null-terminated filename, **FindFile** searches through the current directory and returns the directory entry in **dirEntryBuf**. If the file specified with **FILENAME** is not found, a **FILE_NOT_FOUND** error is returned.

Since Commodore GEOS 2.0 does not support a hierarchical file system, the current directory is actually the entire disk. The directory entry is deleted by setting its OFF_CFILE_TYPE byte to $00.

Note:[1] If the flag byte at $886E is $FF, then both drives 8 and 9 will be scanned if necessary. If the flag is $00, then the lookup is only to the current drive. If there is only one drive, then this flag has no effect.

Note:[3] **Wheels, Gateway and MP3 All support** a hierarchical file system. As of this writing, the Author or this document (PBM) does not yet know the details of this support. This section, (and many others I am sure) will be updated when I have researched these systems.

ANote: (Note to Author: Confirm behavior of and Give/Get a Name if confirmed the 886E Flag address)

Example: **LoadBASIC**

See also: GetlstDirEntry, GetNxtDirEntry, FindFTypes

FindFTypes: (C64, C128) C23B

Function: Builds a list of files of a particular GEOS type from the current directory.

Parameters:
- **r6** BUFFER — pointer to buffer for building-out file list; allow ENTRY_SIZE+1 bytes for each entry in the list (word).
- **r10** PERMNAME — pointer to permanent name string to match or $0000 to ignore permanent name string (word).
- **r7H** MAXFUJES — maximum number of filenames to return, usually used to prevent overwriting buffer.
- **r7L** FILETYPE — GEOS file type to search for (byte).

Uses:
- **curDrive**
- **curType** GEOS 64 vl.3 and later for detecting REU shadowing

Returns:
- **x** error ($00 = no error).
- **r7H** decremented once for each file name (Apple GEOS: high-bit is always cleared).

Alters: **diskBlkBuf** used as temporary buffer for directory blocks.

Destroys: **a, y, rO-r2L, r4, r6.**

Description: **FindFTypes** build a list of files that match a particular GEOS file type and, optionally, a specific permanent name string.

The data area at BUFFER, where the list is built-out, must be large enough to accommodate MAXFILES filenames of ENTRY_SIZE+1 bytes each.

FindFTypes first clears enough of the area at BUFFER to hold MAXFILES filenames then calls **Get1stDirEntry** and **GetNxtDirEntry** to go through each directory entry in the current directory. When the GEOS file type of a directory entry matches the FILETYPE parameter, **FindFTypes** goes on to check for a matching permanent name string.

If the PERMNAME parameter is $0000, then this check is bypassed and the filename is added to the list. If the PERMNAME parameter is non-zero, the null terminated string it points to is checked, character-by-character, against the permanent name string in the file's header block. Although the permanent name string in the GEOS file header is 16 characters long, the comparison only extends to the character before the null-terminator in the string at PERMNAME.

Since permanent name strings typically end with Vx.x, where x.x is a version number (e.g., 2.1), a shorter string can be passed so that the specific version number is ignored. For example, a program called geoQuiz version 1.3 might use "geoQuiz V1.3" as the permanent name string it gives its data files. When geoQuiz version 3.0 goes searching for its data files, it can pass a PERMNAME string of "geoQuiz V" so data files for all versions of the program will be added to the list

When a match is found, the filename is copied into the list at BUFFER. The filenames are placed in the buffer as they are found (the same order they appear on the pages of the deskTop notepad). With a small buffer, matching files on higher-numbered pages may never get added to the list.

Note: Since Commodore GEOS does not support a hierarchical file system, the "current directory" is actually the entire disk. The filenames appear in the list null terminated even though they are padded with $a0 in the directory.

Example:

disk high-level

GetFile: (C64, C128) C208

Function: General-purpose file routine that can load an application, desk accessory, or data file.

Parameters: **r6** **FILENAME**— pointer to null-terminated filename (word).

When loading an application:

r0L LOAD_OPT:

bit 0: 0 load at address specified in file header; application will be started automatically

1 load at address in r7; application will not be started automatically.

bit 7: 0 not passing a data file.

1 **r2** and **r3** contain pointers to disk and data file names.

bit 6: 0 not printing data file.

1 printing data file; application should print file and exit

r7 LOAD_ADDR — optional load address, only used if bit 0 of LOADJOPT is set (word).

r2 DATA_DISK — only valid if bit 7 or bit 6 of LOAD_OPT is set: pointer to name of the disk that contains the data file, usually a pointer to one of the **DrXCurDkNm** buffers (word).

r3 DATA_FILE — only valid if bit 7 or bit 6 of LOAD_OPT is set: pointer to name of the data file (word).

When loading a desk accessory:

rl0L RECVR_OPTS — no longer used; set to $00 (see below for explanation (byte).

Uses: **curDrive**

curType GEOS 64 vl.3 and later for detecting REU shadowing

Returns: When loading an application:

only returns if alternate load address or disk error.

x error ($00 = no error).

r0, **r2**, **r3**, and **r7** unchanged.

When loading a desk accessory:

returns when desk accessory exits with a call to **RstrAppl**

x error ($00 = no error).

When loading a data file:

x error ($00 = no error).

Passes: When loading an application:

warmstarts GEOS and passes the following to the application

r0 as originally passed to **GetFile**.

r2 as originally passed to **GetFile** (use **dataDiskName**).

r3 as originally passed to **GetFile**.(use dataFileName).

dataDiskName contains name of data disk if bit 7 of **r0** is set

dataFileName contains name of data file if bit 6 of **r0** is set

When loading a desk accessory:

warmstarts GEOS and passes the following:

rl0L as originally passed to **GetFile**.

See also: FindFile, GetlstDirEntry, GetNxtDirEntry.

When loading a data file:
not applicable.

Alters: When loading an application:
GEOS brought to a warmstart state.

Destroys: **a,x,y,r0-r10** (only applies to loading a data file).

Description: **GetFile** is the preferred method of loading most GEOS files, whether a data file, application, or desk accessory. (The only exception to this is a VLIR file, which is better handled with the VLIR routines such as **OpenRecordFile** and **ReadRecord**). Most applications will use **GetFile** to load and execute desk accessories when the user clicks on an item in the geos menu. Some applications will use **GetFile** to load other applications. The GEOS deskTop, in fact, is just another application like any other. Depending on the user's choice of actions — open an application, open an application's data file, print an applications' data file — the deskTop sets *LOAD_OPT, DATA_DISK, DATA_FILE* appropriately and calls **GetFile.**

GetFile first calls **FindFile** to locate the file at FILENAME, then checks the GEOS file type in the directory entry. If the file is type **DESK_ACC**, then **GetFile** calls **LdDeskAcc.** If the file is type **APPLICATION** or type **AUTO_EXEC**, **GetFile** calls **LdApplic.** All other file types are loaded with the generic **LdFile.**

The following GEOS constants can be used to set the LOAD_OPT parameter when loading an application:

ST_LD_AT_ADDR	$01	Load at address: load application at the address passed in **r7** as opposed to the address in the file header.
ST_LD_DATA	$80	Load data file: application is being passed the name of a data file to load.
ST_PR_DATA	$40	Print data file: application is being passed the name of a data file to print.

Note: The RECVR_OPTS flag used when loading desk accessories originally carried the following significance:

bit 7: 1 force desk accessory to save foreground screen area and restore it on return to application.
0 not necessary for desk accessory to save foreground.
bit 6: 1 force desk accessory to save color memory and restore it on return to application.
0 not necessary for desk accessory to save color memory.

The application should always set r10H to $00 and bear the burden of saving and restoring the foreground screen and the color memory. (Color memory only applicable to GEOS 64 and GEOS 128 in 40-column mode.)
See **LdDeskAcc Note** for more information.

Example:

See also: LdFile, LdDeskAcc, LdApplic

GetPtrCurDkNm: (C64, C128) **C298**

Function: Search for a particular file in the current directory.

Parameters: **x** PTR — zero-page address to place pointer (byte pointer to a word variable).

Uses: **curDrive**

Returns: **x** error ($00 = no error).
zero-page word at $00,x (PTR) contains a pointer to the current disk name.

Destroys: **a,y**

Description: **GetPtrCurDkNm** returns an address that points to the name of the current disk. Disk names are stored in the **DrXCurDkNm** variables, where x designates the drive (A, B, C, or D). If drive A is the current drive then **GetPtrCurDkNm** would return the address of **DrACurDkNm.** If drive B is the current drive then **GetPtrCurDkNm** would return the address of **DrBCurDkNm.** And so on.

Although the locations of the **DrXCurDkNm** buffers are at fixed memory locations, they are not contiguous in memory. It is easier to call **GetPtrCurDkNm** than hardcode the addresses into the application. This will also ensure upward compatibility with future versions of GEOS that might support more drives.

C64: Versions of GEOS before v 1.3 only support two disk drives and therefore only have two disk name buffers allocated (**DrACurDkNm** and **DrBCurDkNm**). GEOS vl.3 and later support additional drives C and D. **GetPtrCurDkNm** will return the proper pointer values in any version of GEOS as long as **numDrives** does not exceed the number of disk name buffers. Trying to get a pointer to **DrDCurDkNm** under GEOS vl.2 will return an invalid pointer because the buffer does not exist

C64 & C128: Commodore disk names are always a fixed-length 16 character string. If the name is less than 16 characters, the string is padded with **$AO.**

Example: **KeyTrap**

See also:

OpenDisk: (C64, C128) C2A1

Function: Open the disk in the current drive

Parameters: None:

Uses:
curDrive drive that disk is in. Set by call **SetDevice**
driveType type of drive to open (for shadowing information)

Calls: **NewDisk, GetDirHead , ChkDkGEOS, GetPtrCurDkNm**

Returns: **x** error ($00 = no error).
r5 pointer to disk name buffer as returned from **GetPtrCurDkNm.** This is a pointer to one of the **DrXCurDkNm** arrays.

Alters:
Dn*x*CurDkNm current disk name array contains disk name
curDirHead current directory header
isGEOS set to TRUE if disk is a GEOS disk, otherwise set to FALSE.
dir2Head (BAM for 1571 and 1581 drives only)
dir3Head (BAM for 1581 drive only)

Destroys: **a, y, r0-r4.**

Description: **OpenDisk** initiates access to the disk in the current drive. **OpenDisk** is meant to be called after a new disk has been inserted into the disk drive. It prepares the drive and disk variables for dealing with a new disk. An application will usually call **OpenDisk** immediately after calling **SetDevice**

Note: Because GEOS uses the same allocation and file buffers for each drive, it is important to close all files and update the BAM if necessary (use **PutDirHead**) before accessing another disk.

OpenDisk first calls **NewDisk** to tell the disk drive a new disk has been inserted (if the disk is shadowed, the shadow memory is also cleared). **GetDirHead** is then called to load the disk's header block and BAM into **curDirHead.** With a valid header block in memory, **ChkDkGEOS** is called to check for the GEOS I.D. string and set the **isGEOS** flag to TRUE if the disk is a GEOS disk. Finally, **OpenDisk** copies the disk name string from **curDirHead** to the disk name buffer returned by **GetPtrCurDkNm.**

Note: This Routine calls **GetDirHead** which loads in the BAM from disk. **PutDirHead** should be called prior to this routine if the BAM has been modified by Freeing or allocating blocks.

Example:

See also: DeleteDir, FreeDir, FreeFile, FreeBlock, SetDevice.

RenameFile: (C64, C128) C259

Function: Renames a file that is in the current directory.

Parameters:
r6 OLDNAME — pointer to null-terminated name of file as it appears on the disk (word).
r0 NEWNAME — pointer to new null-terminated name (word).

Uses:
curDrive drive that disk is in. Set by call **SetDevice**
driveType type of drive to open (for shadowing information)

Calls: **NewDisk, GetDirHead , ChkDkGEOS, GetPtrCurDkNm**

Returns: **x** error ($00 = no error).

Alters:
diskBlkBuf used for temporary block storage.
dirEntryBuf old directory entry.
curDirHead BAM updated to reflect newly freed blocks.
dir2Head (BAM for 1571 and 1581 drives only)
dir3Head (BAM for 1581 drive only)

Destroys: **a, y, r4-r6.**

Description: **RenameFile** searches the current directory for OLDFILE and changes the name string in the directory entry to NEWFILE.

RenameFile first calls **FindFile** to get the directory entry and ensure the OLDFILE does in fact exist. (If it doesn't exist, a **FILE_NOT_FOUND** error is returned.)

The directory entry is read in, the new file name is copied over the old file name, and the directory entry is rewritten. The date stamp of the file is not changed.

When using **Get1stDirEntry** and **GetNxtDirEntry** to establish the old file name, do not pass **RenameFile** a pointer into **diskBlkBuf.** Copy the file name from **diskBlkBuf** to another buffer (such as **dirEntryBuf**) and pass **FreeFile** the pointer to that buffer. Otherwise when **FreeFile** uses **diskBlkBuf** it will corrupt the file name.

Note[3]: This Routine calls **FindFile** which loads in the BAM in from disk. it is important to close all VLIR files and update the BAM if necessary (use **PutDirHead**) before using **RenameFile.**

Example:

See also: FreeFile, FreeBlock.

RstrAppl: (C64, C128) C23E

Function: Standard desk accessory return to application.

Parameters: none:

Uses: **curDrive** drive that disk is in. Set by call **SetDevice**

Returns: *never returns to desk accessory.*

Description: A desk accessory calls **RstrAppl** when it wants to return control to the application that called it. **RstrAppl** loads the swapped area of memory from the **SWAP FILE**, restores the saved state of the system from the internal buffer, resets the stack pointer to its original position, and returns control to the application.

It is the job of the desk accessory to ensure that if the current drive (curDrive) is changed that it be returned to its original value so that **RstrAppl** can find **SWAP FILE**. Under Apple GEOS it is not necessary to save the current directory.

Note: If a disk error occurs when reading in **SWAP FILE**, the remainder of the context switch (restoring the state of the system, etc.) is bypassed and control is immediately returned to the caller of the desk accessory. The application will have only a moderate chance to recover, however, because the area of memory that the desk accessory overlayed may very well include the area where the jsr to **GetFile** or **LdDeskAcc** resides. The return, therefore, may end up in the middle of desk accessory code.

Example:

See also: StartAppl, GetFile.

SaveFile: (C64, C128) C1ED

Function: create a GEOS sequential OR VLIR file and save a region of memory.

Parameters: **r9** HEADER pointer to GEOS file header for file.
r10L DIRPAGE Directory page to begin searching for an empty directory slot.

Uses: **curDrive** device number of active drive.
year, month, day, hours, minutes for date-stamping file.
curType GEOS 64 v1.3 and later for detecting REU shadowing
interleave desired physical sector **interleave** (usually 8).

Returns: **x** error ($00 = no error).
r1 Track and Sector of last block written
r9 Unchanged
r6 pointer to **fileTrScTab**

Alters: **dirEntryBuf** contains newly-built directory entry.
diskBlkBuf contains contents of last block written
fileTrScTab $00-$01 contain T/S of File Header.
End of Table is marked with Track**=**0
curDirHead BAM updated to reflect newly allocated block.
dir2Head (BAM for 1571 and 1581 drives only)
dir3Head (BAM for 1581 drive only)

Destroys: **a,y, r0-r8**

Description: **SaveFile** is the most general purpose write data routine in GEOS. It creates a new file, either sequential or VLIR with a Header Block. VLIR files will have all of the memory written to Record 0 of the VLIR.

SaveFile calls **SetGDirEntry** and **BlkAlloc** to construct the file, then calls **WriteFile** to put the data into it. **After the file is saved, the BAM is written to disk**

Note[1,3]: If the Start Address = $0000 and the End Address = $FFFF (Or if Start Address = End Address) no data blocks are written. A VLIR's VLIR block will have all empty records. An empty SEQ file's directory entry will have a start T/S of 00/FF. (This is not a normal valid state for a SEQ file and should have at least one block added to it).

Note[3]: The HEADER holds all the information needed to create the file. All of the information listed as Required must be populated in the HEADER.

Required Offsets into GEOS File Header to Set

Offset	Constant	Size	Description
$00		word	Pointer to Filename
$44	O_GHCMDR_TYPE	byte	DOS File Type
$45	O_GHGEOS_TYPE	byte	GEOS file type
$46	O_GHSTR_TYPE	byte	GEOS file structure type (SEQ or VLIR)
$47	O_GHST_ADDR	word	Memory to Save Start Address *note: (Set to $0000 for an empty file)*
$49	O_GHEND_ADDR	word	Memory to Save End Address *note: (Set to $FFFF for an empty file)*

Example:

See also: **GetFile, OpenRecordFile.**

SetDevice: (C64, C128) C2B0

Function: Establish communication with a new peripheral

Parameters: a DEVNUM — 8,9,10,11 (DRIVE A through DRIVE D) for disk drives, PRINTER for serial printer, or any other valid serial device bus address (byte).

Uses: **curDevice** currently active device.

Returns: **x** error ($00 = no error).

Alters: **curDevice** new current device number.
curDrive new current drive number if device is a disk drive.
curType GEOS vl.3 and later: current drive type (copied from **driveType** table).

Destroys: **a,y**

Description: **SetDevice** changes the active device and is used primarily to switch from one disk drive to another. **SetDevice** also allows a printer driver to gain access to the serial bus by using a DEVNUM value of PRINTER.

Each I/O device has an associated device number that distinguishes its I/O from devices. At any given time only one device is active. The active device is called the current device and to change the current device an application calls **SetDevice**.

SetDevice is designed to switch between serial bus devices, DEVNUM reflects the architecture of serial bus: disk drives are numbered 8 through 11 and the printer is numbered 4. However, not all I/O devices are actual serial bus peripherals. A RAMdisk, for example, uses a special device driver to make a cartridge port RAM-expansion unit emulate a Commodore disk drive. **SetDevice** switches between these devices just as if they were daisy chained off of the serial bus.

GEOS up through vl.2 supports two disk devices, DRIVE A and DRIVE B. Commodore GEOS vl.3 and later supports up to four disk devices, DRIVE~A through DRIVE-D. Desktop Only Supports 3 Devices.

Note: **SetDevice** calls **ExitTurbo** so that the old device is no longer actively sensing the serial bus, then installs the new device driver as necessary to make the new device (DEVNUM) the current device. With more than one type of device attached (e.g., a 1541 and a 1571), GEOS must switch the device drivers, making the driver for the selected device active. GEOS stores inactive device drivers in the Commodore 128 back RAM and in special system areas in an REU. GEOS applications must use **SetDevice** to change the active device. An application should never directly modify **curDrive** or **curDevice**.

Example: **KeyTrap**

See also: OpenDisk, ChangeDiskDevice

SetGEOSDisk: (C64, C128) C1EA

Function: Convert Commodore disk to GEOS format.

Parameters: none.

Uses: **curDrive**
curType GEOS 64 vl.3 and later for detecting REU shadowing.

Returns: **x** error ($00 = no error).

Alters: **curDirHead** directory header is read from disk.
dir2Head (BAM for 1571 and 1581 drives only)
dir3Head (BAM for 1581 drive only)

Destroys: **a,y**

Description: **SetGEOSDisk** converts a standard Commodore disk into GEOS format by writing the GEOS ID string to the directory header (at **OFF_GEOS ID**) and creating an off-page directory block. An application can call **SetGEOSDisk** after **OpenDisk** returns the **isGEOS** flag set to FALSE. Typically the user is prompted before the conversion.

SetGEOSDisk expects the disk to have been previously opened with **OpenDisk.** It first calls **GetDirHead** to read the directory header into memory then calls **CalcBlksFree** to see if there is block available for the off-page directory (if there isn't, an **INSUFFICIENT_SPACE** error is returned). **SetNextFree** is then called to allocate the off-page directory block. The off-page directory block is written with empty directory entries and a pointer to it is placed in the directory header (at **OFF_OP_TR_SC**). Finally **PutDirHead** is called to write out the new BAM and directory header.

Example:

See also: ChkDkGEOS

disk VLIR

AppendRecord: (C64, C128) **C289**

Function: Adds an empty record after the current record in the index table, moving all subsequent records down one slot to make room.

Parameters: none.

Uses:

curDrive	drive that disk is in. Set by call **SetDevice**
fileWritten†	if FALSE, assumes record just opened (or updated) and reads BAM/VBM into memory. *ANOTE: Confirm*
curRecord	Current record number
fileHeader	VLIR index table.
curType	GEOS 64 v1.3 and later for detecting REU shadowing
curDirHead	BAM updated to reflect newly allocated block.
dir2Head†	(BAM for 1571 and 1581 drives only)
dir3Head†	(BAM for 1581 drive only)

†used internally by GEOS disk routines; applications generally don't use.

Returns: **x** error ($00 = no error).
OUT_OF_REC0RDS

Alters:

curRecord	new record number
usedRecords	number of records in file that are currently in use.
fileWritten†	set to TRUE to indicate the file has been altered since last updated.
fileHeader	buffer contains VLIR index table.

note: *When making manual changes to the VLIR setting* **fileWritten** *to* ***TRUE*** *will cause* **CloseRecordFile** *to write the changes to disk.*

Destroys: **a,y, r0L, r1L, r4**

Preparatory routines[1]: OpenRecordFile

Description: **AppendRecord** inserts an empty VLIR record following the current record in the index table of an open VLIR file, moving all subsequent records down in the record list. The new record becomes the current record. A VLIR file can have a up to **MAX_VLIR_RECS** records (127 on the Commodore and 254 on the Apple). If adding a Record exceeds this value, then an **OUT_OF_RECORDS** error is returned.

A record added with **AppendRecord** occupies no disk space until data is written to it. The new record is marked as empty in the VLIR index table ($00 $FF). When a VLIR file is first created by SaveFile, all records are marked as unused ($00 $00). Some applications call **AppendRecord** repeatedly after creating a new file until an **OUT_OF_RECORDS** error is returned This marks all the records as used and prepares them to accept data with calls to **WriteRecord**.

Note: **AppendRecord** does not write the VLIR index table out to the disk. Call **CloseRecordFile** or **UpdateRecordFile** to save the index table when all modifications are complete.

Note: Use **PointRecord** to check the status of a particular record (unused, empty, or filled).

Example: **SaveRecord**

See also: InsertRecord, DeleteRecord, PointRecord

CloseRecordFile: (C64, C128) C277

Function: Close the current VLIR file (updating it in the process) so that another may be opened

Parameters: none.

Uses:

curDrive	drive that disk is in. Set by call **SetDevice**
fileWritten†	if FALSE, assumes record just opened (or updated) and reads BAM/VBM into memory. *ANOTE: Confirm*
fileHeader	VLIR index table.
fileSize	total number of disk blocks used in file (includes index block, GEOS file header, and all records).
curType	GEOS 64 v1.3 and later for detecting REU shadowing
curDirHead	BAM updated to reflect newly allocated block.
dir2Head†	(BAM for 1571 and 1581 drives only)
dir3Head†	(BAM for 1581 drive only)

†used internally by GEOS disk routines; applications generally don't use.

Returns: **x** error ($00 = no error).

Alters:

fileWritten†	set to TRUE to indicate the file has been altered since last updated.
fileHeader	buffer contains VLIR index table.

note: *When making manual changes to the VLIR setting* **fileWritten** *to TRUE will cause* **CloseRecordFile** *to write the changes to disk.*

Destroys: **a,y, r1, r4, r5.**

Preparatory routines[1]: OpenRecordFile

Description: **CloseRecordFile** first calls **UpdateRecordFile** then closes the VLIR file so that another may be opened.

Because Commodore GEOS stores the BAM in global memory, the application must be careful not to corrupt it before the VLIR file is updated or closed. For more information, refer to **UpdateRecordFile**.

Example: **SaveRecord**

See also: OpenRecordFile, UpdateRecordFile

DeleteRecord: (C64, C128) C283

Function: Removes the current VLIR record from the record list, moving all subsequent records upward to fill the slot and freeing all the data blocks associated with the record.

Parameters: none.

Uses:

curDrive	drive that disk is in. Set by call **SetDevice**
fileWritten†	if FALSE, assumes record just opened (or updated) and reads BAM into memory.
curRecord	Current record number
fileHeader	VLIR index table.
curType	GEOS 64 vl.3 and later for detecting REU shadowing
curDirHead	BAM updated to reflect newly allocated block.
dir2Head†	(BAM for 1571 and 1581 drives only)
dir3Head†	(BAM for 1581 drive only)

†used internally by GEOS disk routines; applications generally don't use.

Returns: **x** error ($00 = no error).

Alters:

curRecord	only changed if deleting the last record in the table, in which case it becomes the new last record.
fileWritten	set to **TRUE** to indicate the file has been altered since last updated.
fileHeader	Record Marked as empty ($00 $FF)
fileSize	decremented to reflect any deleted record blocks.
curDirHead	current directory header/BAM modified to free blocks.
dir2Head†	(BAM for 1571 and 1581 drives only)
dir3Head†	(BAM for 1581 drive only)

Destroys: **a,y, r0-r9**

Description: **DeleteRecord** removes the current record from the record list by moving all subsequent records upward to fill the current record's slot. Any data blocks associated with the record are freed.

DeleteRecord does not update the BAM and VLIR file information on the disk. Call **CloseRecordFile** or **UpdateRecordFile** to update the file when done modifying.

Example:

See also: AppendRecord, InsertRecord

InsertRecord: (C64, C128) C286

Function: Adds an empty record before the current record in the index table, moving all subsequent records (including the current record) downward.

Parameters: none.

Uses:

curDrive	drive that disk is in. Set by call **SetDevice**
fileWritten†	if FALSE, assumes record just opened (or updated) and reads BAM/VBM into memory.
curRecord	Current record number
fileHeader	VLIR index table.

curType GEOS 64 v1.3 and later for detecting REU shadowing

curDirHead	BAM updated to reflect newly allocated block.
dir2Head†	(BAM for 1571 and 1581 drives only)
dir3Head†	(BAM for 1581 drive only)

†used internally by GEOS disk routines; applications generally don't use.

Returns: **x** error ($00 = no error).
MAX_VLIR_RECS
OUT_OF_REC0RDS

Alters:

curRecord	new record number
fileWritten†	set to TRUE to indicate the file has been altered since last updated.
fileHeader	buffer contains VLIR index table.
usedRecords	number of records in file that are currently in use.

Destroys: **a,y, r0L**

Preparatory routines[1]: OpenRecordFile

Description: **InsertRecord** attempts to insert an empty VLIR record in front of the current record in the index table 7of an open VLIR file, moving all subsequent records downward in the record list. The new record becomes the current record. A VLIR file can have a maximum of **MAX_VLIR_RECS** records. If adding a record will exceed this value, an **OUT_OF_REC0RDS** error is returned. In the index table, the new record is marked as used but empty ($00,$FF) [1].

Note: An application can create an empty VLIR file with **SaveFile.**

Note: GEOS up to 2.0 does not support a hierarchical file system, the "current directory" is actually the entire disk.

Note:[3] This Routine calls **GetDirHead** which loads in the BAM from disk. **PutDirHead** should be called prior to this routine if the BAM has been modified by Freeing or allocating blocks.

Example: **SaveRecord**

See also: ReadRecord, WriteRecord, CloseRecordFile, UpdateRecordFile

NextRecord:	(C64, C128)	C27A

Function: Makes the next record the current record.

Parameters: none

Uses: **fileHeader** index table checked to establish whether record exists.

Returns:

x error ($00 = no error).
  INV_RECORD
y Track of VLIR Chain. A value of $00 here means record is allocated but not in use (has no data blocks).
a new current record number.
r1L Track of VLIR Chain
r1H Sector of VLIR Chain

Alters: **curRecord** new record number

Destroys: nothing

Preparatory routines[1]**: OpenRecordFile**

Description: **NextRecord** makes the current record plus one the new current record. A subsequent call to **ReadRecord** or **WriteRecord** will operate with this record.

If the record does not exist, then **NextRecord** returns an then **NextRecord** returns an **INV_RECORD** (invalid record) error.

Example: **SaveRecord**

See also: PreviousRecord, PointRecord.

OpenRecordFile: (C64, C128) C274

Function: Open an existing VLIR file for access.

Parameters: **r0** FILENAME—pointer to null-terminated name of file (word).

Uses:
curDrive drive that disk is in. Set by call SetDevice
curType GEOS 64 v1.3 and later for detecting REU shadowing

Returns:
x error ($00 = no error).
STRUCT_MISMATCH
r1L track/Sector of VLIR Block
r1H

r5 pointer into **diskBlkBuf** to start of directory entry.

Alters:
fileHeader buffer contains VLIR index table.
usedRecords number of records in file that are currently in use.
curRecord new record number
fileWritten set to FALSE to indicate VLIR file has not been written to.
fileSize total number of disk blocks used in file (includes index block, GEOS file header, and all records).
dirEntryBuf directory entry of VLIR file.

Destroys: **a,y,r1,r4-r6**

Preparatory routines: SetDevice, OpenDisk

Description: Before accessing the data in a VLIR file, an application must call **OpenRecordFile. OpenRecordFile** searches the current directory for FILENAME and, if it finds it, loads the index table into **fileHeader. OpenRecordFile** initializes the GEOS VLIR variables (both local and global) to allow other VLIR routines such as **WriteRecord** and **ReadRecord** to access
the file. Only one VLIR file may be open at a time. A previously opened VLIR file should be closed before opening another.

If an application passes a FILENAME of a non-VLIR file, **OpenRecordFile** will return a **STRUCT_MISMATCH** error.

Note: An application can create an empty VLIR file with **SaveFile.**

Note: GEOS up to 2.0 does not support a hierarchical file system, the "current directory" is actually the entire disk.

Note:[3] This Routine calls **GetDirHead** which loads in the BAM from disk. **PutDirHead** should be called prior to this routine if the BAM has been modified by Freeing or allocating blocks.

Example: **SaveRecord**

See also: ReadRecord, WriteRecord, CloseRecordFile, UpdateRecordFile

PointRecord: (C64, C128) C280

Function: Make a particular record the current record.

Parameters: **a** RECORD — record number to make current (byte).

Uses:

fileHeader index table checked to establish whether record exists.
usedRecords Number of Currently Used Records in the VLIR file

Returns:

x error ($00 = no error).
y Track of VLIR Chain. A value of $00 here means record is allocated but not in use (has no data blocks).
a new current record number.
r1L Track of VLIR Chain
r1H Sector of VLIR Chain

Alters: **curRecord** new record number

Destroys: nothing

Preparatory routines[1]**: OpenRecordFile**

Description: **PointRecord** makes RECORD the current record so that a subsequent call to **ReadRecord** or **WriteRecord** will operate with RECORD. VLIR records are numbered zero through MAX_VLIR_RECS-1.

If the record does not exist (you pass a record number that is larger than the number of currently used records), then **PointRecord** returns an **INV_RECORD** (invalid record) error.

Example: **SaveRecord**

See also: NextRecord, PreviousRecord.

PreviousRecord: (C64, C128) C27D

Function: Makes the previous record the current record.

Parameters: none

Uses: **fileHeader** index table checked to establish whether record exists.

Returns:
- **x** error ($00 = no error).
  - **INV_RECORD**
- **y** Track of VLIR Chain. A value of $00 here means record is allocated but not in use (has no data blocks).
- **a** new current record number.
- **r1L** Track of VLIR Chain
- **r1H** Sector of VLIR Chain

Alters: **curRecord** new record number

Destroys: nothing

Preparatory routines[1]**: OpenRecordFile**

Description: **PreviousRecord** makes the current record minus one the new current record. A subsequent call to **ReadRecord** or **WriteRecord** will operate with this record.

If the record does not exist, then **PreviousRecord** returns an **INV_RECORD** (invalid record) error.

Example: **SaveRecord**

See also: NextRecord, PointRecord.

ReadRecord: (C64, C128) **C28C**

Function: Read in the current VLIR record.

Parameters: **r7** BUFFER — pointer to start buffer where data will be read into (word).
r2 BUFSIZE — size of buffer: Commodore version can read up to 32,258 bytes (127 Commodore blocks); Apple version can read up to the maximum two-byte number that can be passed in r2: 65,535 ($FFFF) bytes (word).

Uses:

curDrive	drive that disk is in. Set by call **SetDevice**
curRecord	Current record number
fileHeader	VLIR index table. Table holds Track / Sector of first block of each record.
curType	GEOS 64 v1.3 and later for detecting REU shadowing

Returns:

x error ($00 = no error).

a $00 = empty record, no data read.
$ff = record contained data.

r7 pointer to last byte read into **BUFFER** plus one if not an empty record, otherwise unchanged.

rl if **BFR_OVERFLOW** error returned, contains the track/sector of the block that, had it been copied from **diskBlkBuf** to the application's buffer space, would have exceeded the size of **BUFFER**. The process of copying any extra data from **diskBlkBuf** to the end of **BUFFER** is left to the application. The data starts at **diskBlkBuf**+2. If no error, then **rl** is destroyed

Alters: **fileTrScTab** As the chain blocks in the record is followed, the track/sector pointer of each block is added to the file track/sector table. The track and sector of the first block in the record is added at fiieTrScTab+2 and **fileTrScTab**+3. Refer to **ReadFile** for more information.

Destroys: y,(**r1**),**r2-r4** (see above for **r1**)

Preparatory routines[1]**: OpenRecordFile**

Description: **ReadRecord** reads the current record into memory at BUFFER. If the record contains more than BUFSIZE bytes of data, then a **BFR_OVERFLOW** error is returned.

ReadRecord calls **ReadFile** to load the chain of blocks into memory.

Example:

See also: WriteRecord, ReadFile.

UpdateRecordFile: (C64, C128) C295

Function: Update the disk copy of the VLIR index table, BAM and other VLIR information such as the file's time/date-stamp. This update only takes place if the file has changed since opened or last updated.

Parameters: none.

Uses:

curDrive	drive that disk is in. Set by call **SetDevice**
fileWritten	if FALSE, assumes record just opened (or updated) and
fileHeader	VLIR index table. Table holds Track / Sector of first block of each record.
fileSize	total number of disk blocks used in file (includes index block, GEOS file header, and all records).
dirEntryBuf	directory entry of VLIR file.
year, month, day, hours, minutes	for date-stamping file.
curType	GEOS 64 v1.3 and later for detecting REU shadowing
curDirHead	BAM updated to reflect newly allocated block.
dir2Head†	(BAM for 1571 and 1581 drives only)
dir3Head†	(BAM for 1581 drive only)

†used internally by GEOS disk routines; applications generally don't use.

Returns: x error ($00 = no error).

Alters: **fileWritten** set to FALSE to indicate that file hasn't been altered since last updated

Destroys: a, y, **r1**, **r4**, **r5**

Description: **UpdateRecordFile** checks the **fileWritten** flag. If the flag is **TRUE**, which indicates the file has been altered since it was last updated, **UpdateRecordFile** writes the various tables kept in memory out to disk (e.g., index table, BAM) and time/date-stamps the directory entry. If the **fileWritten** flag is FALSE, it does nothing.

UpdateRecordFile writes out the index block, adds the time/date-stamp and **fileSize** information to the directory entry, and writes out the new BAM with a call to **PutDirHead**.

Because Commodore GEOS stores the BAM in global memory, the application must be careful not to corrupt it before the VLIR file is updated. If the **fileWritten** flag is **TRUE** and the BAM is reread from disk, the old copy (on disk) will overwrite the current copy in memory. In the normal use of VLIR disk routines, where a file is opened, altered, then closed before any other disk routines are executed, no conflicts will arise.

Example:

See also: CloseRecordFile, OpenRecordFile.

WriteRecord: (C64, C128) C28F

Function: Write data to the current VLER record.

Parameters: none.

Uses:

curDrive	drive that disk is in. Set by call **SetDevice**
fileWritten	if FALSE, assumes record just opened (or updated) and
curRecord	Current record number
fileHeader	VLIR index table.
curType	GEOS 64 v1.3 and later for detecting REU shadowing
curDirHead	BAM updated to reflect newly allocated block.
dir2Head†	(BAM for 1571 and 1581 drives only)
dir3Head†	(BAM for 1581 drive only)

†*used internally by GEOS disk routines; applications generally don't use.*

Returns: x error ($00 = no error).

Alters:

fileWritten	set to FALSE to indicate that file hasn't been altered since last updated
fileHeader	index table adjusted to point to new chain of blocks for current record.
fileSize	adjusted to reflect new size of file.
fileTrScTab	Contains track/sector table for record as returned from **BlkAlloc**. The track and sector of the first block in the record is at **fileTrScTab**+0 and **fileTrScTab**+1 The end of the table is marked with a track value of $00.
curDirHead	BAM updated to reflect newly freed and allocated blocks.
dir2Head†	(BAM for 1571 and 1581 drives only)
dir3Head†	(BAM for 1581 drive only)

Destroys: a, y, **r1**, **r4**, **r5**

Description: **WriteRecord** writes data to the current record All blocks previously associated with the record are freed. **BlkAlloc** is then used to allocate enough new blocks to hold BYTES amount of data The data is then written to the chain of sectors by calling **WriteFile**. The **fileSize** variable is updated to reflect the new size of the file.

WriteRecord does not write the BAM and internal VLIR file information to disk. Call **CloseRecordFile** or **UpdateRecordFile** when done to update the disk with this information.

Note: **WriteRecord** correctly handles the case where the number of bytes to write (BYTES, R2) is zero. The record is freed and marked as allocated but not in use.

Example:

See also: ReadRecord, WriteFile.

disk DRIVER

AddDirBlock	9039	**Needs Documenting...**	
CallDrvRoutine:	9042	**Needs Documenting...**	
CheckDrvStatus:	9045	**Needs Documenting...**	
Get**diskBlkBuf**	903C	**Needs Documenting...**	
Put**diskBlkBuf**	903F	**Needs Documenting...**	
JmpIndX:	9D80	Jump Table **Needs Documenting...**	

AddDirBlock:	(C64, C128)	**9042**

Function: Call Directly into Disk Driver

Parameters:

Uses:

Returns:

Alters:

Destroys:

Description:

Note: **Needs Documenting...**

Example:

See also:

CallDrvRoutine: (C64, C128) 9042

Function: Call Directly into Disk Drive

Parameters:

Uses:

Returns:

Alters:

Destroys:

Description:

Note: **Needs Documenting...**

Example:

See also:

GetDiskBlkBuf: (C64, C128) 9045

Function: Call Directly into Disk Driver

Parameters:

Uses:

Returns:

Alters:

Destroys:

Description:

Note: **Needs Documenting...**

Example:

See also:

PutDiskBlkBuf: (C64, C128) **9045**

Function: Call Directly into Disk Driver

Parameters:

Uses:

Returns:

Alters:

Destroys:

Description:

Note: **Needs Documenting...**

Example:

See also:

CheckDrvStatus: (C64, C128) 9045

Function: Call Directly into Disk Driver

Parameters:

Uses:

Returns:

Alters:

Destroys:

Description:

Note: **Needs Documenting...**

Example:

See also:

JmpIndX:	(C64, C128)	**9D80**

Function: Jump Table

Parameters:

Uses:

Returns:

Alters:

Destroys:

Description:

Note: **Needs Documenting...**

Example:

See also:

graphics

BitmapClip: (C64, C128) C2AA

Function: Place a rectangular subset of a compacted bitmap on the screen.

Parameters:

r0	DATA	— pointer to the compacted bitmap data (word).
r1L	XPOS	— x card coordinate: pixel position /8 (byte).
r1H	Y	— y-coordinate (byte).
r2L	W_WIDTH	— width in cards: pixel width/8 (byte).
r2H	W_HEIGHT	— height in pixels (byte).
r11L	DX1	— delta-x1: offset of left edge of clipping window in cards from left edge of full bitmap (byte).
r11H	DX2	— delta-x2: offset of right edge of clipping window in cards from right edge of full bitmap (byte).
r12	DY1	— delta-y 1: offset of top edge of clipping window in pixels from top edge of full bitmap (word).

*where the upper-left corner of the clipped bitmap is placed at (XPOS*8,Y). The lower-right corner is at (XPOS*8+W_WIDTH*8,Y+W_HEIGHT).*

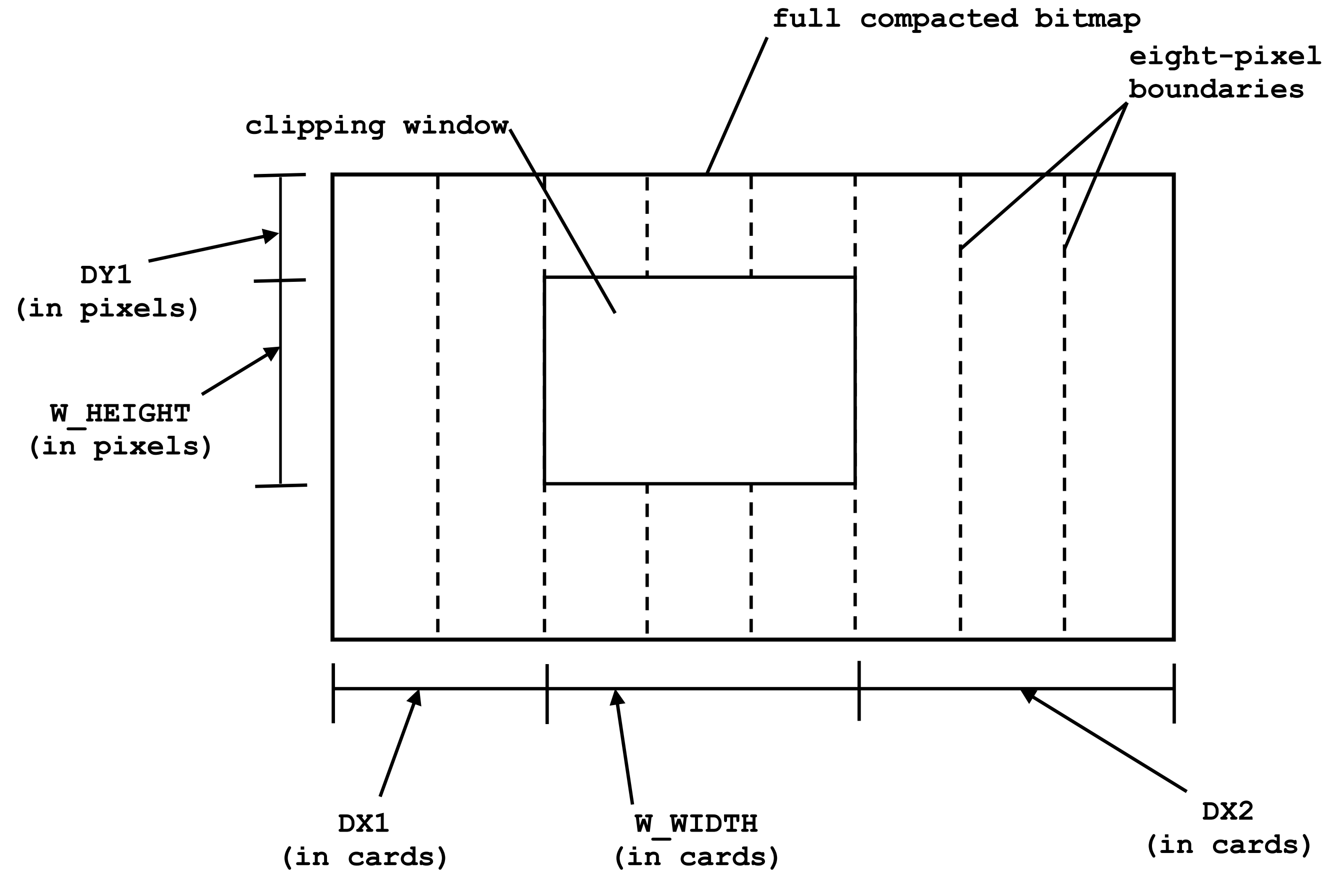
Uses: **dispBufferOn:**
bit 7 — write to foreground screen if set.
bit 6 — write to background screen if set.

Returns: Nothing

Destroys: a, x, y, **r0-r12**

Description: **BitmapClip** uncompacts a rectangular area of a full bitmap, clipping (ignoring) any data that exists outside of the desired area. The rectangular subset is called the *clipping window*.

The following diagram illustrates the eight **BitmapClip** parameters:



No checks are made to determine if the data, dimensions, or positions are valid. Be careful to pass accurate values. Do not pass a value of $00 for either the *W_WIDTH* or *W_HEIGHT* parameters, and pay special attention to the fact that *XPOS*, *W_WIDTH*, *DX1*, and *DX2* are specified in cards (groups of eight pixels horizontally), not in individual pixels.

NOTE: It may be helpful to think of *DY1* as the number of scanlines in the bitmap to skip initially, to think of *W_HEIGHT* as the number of scanlines to display, to think of *DX1* as the number of cards to skip at the beginning of each scanline, to think of *W_WIDTH* as the number of cards to display, and to think of *DX1* as the number of bytes to skip at the end of each scanline.

C128: Under GEOS 128, OR'ing **DOUBLE_B** into the *XPOS* and *W_WIDTH* parameters automatically doubles the x-position and the width of the bitmap (respectively) when running in 80-column mode.

BitmapClip in the first release of GEOS 128 does not call **TempHideMouse** to disable the sprites and does not properly double the width when drawing to the 80-column screen. On Kernal's where the release byte is greater than $01, these problems have been fixed.

Example: DisplayImage

See also: BitmapUp , BitOtherClip

BitmapUp , i_BitmapUp: (C64, C128) C142, C1AB

Function: Place a compacted bitmap onto the screen.

Parameters: Normal:

```
r0     DATA    — pointer to the compacted bitmap data (word).
r1L    XPOS    — x card coordinate: pixel position /8 (byte).
r1H    Y       — y-coordinate (byte).
r2L    WIDTH   — width in cards: pixel width/8 (byte).
r2H    HEIGHT  — height in pixels (byte).
```

Inline:

data appears immediately after the jsr **i_BitmapUp**

```
.word  DATA      pointer to the compacted bitmap data.
.byte  XPOS      x card position: pixel position /8.
.byte  Y         y-coordinate.
.byte  WIDTH     width in cards: pixel width/8.
.byte  HEIGHT    height in pixels.
```

*where the upper-left corner of the bitmap is placed at (XPOS*8,Y). The lower-right corner is at (XPOS*8+WIDTH*8,Y+HEIGHT).*

Uses: **dispBufferOn:**
bit 7 — write to foreground screen if set.
bit 6 — write to background screen if set.

Returns: Nothing

Destroys: **a, x, y, r0-r9L**

Description: **BitmapUp** uncompacts a GEOS compacted bitmap according to the width and height information and places it at the specified screen position. No checks are made to determine if the data, dimensions, or positions are valid, and bitmaps which exceed the screen edge will not be clipped. Be careful to pass accurate values. Do not pass a $00 for the WIDTH or the HEIGHT parameter, and pay special attention to the fact that both the x-position and the width are specified in cards (groups of eight pixels horizontally), not in pixels.

128: Under GEOS 128, OR'ing DOUBLE_B into the XPOS and WIDTH parameters will automatically double the x-position and the width (respectively) in 80-column mode. The first release of GEOS 128 did not properly remove the sprites before placing the bitmap on the screen. The easiest way to correct for this is to always precede a call to **BitmapUp** with a call to **TempHideMouse**. The redundant call to **TempHideMouse** when running under later releases is minimal compared to the number of cycles it takes to decompact and draw the bitmap.

```
jsr TempHideMouse ; correct for bug in release 1 of GEOS 128
jsr BitmapUp      ; then put up the bitmap
```

Example: **ShowBitmap**

See also: **BitmapClip, BitOtherClip**

DrawLine: (C64, C128) C130

Function: Draw, clear, or recover a line defined by two arbitrary endpoints.

Parameters:
r3 X1 — x-coordinate of pixel (word).
r11L Y1 — y-coordinate of pixel (byte).
r4 X2 — x-coordinate of second endpoint (word).
r11H Y2 — y-coordinate of second pixel (byte).
st MODE:

N	C	Operation
1	x	recover pixel from background screen to foreground
0	1	set pixel using **dispBufferOn**.
0	0	clear pixel using **dispBufferOn**.

where (X1,Y1) and (X2,Y2) are the two endpoints of the line.

Uses: if n is set (drawing, not recovering):
when setting or clearing pixels:
dispBufferOn:
bit 7 — write to foreground screen if set.
bit 6 — write to background screen if set.

Destroys: **a, x, y, r3-13**

Description: **DrawLine** will set, clear, or recover the pixels which comprise the line between two arbitrary endpoints. Setting a pixel sets its bit value to one, clearing a pixel sets its bit value to zero, and recovering a pixel copies the bit value from the background buffer to foreground screen.

DrawLine uses the Bresenham DDA (Digital Differential Analyzer) algorithm to determine the proper points to draw. The line will be drawn correctly regardless of which endpoint is used for (X1,Y1) and which is used for (X2,Y2). In fact, the line is reversible: the same line will be drawn even if the endpoints are swapped.

The carry (c) flag and sign (n) flag in the processor status register (s) are used to pass information to **DrawLine.** The following tricks can be used to set or clear these flags appropriately:

• Use sec and clc to set or clear the carry (c) flag.
• Use lda #[-1 to set the sign (n) flag.
• Use lda #0 to clear the sign (n) flag.

Note: Calculates each pixel position on the line and calls DrawPoint repeatedly.

C128: Under GEOS 128, OR'ing DOUBLE W into the XI andX2 parameters will automatically double the x-position in SO-column mode. OR'ing in ADD1_W will automatically add 1 to a doubled x-position- (Refer to "GEOS 128 X-position and Bitmap Doubling" in "Graphics Routines" for more information.)

Example:

See also: **TestPoint, DrawLine.**

DrawPoint: (C64, C128) C133

Function: Set, clear, or recover a single screen point (pixel).

Parameters:
r3 X1 — x-coordinate of pixel (word).
r11L Y1 — y-coordinate of pixel (byte).
st MODE:

N	C	Operation
1	x	recover pixel from background screen to foreground
0	1	set pixel using **dispBufferOn**.
0	0	clear pixel using **dispBufferOn**.

where (X1,Y1) is the coordinate of the point.

Uses: when setting or clearing pixels:
dispBufferOn:
bit 7 — write to foreground screen if set.
bit 6 — write to background screen if set.

Destroys: **a, x, y, r5-r6**

Description: **DrawPoint** will set, clear, or recover a single pixel. Setting a pixel sets its bit value to one, clearing a pixel sets its bit value to zero, and recovering a pixel copies the bit value from the background buffer to foreground screen.

The carry (c) flag and sign (n) flag in the processor status register (s) are used to pass information to **DrawPoint.** The following tricks can be used to set or clear these flags appropriately:

- Use sec and clc to set or clear the carry (c) flag.
- Use lda #[-1 to set the sign (n) flag.
- Use lda #0 to clear the sign (n) flag.

C128: Under GEOS 128, OR'ing DOUBLE W into the XI andX2 parameters will automatically double the x-position in SO-column mode. OR'ing in ADD1_W will automatically add 1 to a doubled x-position- (Refer to "GEOS 128 X-position and Bitmap Doubling" in "Graphics Routines" for more information.)

Example:

See also: **TestPoint, DrawLine.**

FrameRectangle, i_FrameRectangle: (C64, C128) C127, C1A2

Function: Draw a rectangular frame (one-pixel thickness).

Parameters: Normal:

a eight-bit line pattern.
r3 X1 — x-coordinate of upper-left (word).
r2L Y1 — y-coordinate of upper-left (byte).
r4 X2 — x-coordinate of lower-right (word).
r2H Y2 — y-coordinate of lower-right (byte).

where (X1,Yl) is the upper-left corner of the frame and (X2,Y2) is the lower-right corner.

Inline:

data appears immediately after the jsr **i_FrameRectangle**

```
.byte Yl          y-coordinate of upper-left
.byte Y2          y-coordinate of lower-right.
.word X1          x-coordinate of upper-left
.word X2          x-coordinate of lower-right.
.byte PATTERN     eight-bit line pattern.
```

Uses: **dispBufferOn:**
bit 7 — write to foreground screen if set.
bit 6 — write to background screen if set.

Destroys: **a, x, y, r5-r9,r11**

Description: **FrameRectangle** draws a one-pixel rectangular frame on the screen as determined by the coordinates of the upper-left and lower-right corners. The horizontal and vertical lines which comprise the frame are drawn with the specified line pattern.

FrameRectangle operates by calling **HorizontalLine** and **VerticalLine** with the desired line-pattern. As with these two routines, the line pattern is drawn as if aligned on an eight-pixel boundary. The values of the corner pixels will be determined by the vertical sides because they are drawn after the horizontal sides.

Because all GEOS coordinates are inclusive, framing a filled rectangle requires either calling **FrameRectangle** after calling **Rectangle** (and thereby overwriting the perimeter of the filled area) or calling FrameRectangle with (*X1*-1,*Y7*-1) and (*X2*+1,*Y2*+1) as the corner points.

Under GEOS 128, OR'ing **DOUBLE_W** into the *X1* and *X2* parameters will automatically double the x-position in 80-column mode. OR'ing in **ADD1_W** will automatically add 1 to a doubled x-position. (Refer to "GEOS 128 X-position and Bitmap Doubling" in Chapter "Graphics Routines" for more information.)

Example:

See also: **Rectangle, ImprintRectangle, RecoverRectangle, InvertRectangle.**

GetScanLine: (C64, C128) C13C

Function: Calculate the memory address of a particular screen line.

Parameters: **x** Y — y-coordinate of line.

Uses: **dispBufferOn**:
bit 7 — write to foreground screen if set.
bit 6 — write to background screen if set.

Returns: x unchanged.
addresses in r5 and r6 based on **dispBufferOn** status:

bit 7	bit 6	returns
1	1	r5 = foreground; r6 = background
0	1	r5, r6 = background
1	0	r5, r6 = foreground
0	0	*error: r5,r6 = address of screen center*

Destroys: **a**

Description: **GetScanLine** calculates the address of the first byte of a particular screen line. The routine always places addresses in both r5 and r6, depending on the value in **dispBufferOn**. This allows an application to automatically manage both foreground screen and background buffer writes according to the bits set in **dispBufferOn** by merely doing any screen stores twice, indirectly off both r5 and r6 as in:

```
/ Note: this code is C64 specific (see notes below for 128)
      ldy    xpos          ; byte index into current line
      lda    grByte        ; graphics byte to store
      sta    (r5),y        ; store using both indexes
      sta    (r6),y
```

128: When GEOS 128 is operating in 80-column mode, all foreground writes are sent through the VDC chip to its local RAM. In this case, the address of the foreground screen byte is actually an index into VDC RAM for the particular scanline. For background writes, the address of the background screen byte is an absolute address in main memory (be aware, though, that the background screen is broken into two parts and is not a contiguous chunk of memory).

In 40-column mode, **GetScanLine** operates as it does under GEOS 64.

Example:

See also:

graphics

NormalizeX:	(C128)	**C2E0**

Function: Adjust an x-coordinate to compensate for the higher-resolution 80-column mode.

Parameters: **x** GEOSREG — zero-page address of word-length GEOS register which contains the word-length X-coordinate to adjust..

Returns: x unchanged.
register passed as *GEOSREG* parameter contains the adjusted x-coordinate.

Destroys: **a**

Description: **NormalizeX** is used by nearly every GEOS 128 routine that writes to the screen. It adjusts an x-coordinate (two's complement signed word) based on the graphics mode (40- or 80-column) and the status of the special bits in the coordinate. **NormalizeX** allows an application to run in both 40- and 80-column modes with a minimum of programming effort If the proper bits in a 40-column coordinate is set, **NormalizeX** will automatically double the value when in 80-column mode.

Since GEOS graphics operations automatically call **NormalizeX** to adjust the coordinates, most applications will not need to call it directly. Bit 15 of the coordinate specifies doubling. Bit 13 adds one to a doubled coordinate (allowing odd-pixel addressing). Bit 14 is a pseudo-sign bit. Use the **DOUBLE_W** and **ADD1_W** constants to access these bits. If the coordinate might be negative, the **DOUBLE_W** and **ADD1_W** constants should be exclusive-or'ed into the x-position so that the sign is preserved. However, if the coordinate is guaranteed to be a positive number, the constants may simply be or'ed in.

The *GEOSREG* parameter is an actual zero-page address. Usually this will be a GEOS register (**r0-r15**) or an application's register (**a0-a9**). If, for example, an application had a value in **r9** which it wanted normalized, it would first exclusive-or in the special bits, then call **NormalizeX** in the following manner:

```
ldx     #r9                     ; load x with addr of r9
jsr     NormalizeX              ; normalize the val in r9
```

The following breakdown of the word-length x-coordinate illustrates how the special bits affect the adjustment process.

b15	b14	b13	x-pixel coordinate (b0-b12)

b0-12 x-coordinate in pixels (two's comp. number).
b13 add one to doubled x-coordinate (flag).
b14 x-coordinate sign-extension from b12 (pseudo sign-bit).
b15 double x-coordinate (flag).

If in 40-column mode, then the special bits are ignored and the x-coordinate is returned to its original state (the state it was in before any special constants were exclusive-or'ed in).

If in 80-column mode, then the following applies:

b15	b14	b13	Effect
0	0	n	x value changed (normal positive).
1	1	n	x value changed (normal negative).
0	1	n	x=x*2-n (double negative)
1	0	n	x=x*2+n (double positive)

Note: For more information, refer to "GEOS 128 X-position and Bitmap Doubling" in Chapter "Graphics Routines"

Example:

See also:

Rectangle, i_Rectangle: (C64, C128) C124, C19F

Function: Draw a rectangle in the current fill pattern.

Parameters: Normal:

r3 X1 — x-coordinate of upper-left (word).
r2L Y1 — y-coordinate of upper-left (byte).
r4 X2 — x-coordinate of lower-right (word).
r2H Y2 — y-coordinate of lower-right (byte).

where (X1,Y1) is the upper-left corner of the frame and (X2,Y2) is the lower-right corner.

Inline:
data appears immediately after the jsr **i_FrameRectangle**

```
.byte Y1         y-coordinate of upper-left
.byte Y2         y-coordinate of lower-right.
.word X1         x-coordinate of upper-left
.word X2         x-coordinate of lower-right.
```

Uses: **dispBufferOn:**
bit 7 — write to foreground screen if set.
bit 6 — write to background screen if set.

Destroys: a, x, y, **r5-r8**

Description: **Rectangle** draws a filled rectangle on the screen as determined by the coordinates of the upper-left and lower-right corners. The rectangle is filled with the current 8x8 (card-sized) fill pattern.

The 8x8 pattern within the rectangle is drawn as if it were aligned to a card boundary: that is, the bit-pattern is synchronized with (0,0), and, since the patterns are 8x8, they are aligned with every eighth pixel thereafter. This allows the patterns in adjacent or overlapping rectangles to line-up regardless of the actual pixel positions.

Rectangle operates by calling HorizontalLine in a loop, changing the bit-pattern byte after every line based on the current 8x8 fill pattern.

Because all GEOS coordinates are inclusive, framing a filled rectangle requires either calling **FrameRectangle** after calling **Rectangle** (and thereby overwriting the perimeter of the filled area) or calling **FrameRectangle** with (*X1*-1,*Y1*-1) and (*X2*+1,*Y2*+1) as the corner points.

Note: Under GEOS 128, OR'ing **DOUBLE_W** into the *X1* and *X2* parameters will automatically double the x-position in 80-column mode. OR'ing in **ADD1_W** will automatically add 1 to a doubled x-position. (Refer to "GEOS 128 X-position and Bitmap Doubling" in Chapter "Graphics Routines" for more information.)

Example:

See also: **FrameRectangle, ImprintRectangle, RecoverRectangle, InvertRectangle.**

SetNewMode: (C128) $C2DD

Function: Set 128 mode to 40 or 80 column mode.

Uses: **graphMode** GRMODE — new graphics mode to change to:

40-Column:	**GR_40**
80-Column:	**GR_80**

Returns: nothing.

Destroys: **a, x, y, r0-r15**

Description: **SetNewMode** the Operation mode of the Commodore 128.

40-column mode (**graphMode** == GR_40)

1: 8510 clock speed is slowed down to 1Mhz because VIC chip cannot operate at 2Mhz.
2: **rightMargin** is set to 319.
3: **UseSystemFont** is called to begin using the 40-column font.
4: 40-column VIC screen is enabled.
5: 80-column VDC is set to black on black, effectively disabling it.

80-column mode (**graphMode** == GR_80)

1: 8510 clock speed is raised to 2Mhz.
2: **rightMargin** is set to 639.
3: **UseSystemFont** is called to begin using the 80-column font.
4: 40-column VIC screen is disabled.
5: 80-column VDC screen is enabled.

Example: **Change Mode**

See also: **TestPoint, DrawLine, GetScanLine.**

text

SetPattern: (C64, C128) C139

Function: Set the current fill pattern.

Parameters: a GEOS system pattern number (must be between 0 and 31) (byte).

Returns: *nothing.*

Alters: **curPattern** Contains an address pointing to the eight-byte pattern.

Destroys: **a**

Description: **SetPattern** sets the current fill pattern. There are 34 system patterns (numbered 0-33) in GEOS; Unfortunately, **SetPattern** will only work correctly with patterns numbered 0-31. To access higher number patterns, call **SetPattern** with a value of 31 and add 8 to **curPattern** in order to access pattern 32, add 16 to access pattern 33, and so on.

Example:

See also: GetPattern.

text

TestPoint: (C64, C128) C13F

Function: Test and return the value of a single point (pixel).

Parameters:
r3 X1 — x-coordinate of pixel (word).
r11L Y1 — y-coordinate of pixel (byte).

where (X1,Y1) is the coordinate of the point to test.

Uses: **dispBufferOn:**
bit 7 — write to foreground screen if set.
bit 6 — write to background screen if set.
(If both bit 6 and bit 7 are set, then only the pixel in the background screen is tested).

Returns: **r3L, r11L** unchanged.

Destroys: **a, x, y, r5-r6**

Description: **TestPoint** will test a pixel in cither the foreground screen or the background buffer (or both simultaneously) and return the pixel's status by either setting or clearing the carry (x) flag accordingly. The jsr **TestPoint** is usually followed immediately by a bcc or bcs so that a set or clear pixel may be handled appropriately.

C128: Under GEOS 128, OR'ing **DOUBLE_W** into the X1 will automatically double the x-position in 80-column mode. OR'ing in **ADD1_W** will automatically add 1 to a doubled x-position. (Refer to "GEOS 128 Xposition and Bitmap Doubling" in Chapter "Graphics Routines" for more information.)

Example:

See also: DrawPoint.

VerticalLine: (C64, C128) C121

Function: Draw a vertical line with a repeating bit-pattern.

Parameters:
a eight-bit repeating pattern to use (not a GEOS pattern number).
r4 X1 — x-coordinate of line (word).
r3L Yl — y-coordinate of topmost endpoint (byte).
r3H Y2 — y-coordinate of bottommost endpoint (byte).

where (X1,Yl) and (X1,Y2) define the endpoints of the vertical line.

Uses: **dispBufferOn:**
bit 7 — write to foreground screen if set.
bit 6 — write to background screen if set

Returns: **r3L**, **r3H**, **r4** unchanged.

Destroys: **a**, **x**, **y**, **r5L-r8L**

Description: **VerticalLine** sets and clears pixels on a single vertical line according to the eight-bit repeating pattern. Wherever a 1-bit occurs in the pattern byte, a pixel is set, and wherever a O-bit occurs, a pixel is cleared.

Bits in the pattern byte are used top-to-bottom, where bit 7 is at the top. A bit pattern of %11110000 would create a vertical line like:

The pattern byte is always drawn as if aligned to a card boundary. If the endpoints of a line do not coincide with card boundaries, then bits are masked off the appropriate ends. The effect of this is that a pattern is always aligned to specific pixels, regardless of the endpoints, and that adjacent lines drawn in the same pattern align.

Note: To draw patterned vertical lines using the 8x8 GEOS patterns, draw rectangles of one-pixel width by calling the GEOS Rectangle routine with identical Xcoordinates.

Example:

See also: HorizontalLine.

icon/menu

DoIcons: (C64, C128) C15A

Function: Display and activate an icon table.

Parameters: **r0** ICONTABLE — pointer to the icon table to use.

Uses: **dispBufferOn:**
bit 7 — draw icons to foreground screen if set.
bit 6 — draw icons to background screen if set.

Destroys: **rO-rl5**, a, x, y

Description: **DoIcons** takes an icon, draws the enabled icons (those whose **OFF_I_PIC** word is non-zero) and instructs **MainLoop** to begin tracking the user's interaction with the icons. This routine is the only way to install icons. Every application should install at least one icon, even if only a dummy icon.

If **DoIcons** is called while another icon table is active, the new icons will take precedence. The old icons are not erased from the screen before the new ones are displayed.

DoIcons is a complex routine which affects a lot of system variables and tables. The following is an outline of its major actions:

1: All enabled icons in the table are drawn to the foreground screen and/or the background buffer based on the value in **dispBufferOn**.

2: **StartMouseMode** is called. If the **OFF_IC XMOUSE** word of the icon table header is non-zero, then **StartMouseMode** loads **mouseXPosition** and **mouseYposition** with the values in the **OFF_IC_XMOUSE** and the **OFF_IC_YMOUSE** parameters of the icon table header (see **StartMouseMode** for more information).

4: **faultData** is cleared to $00, indicating no faults.

5: If the **MOUSEON_BIT** of **mouseOn** is *clear*, then the **MENUON BIT** is forced to one. This is because GEOS assumes that it is in a power-up state and that mouse tracking should be fully enabled. If the **MOUSEON**_BIT bit is set, GEOS leaves the menu-scan alone, assuming that the current state of the **MENUON_BIT** is valid.

6: The **ICONSON_BIT** and **MOUSEON BIT** bits of **mouseOn** are set thereby enabling icon-scanning.

When an icon event handler is given control, **r0L** contains the number of the icon clicked on (beginning with zero) and **r0H** contains **TRUE** if the event is a double-click or FALSE if the event is a single click.
.

Example: **IconsUp**

See also: DoMenu.

DoMenu: (C64, C128) C151

Function: Display and activate a menu structure.

Parameters: **r0** MENU — pointer to the menu structure to display.
a POINTER_OVER — which menu item (numbered starting with zero) to center the pointer over.

Destroys: a, x, y, **r0-r13**

Description: **DoMenu** draws the main menu (the first menu in the menu structure) and instructs **MainLoop** to begin tracking the user's interaction with the menu. This routine is the only way to install a menu.

If **DoMenu** is called while another menu structure is active, the new menu will take precedence. The old menu is not erased from the screen before the new menu is displayed. If the new menu is smaller (or at a different position) than the old menu, parts of the old menu may be left on the screen. A typical way to avoid this is to erase the old menu with a call to **Rectangle**, passing the positions of the main menu rectangle and drawing in a white pattern. However, a more elegant solution involves calling **GotoFirstMenu**, which will erase any extant menus by recovering from the background buffer.

DoMenu is a complex routine which affects a lot of system variables and tables. The following is an outline of its major actions:

1: Menu level 0 (main menu) is drawn to the foreground screen.

2: **StartMouseMode** is called. mouseXPosition and mouseYposition are set so that the pointer is centered over the selection number passed in a. Under Apple GEOS, if the CallRoutine POINTER_OVER number in the accumulator has its high-bit set, then the mouse will not be repositioned Under GEOS 64 and GEOS 128, **DoMenu** always forces the mouse to a new position. If you do not want the mouse moved, surround the call to **DoMenu** with code to save and restore the mouse positions. The following code fragment will install menus without moving the mouse.

```
DoMenu2:
    php                     ; Save Processor Status Register
    sei                     ; disable interrupts around call
    PushW   mouseXPos       ; save mouse x
    PushB   mouseYPos       ; save mouse y
    lda     #0              ; dummy menu value
    jsr     DoMenu          ; install menus (mouse will move)
    PopB    mouseYPos       ; restore original mouse y
    PopW    mouseXPos       ; restore original mouse x
    plp                     ; Restore Interrupts to their saved state
    rts
```

3: SlowMouse is called. With a joystick this will kill all accumulated speed in the pointer, requiring the user to reaccelerate. With a proportional mouse, this will have no effect.

4: **faultData** is cleared to $00, indicating no faults.

5: If the MOUSEON_BIT of **mouseOn** is clear, then the ICONSON_BIT is forced to one. This is because GEOS assumes that it is in a power-up state and that mouse tracking should be fully enabled. If the MOUSEON_BIT bit is set, GEOS leaves the icon-scan alone, assuming that the ICONSON_BIT is valid.

6: The **MENUON_BIT** and **MOUSEON_BIT** bits of **mouseOn** are set, thereby enabling menu-scanning.

7: The mouse fault variables (**mouseTop**, **mouseBottom**, **mouseLeft**, and **mouseRight**) are set to the full screen dimensions.

Example:

See also: DoIcons, GotoFirstMenu, DoPreviousMenu, ReDoMenu.

DoPreviousMenu: (C64, C128) **C190**

Function: Retracts the current sub-menu and reactivates menus at the previous level.

Parameters: none:

Destroys: assume **rO-rl5**, a, x, y

Description: **DoPreviousMenu** is used by a menu event handler to instruct GEOS to back up one level of menus, erasing the current menu from the foreground screen and making the parent menu active when control is returned to **MainLoop. menuNumber** is decremented.

When using **DoPreviousMenu,** if the parent menu (the one which will be given control) is of type **UN_CONSTRAINED,** then the mouse must be manually repositioned over the parent menu. This can be done by loading **mouseXPosition** and **mouseYPosition** with values calculated from the menu structure. If the parent menu is of type **CONSTRAINED,** then the mouse is automatically positioned over the selection in the parent menu which led to the sub-menu..

Note: **DoPreviousMenu** may be called repeatedly to back up more than one level.

Do not call **DoPreviousMenu** when the menu is at level 0 (menuNumber = $00). The effects may be disastrous.

Example:

See also: DoMenu, GotoFirstMenu, ReDoMenu, RecoverMenu.

GotoFirstMenu:	(C64, C128)	**C1BD**

Function: Retracts the current sub-menu and reactivates menus at the previous level.

Parameters: none:

Destroys: assume **r0-r15**, a, x, y

Description: **GotoFirstMenu** is used by a menu event handler to instruct GEOS to back up to the main menu level, erasing the current menu and any parent menus (except the main menu) from the foreground screen, making the main menu active when control is returned to **MainLoop**. **menuNumber** is set to $00.

GotoFirstMenu can be called from a menu event routine at any menu level, including main menu level. It operates by checking for level zero and calling **DoPreviousMenu** in a loop.

Example:

See also: DoMenu, DoPreviousMenu, ReDoMenu, RecoverAllMenus.

RecoverAllMenus: (C64, C128) C157

Function: Removes all menus (including the main menu) from the foreground screen by recovering from the background buffer.

Parameters: none:

Destroys: assume **r0-r15**, a, x, y

Description: **RecoverAllMenus** is a very low-level menu routine which recovers the area obscured by the opened menus from the background buffer. Usually this routine is only called internally by the higher-level menu routines. It is of little use in most applications and is included in the jump table mainly for historical reasons.

RecoverAllMenus operates by loading the proper GEOS registers with the coordinates of the menu rectangles and calling the routine whose address is in **recoverVector** (normally RecoverRectangle) repeatedly.

Example:

See also: DoPreviousMenu, ReDoMenu, GotoFirstMenu, RecoverMenu.

RecoverMenu: (C64, C128) **C154**

Function: Removes the current menu from the foreground screen by recovering from the background buffer.

Parameters: none.

Destroys: assume **r0-r15**, a, x, y

Description: **RecoverMenu** is a very low-level menu routine which recovers the rectangular area obscured by the current menu. Usually this routine is only called internally by the higher-level menu routines such as **DoPreviousMenu.** It is of little use in most applications and is included in the jump table mainly for historical reasons.

RecoverMenu operates by loading the proper GEOS registers with the coordinates of the current menu's rectangle and calling the routine pointed to by recoverVector (normally RecoverRectangle).

Example:

See also: DoMenu.

ReDoMenu: (C64, C128) C193

Function: Reactivate menus at the current level.

Parameters: none.

Destroys: **assume r0-r15**, a, x, y

Description: **ReDoMenu** is used by the application's menu event handler to instruct GEOS to leave all menus (including the current menu) open when control is returned to **MainLoop**. **menuNumber** is unchanged. Keeping the current menu open allows another selection to be made immediately.

ReDoMenu will redraw the current menu. If menu event routine changes the text in the menu (adding a selection asterisk, for example), a call to **ReDoMenu** will redraw the menu with the new text while leaving the menu open for another selection.

Example:

See also: DoMenu, GotoFirstMenu, DoPreviousMenu.

input driver

InitMouse	$FE80	Initialize input device.	
SetMouse	$FE89	Reset input device scanning circuitry.	
SlowMouse	$FE83	Reset mouse velocity variables.	
UpdateMouse	$FE86	Update mouse variables from input device.	

InitMouse: (c64,C128) FE80

Function: Initialize the input device.

Parameters: none.

Returns: nothing.

Alters:

mouseXPos	**initialized (typically 8).**
mouseYPos	**initialized (typically 8).**
mouseData	**initialized (typically reflects a released button).**
pressFlag	**initialized (typically set to $00).**

Destroys: **assume a,x,y,r0-r15**

Description: GEOS calls **InitMouse** after first loading an input driver. The input driver is expected to initialize itself and begin tracking the input device. An application should never need to call InitMouse.

Example:

See also: SlowMouse, UpdateMouse, SetMouse, StartMouseMode, MouseUp.

SetMouse: (C128) FE89

Function: Input device scan reset.

Parameters: none.

Returns: nothing.

Destroys: **assume a,x,y,r0-r15**

Description: GEOS 128 calls **SetMouse** during Interrupt Level, immediately after the keyboard is scanned for a new key, to reset the pot (potentiometer) scanning lines so that they will recharge with the new value of. It is primarily of use with the Commodore 1351 mouse, which requires having the pot lines reset regularly. Other input drivers will have a **SetMouse** routine that merely performs an rts. An application should never need to call **SetMouse.**

Example:

See also: SlowMouse, UpdateMouse, Initmouse.

SlowMouse: (c64,C128) FE83

Function: Kills any accumulated speed in a non-proportional input device.

Parameters: none.

Returns: nothing.

Alters: internal input-driver speed variables, if any.

Destroys: **assume a,x,y,r0-r15**

Description: Input drivers for non-proportional input devices, such as a joystick, will often internally associate a speed and velocity with movement. This way the pointer can speed up when the user is trying to move large distances. **SlowMouse** will tell the input driver to kill any accumulated speed, effectively stopping the pointer at a specific location and forcing it to regain momentum. Depending on the input driver, **SlowMouse** may or may not have an effect on the pointer's movement The standard mouse driver, for example, simply performs an rts but some other input driver may actually copy the value in **minMouseSpeed** to its own internal speed variable.

GEOS calls **SlowMouse** when it drops menus down. A driver that has velocity variables should adjust the current speed so that the pointer does not immediately jump off the menu. An application may want to call **SlowMouse** when the user is required to make precise movements.

Example:

See also: UpdateMouse, Initmouse, SetMouse.

UpdateMouse: (c64,C128) FE86

Function: Update the mouse variables based on any changes in the state of the input device.

Parameters: none.

Returns: nothing.

Alters:

mouseXPos mouse x-position.
mouseYPos mouse y-position.
mouseData state of mouse button: high bit set if button is released; clear if pressed.
pressFlag MOUSE_BIT and INPUT_BIT set appropriately.
inputData depends on device

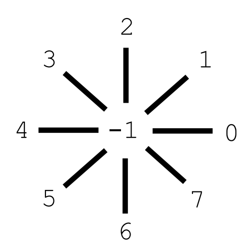
Destroys: **assume a,x,y,r0-r15**

Description: GEOS calls **UpdateMouse** at Interrupt Level to update the GEOS mouse variables with the actual state of the input device. An application should never need to call **UpdateMouse.**

A typical input driver's **UpdateMouse** routine will scan the device hardware and update **MouseXPos** and **MouseYPos** with new positions if the coordinates have changed It will also update **mouseData** with the current state of the input button (high-bit set if released; cleared if pressed) and set MOUSE_BIT in pressFlag if the button state has changed since the last call to **UpdateMouse.**

The four byte inputData field, which was originally for device-dependent information, has adopted the following standard offsets:

inputData+0 (byte) 8-position device direction (joystick direction; mouse drivers convert a moving mouse to an appropriate direction):



inputData+1 (byte) current speed (Commodore joystick drivers only).

Standard GEOS input drivers should set the **INPUT_BIT** of **pressFlag** if **inputData**+O has changed since the last time **UpdateMouse** was called. Because most GEOS applications leave **inputVector** set to its default $0000 value, setting this bit will usually have no effect.

Example:

See also: SlowMouse, Initmouse, SetMouse.

internal

internal

BootGeos: (c64,C128) C000

Function: Restart GEOS from a non-GEOS application.

Parameters: none.

Returns: Does not return.

Destroys: n/a

Description: **BootGeos** provides a method for an non-GEOS to run in the GEOS environment—starting up from the deskTop and returning to GEOS when done. The non-GEOS application need only preserve the area of memory between **BootGeos** ($C000) and **BootGeos**+$7f ($CO7f). The rest of the GEOS Kernal may be overwritten. To reboot GEOS, simply jmp **BootGeos**, which completely reloads the operating system (either from disk in a "boot11 procedure or from a RAM-expansion unit in an "rboot11 procedure) and returns to the GEOS deskTop.

A program can check to see if it was loaded by GEOS by checking the memory starting at $c006 (bootName) for the ASCII (not CBMASCII) string "GEOSBOOT". If loaded by GEOS, the program can check bit 5 of $cO12 (**sysFlgCopy**): if this bit is clear, ask the user to insert their GEOS boot disk before continuing, otherwise a boot disk is not needed because GEOS will rboot from the RAM expansion unit. To actually return to GEOS, set **CPU_DATA** to $37 (KRNL BAS 10 IN) on a Commodore 64 and set config to $40 (CKRNL_BAS_IO_IN) on a Commodore 128, then jump to **BootGeos**

Example: RoadTrip

See also: FirstInit, StartAppl, GetFile, EnterDeskTop.

internal

FirstInit: (c64,C128) **C271**

Function: Simulates portions of the GEOS coldstart procedure without actually rebooting GEOS or destroying the application in memory.

Parameters: none.

Returns: GEOS variables and system hardware in a coldstart state; stack and application space unaffected.

Destroys: a, x, y, **r0-r2**

Description: **FirstInit** is part of the GEOS coldstart procedure. It initializes nearly all GEOS variables and data structures (both global and local), including those which are usually only done once, when GEOS is first booted, such as setting the configuration variables to a default, power-up state.

GEOS calls this routine internally. Applications will not find it especially useful.

Note: The GEOS font variables are not reset by **FirstInit**; a call to **UseSystemFont** may be necessary.

Example:

See also: StartAppl

internal

GetSerialNumber: (c64,C128) C196

Function: Return the 16-bit serial number or pointer to the serial string for the current GEOS kernal

Parameters: none.

Returns: **r0** 16-bit serial number

Destroys: a

Description: **GetSerialNumber** gives an application access to an unencrypted copy of the GEOS serial number or serial string for comparison purposes. You cannot change the actual serial string or number by altering this copy.

Example:

See also:

InterruptMain: (c64,C128)

C100

Function: Main Interrupt Level processing.

Parameters: none.

Returns: nothing.

Destroys: a, x, y, r0-r15

Description: **InterruptMain** is the main GEOS Interrupt Level processing loop and that means different things on different systems.

Note: **InterruptMain** is a subset of the full Interrupt Level process. **InterruptMain** is typically called through the **intTopVector.** An application could conceivably jsr **InterruptMain** to "catch up" on some system updating if interrupts have been disabled for a considerable period of time. **InterruptMain** is not re-entrant, so it is important that interrupts be disabled around the catch-up calls.

Example:

See also: MainLoop.

MainLoop: (c64,C128) C1C3

Function: Main Interrupt Level processing.

Parameters: Direct entry into the GEOS **MainLoop**.

Returns: nothing.

Destroys: a, x, y, r0-r15

Description: Although the term **"MainLoop"** usually refers to GEOS **MainLoop** Level processing, it also represents an entry in the GEOS jump table. By performing a jmp **MainLoop**, the application would be returning to the top of the **MainLoop** Level without letting it run through its normal course of events. The application is expected to return to **MainLoop** Level with an rts, not with a call to **MainLoop.** Hence, this jump table entry is not terribly useful to applications and is primarily used internally by GEOS.

The **MainLoop** jump table entry is perhaps useful when debugging. The system could, conceivably, be returned to a "known state" by resetting the stack pointer and executing a jmp **MainLoop.** Of course, there is no guarantee that this will work.

Example:

See also: InterruptMain.

internal

Panic: (c64,C128) C2C2

Function: Display "system error" dialog box.

Parameters: C64
top word on stack is the system error address+2.

C128
top eight bytes on stack are unused, next word on stack is the system error address+2

Returns: Never returns.

Description: **Panic** puts up a system error dialog box. It is usually not called directly by an application. Usually the global GEOS variable **BRKVector** will contain the address of this routine When GEOS encounters a brk (opcode: $00) instruction in memory, it jumps indirectly through **BRKVector** with system-specific status values on the stack. This usually results in a system error dialog box. The hex address in the dialog box is the address of the offending brk instruction.

An application that patches into **BRKVector** processes brk instructions on its own may need to simulate the normal GEOS course of events by performing a jmp **Panic**.

Although this is not a typical use, an application can use **Panic** as a means of communicating fatal error messages. This may be useful in a beta-test version of a software product, for example.

Example: **FatalError**

See also: InterruptMain.

internal

ResetHandle: (c64,C128) **C003**

Function: Internal routine used during the GEOS boot process.

Parameters: none.

Returns: does not return.

Description: **ResetHandle** is only used during the GEOS boot process. It is not useful to applications and is documented here only because it exists in the jump table.

See also: BootGeos

math

BBMult: (c64,C128) C160

Function: Unsigned byte-by-byte multiply: multiplies two unsigned byte operands to produce an unsigned word result.

Parameters: **x** OPERAND1 — zero-page address of single-byte multiplicand in the low-byte of a word variable (byte pointer to a word variable).

y OPERAND2 — zero-page address of the byte multiplier (byte pointer to a byte variable).

Note: *result = OPERAND1(word) * OPERAND2(word).*

Returns: **x, y,** and byte pointed to by 0PERAND2 unchanged. word pointed to by OPERAND1 contains the word result.

Destroys: **a,r7L,r8**

Description: **BBMult** is an unsigned byte-by-byte multiplication routine that multiplies two bytes to produce a 16-bit word result (low/high order). The byte in 0PERAND1 is multiplied by the byte in 0PERAND2 and the result is stored as a word back in OPERAND1. Note OPERANDl starts out as a byte parameter but becomes a word result with the high-byte at OPERAND 7+1.

Note: Because **r7** and **r8** are destroyed in the multiplication process, they cannot be used to hold either operand.

No overflow can occur when multiplying two bytes because the result always fits in a word($ff*$ff = $fe01).

Example: **8BitMultiply**

See also: **BMult, DMult, Ddiv, DSdiv**

BMult: (c64,C128) C163

Function: Unsigned word-by-byte multiply: multiplies an unsigned word and an unsigned byte to produce an unsigned word result

Parameters: **x** OPERAND1 — zero-page address of word multiplicand (byte pointer to word variable).

y OPERAND2 — zero-page address of multiplier (byte pointer to a word variable — use a word variable; only the low-byte is used in the multiplication process, but the high-byte of the word is destroyed).

Note: *result = OPERAND1(word) * OPERAND2(byte).*

Returns: **x, y** unchanged.
word pointed to by 0PERAND2 has its high-byte set to $00, and its low-byte unchanged word pointed to by OPERAND1 contains the word result.

Destroys: **a, r6-r8.**

Description: **BMult** is an unsigned word-by-byte multiplication routine that multiplies the word at one zero-page address by the byte at another to produce a 16-bit word result. **Bmult** operates by clearing the high-byte of 0PERAND2 and calling **Bmult**. The result is stored as a word back in OPERAND 1.

Note: **r6**, **r7** and **r8** are destroyed in the multiplication process, they cannot be used to hold the operands.

Overflow in the result (beyond 16-bits) is ignored.

Example: **16x8Multiply, ConvToUnits**

See also: **BMult, DMult, Ddiv, DSdiv**

Dabs: (c64,C128) C16F

Function: Compute absolute value of a two's-complement signed word

Parameters: **X** OPERAND — zero-page address of word to operate on (byte pointer to a word variable).

Returns: **x,y** unchanged.
word pointed to by OPERAND contains the absolute value result.

Destroys: **a**

Description: **Dabs** takes a signed word at a zero-page address and returns its absolute value. The address of the word (OPERAND) is passed in x. The absolute value of OPERAND is returned in OPERAND.

The equation involved is: if (value < 0) then value = -value.

Example: **DSmult**

See also: **DNegate**

Ddec: (c64,C128) C175

Function: Decrement a word

Parameters: **X** OPERAND — zero-page address of word to decrement (byte pointer to a word variable).

Returns: **x,y** unchanged.
st **z** flag is set if resulting word **is $0000.**
zero page word pointed to by OPERAND contains the decremented word.

Destroys: **a**

Description: **Ddec** is a double-precision routine that decrements a 16-bit zero-page word. The absolute address of the word is passed in x. If the result of the decrement is zero, then the z flag in the status register is set and can be tested with a subsequent beq or bne. **Ddec** is useful for loops which require a two-byte counter.

Note[3]: the Macro **DecW** should be used in cases where speed is more important then code size. Inner loops should always use **DecW** if space allows. **Ddec** should be used when space is at a premium as it costs only 5 bytes to use. The kernal uses **Ddec** in **CRC** because space in the kernal is more valuable then the speed of the **CRC** procedure that is not normally ever used in an inner loop. See Example **DdecvsDecW**

Example: **Kernal_CRC, DdecvsDecW, DecCounter**

Ddiv: (c64,C128) C169

Function: Unsigned word-by-word (double-precision) division: divides one unsigned word by another to produce an unsigned word result.

Parameters: **x** OPERAND1 — zero-page address of word dividend (byte pointer to a word variable).
y OPERAND2 — zero-page address of word divisor (byte pointer to a word variable).

Note: *result = OPERAND1(word) / OPERAND2(word).*

Returns: **x,y** and word pointed to by 0PERAND2 unchanged,
word pointed to by 0PERAND1 contains the result
r8 contains the fractional remainder (word).

Destroys: **a,r9**

Description: **Ddiv** is an unsigned word-by-word division routine that divides the word at one zero-page address (the dividend) by the word at another (the divisor) to produce a 16-bit word result and a 16-bit word fractional remainder The word in OPERAND 1 is divided by the word in 0PERAND2 and the result is stored as a word back in OPERAND1. The remainder is returned in **r8**.

Note: Because **r8** and **r9** are used in the division process, they cannot be used to hold operands.

If the divisor (OPERAND2) is greater than the dividend (OPERAND1), then the fractional result will be returned as $0000 and OPERAND1 will be returned in **r8**.

Although dividing by zero is an undefined mathematical operation, **Ddiv** makes no attempt to flag this as an error condition and will simply return incorrect results. If the divisor might be zero, the application should check for this situation before dividing as in:

```
    lda     zpage,y         ; get low byte of divisor
    ora     zpage+1,y       ; get high byte of divisor
    bne     10$             ; if either non-zero, go divide
    jmp     DivideByZero    ; else, flag error
10$
    jmp     Ddiv
```

There is no possibility of overflow (a result which cannot fit in 16 bits).

Example: **ConvToUnits, CheckDiskSpace**

See also: **DSDiv, DMult, BBMult, BMult**

DivideBySeven: (Apple)

Function: Divide a byte value by 7

Parameters: **r0L** OPERAND1 — byte to divide/7

Returns: **a** result

Destroys: **a**

Description: Bonus Code Page CBM GEOS has no DivideBySeven in the Kernal like Apple. So here is a block to do a similar operation on an 8 bit value

```
DvBy7:
    lda     r0L
    lsr
    lsr
    lsr
    adc     r0L
    ror
    lsr
    lsr
    adc     r0L
    ror
    lsr
    lsr
    rts
```

DMult: (c64,C128) C166

Function: Unsigned word-by-word (double-precision) multiply: multiplies two unsigned words to produce an unsigned word result.

Parameters: **x** OPERAND 1 — zero-page address of word multiplicand (byte pointer to a word variable).
y OPERAND2 — zero-page address of word multiplier (byte pointer to a word variable).

Note: *results OPERANDl (word) * OPERAND2(word).*

Returns: **x,y,** word pointed to by 0PERAND2 unchanged
word pointed to by OPERAND contains the word result.

Destroys: **a, r6-r8**

Description: **DMult** is an unsigned word-by-word multiplication routine that multiplies the word at one zero-page address by the word at another to produce a 16-bit word result (all stored in low/high order). The word in OPERANDl is multiplied by the word in 0PERAND2 and the result is stored as a word back in OPERANDl.

Note: Because r6, r7 and r8 are destroyed in the multiplication process, they cannot be used to hold the operands.

Overflow in the result (beyond 16-bits) is ignored.

Example: **DSmult**

See also: **Bmult, BBMult, Ddiv, DSDiv**

Dnegate: (c64,C128) C172

Function: Negate a signed word (two's complement sign-switch).

Parameters: **X** OPERAND — zero-page address of word to operate on (byte pointer to a word variable).

Returns: **x,y** unchanged.

Destroys: **a**

Description: **Dnegate** negates a zero-page word. The absolute address of the word OPERAND) is passed in x. The absolute value of OPERAND is returned in OPERAND.

The operation of this routine is: value = (value A $FFFF) + 1.

Example: **DSmult**

See also: **Dabs**

DSDiv: (c64,C128) C16C

Function: Signed word-by-word (double-precision) division: divides one two's complement

Parameters: **X** OPERAND1 — zero-page address of signed word dividend (byte pointer to a word variable).
y OPERAND2 — zero-page address of signed word divisor (byte pointer to a word variable).

Returns: x,**y** unchanged.
r8 the fractional remainder (word).
word pointed to by 0PERAND2 equals its absolute value.
word pointed to by OPERAND 1 contains the word result.

Destroys: **a,r9**

Description: **DSDiv** is a signed, two's complement word-by-word division routine that divides the word in one zero-page pseudo register (the dividend) by the word in another (the divisor) to produce a 16-bit word signed result and a 16-bit word fractional remainder The word in OPERAND 1 is divided by the word in 0PERAND2 and the result is stored as a word back in OPERANDl with the remainder in r8.

The remainder is always positive regardless of the sign of the dividend. This will cause problems with some mathematical operations that expect a signed remainder. The following code fragment will fix this problem:

Note: Because **r8** and **r9** are used in the division process, they cannot be used as the operands.

Although dividing by zero is an undefined mathematical operation, **DSDiv** makes no attempt to flag this as an error condition and simply returns incorrect results. If the divisor might be zero, the application should check for this situation before dividing:

```
          zpage = $00

    DSDivPre:
          lda     zpage,y               ;get low byte of divisor
          ora     zpage+l,y             ;get high byte of divisor
          bne     10$                   ;if either non-zero, go divide
          jmp     DivideByZero          ;else, flag error
    10$
          jmp     DSDiv                 ;divide
```

Example:

See also: **Ddiv, DMult, BBMult, BMult**

DShiftLeft:	(c64,C128)	**C15D**

Function: Arithmetically left-shift a zero-page word.

Parameters: **X** OPERAND — address of the zero-page word to shift (byte pointer to a word variable).

y COUNT — number of times to shift the word left (byte).

Returns: **a,y** unchanged.

y #$ff

st c (carry flag) is set with last bit shifted out of word.

zero page address pointed to by OPERAND contains the shifted word.

Destroys: **nothing**

Description: **DShiftLeft** is a double-precision math routine that arithmetically left-shifts a 16-bit zero-page word (low/high order). The address of the word is passed in x and the number of times to shift the word is passed in y. Zeros are shifted into the low-order bit.

An arithmetic left-shift is useful for quickly multiplying a value by a power of two. One left-shift will multiply by two, two left-shifts will multiply by four, three left-shifts will multiply by eight, and so on: value = value * 2^{count}.

Note: If a COUNT of $00 is specified, the word will not be shifted.

Carry Flag	<- High Byte	<- Low Byte	
C	7-6-5-4-3-2-1-0	7-6-5-4-3-2-1-0	<- 0

Example:

See also: **DShiftRight**

DShiftRight: (c64,C128) C262

Function: Arithmetically right-shift a zero-page word.

Parameters: **X** OPERAND — address of the zero-page word to shift (byte pointer to a word variable).
y COUNT — number of times to shift the word right (byte).

Returns: **a,x** unchanged.
y #$ff
st c (carry flag) is set with last bit shifted out of word.
zero page address pointed to by OPERAND contains the shifted word.

Destroys: **nothing**

Description: **DShiftRight** is a double-precision math routine that arithmetically right-shifts a 16-bit zero-page word (low/high order). The address of the word is passed in x and the number of times to shift the word is passed in y. Zeros are shifted into the high-order bit.

An arithmetic left-shift is useful for quickly multiplying a value by a power of two. One left-shift will multiply by two, two left-shifts will multiply by four, three left-shifts will multiply by eight, and so on: value = value * 2^{count}.

Note: If a COUNT of $00 is specified, the word will not be shifted.

	High Byte ->	Low Byte ->	Carry Flag
0 ->	7-6-5-4-3-2-1-0	7-6-5-4-3-2-1-0	C

Example: **MseToCardPos, ConvToUnits**

See also: **DShiftLeft**

memory

ClearRam: (C64,C128) $C187

Function: Clear a region of memory to $00.

Parameters: **r1** ADDR — address of area to clear (word).
r0 COUNT — number of bytes to clear (0 - 64K) (word).

Returns: nothing.

Destroys: **a,y, r0,r1,r2L**

Description: **ClearRam** clears COUNT bytes starting at ADDR to ADDR+COUNT. It useful for initializing ramsect variables and data sections.

Note: Do not use **ClearRam** to initialize **r0-r2L.** Also, for when space is at a premium, it actually takes fewer bytes to call **i_FillRam** with a fill value of $00.

Note:[1] **ClearRam** sets r2L to $00 and calls **FillRam.**

Example: **InitBuffers**

See also: **FillRam, InitRam.**

memory

CmpFString: (C64,C128) $C26E

Function: Compare two fixed-length strings.

Parameters:

x SOURCE — zero-page address of pointer to source string (byte pointer to a word pointer).

y DEST — zero-page address of pointer to destination string (byte pointer to a word pointer).

a LEN — length of strings (1-255). A LEN value of $00 will cause **CmpFString** to function exactly like **CmpString**, expecting a null terminated source string.

Returns: st status register flags reflect the result of the comparison.

Destroys: **a,x,y**

Description: **CmpFString** compares the fixed-length string pointed to by SOURCE to the string of the same length pointed to by DEST.

CmpFString with a LEN value of $00 causes the routine to act exactly like **CmpString**.

CmpFString compares each character in the strings until there is a non-matching pair. The result of the comparison between the non-matching pair is passed back in the processor status register (st). If the strings match, the z flag is set. This allows the application to test the result of a string comparison with standard test and branch operations:

bne	branch if strings don't match
beq	branch if strings match
bcs	branch if source string is greater than or equal to DEST string
bcc	branch if source string is less than DEST string

Note: The strings may contain internal NULL'S. These will not terminate the comparison.

Example: **Find**

See also: **CmpString, CopyFString**

memory

CmpString:	(C64,C128)	**$C26B**

Function: Compare two null-terminated strings.

Parameters:
- **x** SOURCE — zero-page address of pointer to source null terminated string.
- **y** DEST — zero-page address of pointer to destination null terminated string.

Returns: st status register flags reflect the result of the comparison.

Destroys: **a,x,y**

Description: **CmpString** compares the null-terminated source string pointed to by SOURCE to the destination string pointed to by DEST. The strings are compared a byte at a time until either a mismatch is found or a null is encountered in both strings.

CmpString compares each character in the strings until there is a non-matching pair. The result of the comparison between the non-matching pair is passed back in the processor status register (**st**). If the strings match, the **z** flag is set. This allows the application to test the result of a string comparison with standard test and branch operations:

```
bne    branch if strings don't match
beq    branch if strings match
bcs    branch if source string is greater than or equal to DEST string
bcc    branch if source string is less than DEST string
```

Note: **CmpString** cannot compare strings longer than 256 bytes (including the null). The compare process is aborted after 256 bytes.

Example: **Find2**

CopyFString: (C64,C128) $C268

Function: Copy a fixed-length string.

Parameters:
- **x** SOURCE — zero-page address of pointer to source string (byte pointer to a word pointer).
- **y** DEST — zero-page address of pointer to destination buffer (byte pointer to a word pointer).
- **a** LEN — length of strings (1-255)

Returns: Buffer pointed to by DEST contains copy of source string.

Destroys: **a,x,y**

Description: **CopyFString** copies a fixed-length string pointed to by SOURCE to the buffer pointed to by DEST If the source and destination areas overlap, the source must be lower in memory for the copy to work properly.

Note: Because the LEN parameter is a one-byte value, **CopyFString** cannot copy a string longer than 255 bytes. A LEN value of $00 causes **CopyFString** to act exactly like **CopyString.**

Note: The source string may contain internal NULL'S. These will not terminate the copy operation.

Example: **CopyBuffer**

See also: **CmpFString, CopyString.**

See also: **CopyString, CmpFString, MoveData.**

memory

CopyString:	(C64,C128)	$**C268**

Function: Copy a null-terminated string.

Parameters: **x** SOURCE — zero-page address of pointer to a **NULL** terminated source string (byte pointer to a word pointer).
y DEST **—** zero-page address of pointer to destination buffer (byte pointer to a word pointer).

Returns: Buffer pointed to by DEST contains copy of source string, including the terminating **NULL**

Destroys: **a,x,y**

Description: **CopyString** copies a null terminated string pointed to by SOURCE to the buffer pointed to by DEST. All Characters in the string are copied, including the null-terminator. If the source and destination areas overlap, the source must be lower in memory for the copy to work properly.

CopyString cannot copy more than 256 bytes. The Copy process is aborted after 256 bytes.

Note: **NULL** terminated Strings can be an arbitrary size including > 255

Example: **CopyBuffer**

See also: **CopyFString, CmpString, MoveData.**

DoBOp: (C128) $C2EC

Function: Primitive for communicating with REU (RAM-Expansion Unit) devices.

Parameters:
r0 ADDR1 — address of first block in application memory (word).
r1 ADDR2 — address of second block in application memory (word).
r2 COUNT—number of bytes to operate on (word).
r3L A1BANK—ADDR1 bank: 0 = front RAM; 1 = back RAM (byte).
r3H A2BANK—ADDR2 bank: 0 = front RAM; 1 * back RAM (byte).
y MODE — operation mode:

b1	b0	Description
0	0	move from memory at ADDR1 to memory at ADDR2.
0	1	move from memory SAADDR2 to memory at ADDR1.
1	0	swap memory at ADDR1 with memory at ADDR2.
1	1	verify (compare) memory at ADDR1 against memory at ADDR2.

Note: the ***DoBOp*** *MODE parameter closely matches the low nibble of the Do RAM Op CMD parameter.*

Returns: **r0-r3** unchanged.
When verifying:
x $00 if data matches; $ff if mismatch.
DEV_NOT_FOUND if bank or REU not available.

Destroys: **a,x,y**

Description: **DoBOp** is a generalized memory primitive for dealing with both memory banks on the Commodore 128. It is used by **MoveBData**, **SwapBData**, and **VerifyBData**.

Note: **DoBOp** should only be used on designated application areas of memory. When moving memory within the same bank the destination address must be less than source address. When swapping memory within the same bank, ADDR1 must be less than ADDR2.

Example:

See also: **MoveBData, SwapBData, VerifyBData, DoRAMOp**

memory

FillRam: ,I_FillRam: (C64,C128) $C17B,$C1B4

Function: Fills a region of memory with a repeating byte value.

Parameters: Normal:

r0 COUNT — number of bytes to clear (0 - 64K) (word).
r1 ADDR — address of area to clear (word).
r2L FILL — byte value to fill with (byte).

Inline:

.word COUNT — number of bytes to clear (0 - 64K) (word).
.word ADDR — address of area to clear (word).
.byte FILL — byte value to fill with (byte).

Returns: **r2L** unchanged.

Destroys: **a,y, r0,r1**

Description: **FillRam** fills COUNT bytes starting at ADDR with the FILL byte. This routine is useful for initializing a block of memory to any desired value.

Note: Do not use **FillRam** to initialize **r0-r2L**.

Note:

Example: **InitBuffers**

See also: **ClearRam, InitRam.**

InitRam: (C64,C128) $C181

Function: Table driven initialization for variable space and other memory areas.

Parameters: **r0** TABLE —address of initialization table (word).

Returns: nothing.

Destroys: **a,x,y, r0-r2L**

Description: **InitRam** uses a table of data to initialize blocks of memory to preset values. It is useful for setting groups of variables to specific values. It is especially good at initializing a group of noncontiguous variables in a "two bytes here, three bytes there" fashion.

The initialization table that is pointed to by the TABLE parameter is a data structure made up from the following repeating pattern:

```
.word address               ;start address of this block
.byte count                 ;number of bytes to initialize
.byte bytel,byte2,...       ; count bytes of data

.word address               ;start address of next block
...
```

The table is made of blocks that follow the above pattern, count bytes starting at address are initialized with the next count bytes in the table. (A count value of $00 is treated as 256.) To end the table, use

```
.word NULL
```

where **InitRam** expects the next address parameter.

Note: Do not use **InitRam** to initialize **r0-r2L.**

Example:

See also: **FillRam, ClearRam.**

MoveBData: (C128) $C2E3

Function: Special version of **MoveData** that will move data within either front RAM or back RAM (or from one bank to the other).

Parameters:

r0	SOURCE	— address of source block in memory (word)
r1	DEST	— address of destination block in memory (word)
r2	COUNT	— number of bytes to move (word)
r3L	SRCBANK	— source bank: 0 = back RAM; 1 = front RAM (byte)
r3H	DSTBANK	— destination bank: 0 = back RAM; 1 = front RAM (byte)

Returns: **r0-r3** unchanged.
Destroys: **a,x,y**

Description: **MoveBData** is a block move routine that allows data to be moved in either front RAM, back RAM, or between front and back (bank 1, the front bank, is the normal GEOS application area). If the SOURCE and DEST areas are in the same bank and overlap, DEST. must be less than SOURCE.

MoveBData is especially useful for copying data from front RAM to back RAM or from back RAM to front RAM.

MoveBData uses the **DoBOp** primitive by calling it with a MODE parameter of $00.

Note: **MoveBData** should only be used to move data within the designated application areas of memory. **MoveBData** is significantly slower than **MoveData** and should be avoided if the move will occur entirely within front RAM.

Example:

See also: MoveBData, SwapBData, VerifyBData, DoBOp.

memory

MoveData: ,I_MoveData: (C64,C128) $C2E3

Function: Moves a block data from one area to another.

Parameters: Normal:
r0 SOURCE — address of source block in memory (word)
r1 DEST — address of destination block in memory (word)
r2 COUNT — number of bytes to move (word)

Inline:
.word SOURCE
.word DEST
.word COUNT

Returns: **r0**,**R1**,**R2** unchanged.

Destroys: **a**,**y**

Description: **MoveData** will move data from one area of memory to another. The source and destination blocks can overlap in either direction, which makes this routine ideal for scrolling, insertion sorts, and other applications that need to move arbitrarily large areas of memory around. The move is actually a copy in the sense that the source data remains unaltered unless the destination area overlaps it.

64 & 128: If the DMA **MoveData** option in the Configure program is enabled (GEOS vl.3 and later), **MoveData** will use part of bank 0 of the installed RAM-expansion unit for an ultrafast move operation. An application that calls **MoveData** in the normal manner will automatically take advantage of this selection. An application that relies upon a slower **MoveData** (for timing or other reasons) can disable the DMA-move by temporarily clearing bit 7 of **sysRAMFlg**. This bit can also be used to read the status of the DMA-move configuration.

64: Due to insufficient error checking in GEOS, do not attempt to move more than 30,976 ($7900) bytes at one time when the DMA-move option is enabled. Break the move up into multiple calls to **MoveData**.

128: Due to insufficient error checking in GEOS, do not attempt to move more than 14,592 ($3900) bytes at one time when the DMA-move option is enabled. Break the move up into multiple calls to **MoveData**. **MoveData** should only be used to move data within the standard front RAM application space. Use **MoveData** to move memory within back RAM or between front RAM and back RAM.

Because the RAM-expansion unit DMA follows the VIC chip bank select, an application that is displaying a 40-column screen from back RAM must either disable DMA-moves or temporarily switch the VIC chip to front RAM before the **MoveData** call. (Note to **Author:** Confirm information here. I see no reason REU would not function equally well with back ram. And also be able to do ultra fast transfers between front to back. Using stash ram, bank switch, then fetch ram. Needs testing.)

Note: Do not use **MoveData** on **r0**-r6.

Example:

See also: **MoveBData, CopyString.**

memory

SwapBData: (C128) $**C2E6**

Function: Swaps two regions of memory within either front RAM or back RAM (or between one bank and the other).

Parameters:
r0 ADDR1 — address of first block in application memory (word).
r1 ADDR2 — address of second block in application memory (word).
r2 COUNT — number of bytes to swap (word).
r3L A1BANK — ADDR1 bank: 0 = front RAM; 1 = back RAM (byte)
r3H A2BANK — ADDR2 bank: 0 = front RAM; 1 = back RAM (byte).

Returns: **r0-r3** unchanged.

Destroys: **a,x, y**

Description:

SwapBData is a block swap routine that allows data to be swapped in either front RAM, back RAM, or between front and back. If the ADDR1 and ADDR2 areas are in the same bank and overlap, ADDR2. must be less than ADDR1.

SwapBData is especially useful for swapping data from front RAM to back RAM or from back RAM to front RAM.

SwapBData uses the **DoBOp** primitive by calling it with a MODE parameter of $02.

Note: **SwapBData** should only be used to swap data within the designated application areas of memory.

Example:

See also: MoveBData, VerifyBData, DoBOp.

memory

VerifyBData: (C128) $C2E9

Function: Compares (verifies) two regions of memory against each other. The regions may either be in front RAM or back RAM (or one in front and the other in back).

Parameters:
r0 ADDR1 — address of first block in application memory (word).
r1 ADDR2 — address of second block in application memory (word).
r2 COUNT — number of bytes to swap (word).
r3L A1BANK — ADDR1 bank: 0 = front RAM; 1 = back RAM (byte)
r3H A2BANK — ADDR2 bank: 0 = front RAM; 1 = back RAM (byte).

Returns: **r0-r3** unchanged.
x $00 if data matches; $FF if mismatch.

Destroys: a,y

Description: **VerifyBData** is a block verify routine that allows the data in one region of memory to be compared to the data in another region in memory. The regions may be in either front RAM, back RAM, or in front and back. The ADDRl and ADDR2 areas may overlap even if they are in the same bank.

VerifyBData uses the **DoBOp** primitive by calling it with a MODE parameter of $03.

Note: **VerifyBData** should only be used to compare data within the designated application areas of memory.

Example:

See also: **MoveBData, SwapBData, DoBOp.**

DoRAMOp: (c64 v1.3+,C128) $C2D4

Function: Primitive for communicating with REU (RAM-Expansion Unit) devices.

Parameters:

r0 CBMSRC — address in Main Memory (word).
r1 REUDEST — address in REU bank (word).
r2 COUNT — number of bytes to operate with (word).
r3L REUBANK — REU bank number to use (byte).
y CMD — command to send to REU (byte).

Returns:

r0-r3 unchanged.
x error code: $00 (no error) or
DEV_NOT_FOUND if bank or REU not available.
a REU status byte and'ed with $60

Destroys: **y**

Description: **DoRAMOp** is a very low-level routine for communicating with a RAM expansion unit on a C64 or C128. This routine is a "use at your own risk" GEOS primitive

DoRAMOp operates with the with the RAM-expansion unit directly and handles all the necessary communication protocols and clock-speed save/restore (if necessary).

The CMD parameter is stuffed into the REC Command Register (EXP_BASE+$01). Although **DoRAMOp** does no error checking on this parameter, it expects the high-nibble to be %1001 (transfer with current configuration and disable FFOO decoding). The lower nibble can be one of the following:

%00 Transfer from Commodore to REU.
%01 Transfer from REU to Commodore.
%10 Swap.
%11 Verify.
Note: the low nibble of the ***DoRAMOp*** *CMD parameter closely matches the* ***DoBOp*** *MODE parameter.*

Note: Note: On a Commodore 128, if the VIC chip is mapped to front RAM (with the MMU VIC bank pointer), the REU will read/write using front RAM. Similarly, if the VIC chip is mapped to back RAM, the REU will read/write using back RAM. The REU ignores the standard bank selection controls on the 8510. GEOS 128 defaults with the VIC mapped to front RAM.

For more information on the Commodore REU devices, refer to the Commodore 1764 RAM Expansion Module User's Guide or the 170011750 RAM Expansion Module User's Guide.

Example:

See also: **StashRAM, FetchRAM, SwapRAM, VerifyRAM, DoBOp**

memory

FetchRAM: (c64 v1.3+,C128) C2CB

Function: Primitive for transferring data from an REU

Parameters:
r0 CBMDEST — address in Main Memory to start writing (word).
r1 REUSRC — address in REU bank to start reading (word).
r2 COUNT — number of bytes to fetch (word).
r3L REUBANK — REU bank number to fetch from (byte)

Returns: **r0-r3** unchanged.
x error code: $00 (no error) or
DEV_NOT_FOUND if bank or REU not available.
a REU status byte and'ed with $60 ($40 = success).

Destroys: **y**

Description: **FetchRAM** moves a block of data from a REU BANK into Commodore memory.

FetchRAM uses the **DoRAMOp** primitive by calling it with a CMD parameter of %10010001. $91

Note: Refer to **DoRAMOp** for notes and warnings.

Example:

See also: **StashRAM, SwapRAM, VerifyRAM, DoRAMOp, MoveBData**

memory

StashRAM: (c64 v1.3+,C128) $C2C8

Function: Primitive for transferring data to an REU

Parameters:
r0 CBMSRC — address in Main Memory to start reading (word).
r1 REUDEST — address in REU bank to stash data (word).
r2 COUNT — number of bytes to stash (word).
r3L REUBANK — REU bank number to stash to (byte)

Returns:
r0-r3 unchanged.
x error code: $00 (no error) or
DEV_NOT_FOUND if bank or REU not available.
a REU status byte and'ed with $60 ($40 = success).

Destroys: **y**

Description: **StashRAM** moves a block of data from Commodore memory into an REU bank. This routine is a "use at your own risk" low-level GEOS primitive

StashRAM uses the DoRAMOp primitive by calling it with a CMD parameter of %10010000. $90

Note: Refer to **DoRAMOp** for notes and warnings.

Example:

See also: **SwapRAM, FetchRAM, VerifyRAM, DoRAMOp, MoveBData**

memory

SwapRAM: (c64 v1.3+,C128) $C2CE

Function: Primitive for swapping data between Commodore memory and an REU.

Parameters:
- **r0** CBMADDR — address in Commodore to swap (word).
- **r1** REUADDR — address in REU to swap (word).
- **r2** COUNT — number of bytes to swap (word).
- **r3L** REUBANK — REU bank number to fetch from (byte).

Returns:
r0-r3 unchanged.
x error code: $00 (no error) or
DEV_NOT_FOUND if bank or REU not available.
a REU status byte and'ed with $60 ($40 = successful swap).

Destroys: **y**

Description: **SwapRAM** swaps a block of data in an REU bank with a block of data in Commodore memory.

SwapRAM uses the **DoRAMOp** primitive by calling it with a CMD parameter of % 10010010. $92

Note: Refer to **DoRAMOp** for notes and warnings.

Example:

See also: **StashRAM, FetchRAM, VerifyRAM, DoRAMOp, SwapBData**

memory

VerifyRAM: (c64 v1.3+,C128) $C2D1

Function: Verify (compare) data in main memory with data in an REU.

Parameters:
r0 CBMADDR — address in Commodore to start (word)
r1 REUADDR — address in REU bank to start (word).
r2 COUNT — number of bytes to verify (word)
r3L REUBANK — REU bank number to verify with (byte).

Returns:
r0-r3 unchanged.
x error code: $00 (no error) or
DEV_NOT_FOUND if bank or REU not available.
a REU status byte and'ed with $60
$40 = data match
$20 = data mismatch

Destroys: **y**

Description: **VerifyRAM** Compares a block of data in Commodore memory with a block of data in an REU bank to Verify the contents match. If bit 5 of the a register is set, there was an failed comparison during validation.

VerifyRAM uses the **DoRAMOp** primitive by calling it with a CMD parameter of % 10010011. $93

Note: Refer to **DoRAMOp** for notes and warnings.

Example:

See also: **SwapRAM, FetchRAM, StashRAM, DoRAMOp, VerifyBData**

mouse/sprite

ClearMouseMode	$C19C	Reset the mouse	1-815
IsMseInRegion	$C2B3	Check if mouse is inside a window	1-710
MouseUp	$C18A	Turn on the mouse	1-813
MouseOff	$C18D	Turn off the mouse	1-814
SetMsePic	C2DA	Set and preshift new soft-sprite mouse picture.	158
StartMouseMode	$C14E	Initialize the mouse	1-812
TempHideMouse	$C2D7	Hide soft-sprites before direct screen access.	159

IsMseInRegion: (C64, C128) C2B3

Function: Checks to see if the mouse is within a specified rectangular region of the screen.

Parameters:
r3 X1 — x-coordinate of upper-left (word).
r2L Yl — y-coordinate of upper-left (byte).
r4 X2 — y-coordinate of lower-right (word).
r2H Y2 — y-coordinate of lower-right (byte).

where (X1,Yl) and (X1,Y2) is the upper-left corner of the rectangle and (X2,Y2) is the lower-right corner.

Returns: a **TRUE** if in region, **FALSE** if not in region.
st result of loading **TRUE** or **FALSE** into the a register.

Destroys: nothing.

Description: **IsMseInRegion** tests the position of the mouse against the boundaries of a rectangular region (passed in the same GEOS registers as the Rectangle routine). It returns **TRUE** if the mouse is within the region (inclusive) and **FALSE** if the mouse is outside the region. Because the st register reflects the result of loading **TRUE** or **FALSE** into the accumulator, the call can be followed by a branch instruction that tests the result, such as:

```
     beq         InRegion          ; branch if mouse was in region
       -or-
     bne         NotInRegion       ; branch if mouse not in region
```

Note: Interrupts should always be disabled around a call to **IsMseInRegion** . If the php-sei-plp method is used, be aware that the plp will reset the st flags. If this is troublesome, it may warrant creating a new version of **IsMseInRegion** that does its own interrupt disable and leaves the values in the st register intact: See **NewIsMseInRegion**

Example: **NewIsMseInRegion**

See also: HorizontalLine.

MouseOff:	(C64, C128)	**C18D**

Function: Temporarily disables the mouse pointer and GEOS mouse tracking.

Parameters: nothing.

Returns: nothing.

Modifies:

mobenble	sprite #0 bit cleared by **DisablSprite**.
mouseOn	clears the MOUSEON_BIT

Destroys: a

Description: **MouseOff** temporarily disables the mouse cursor and GEOS mouse tracking by clearing the proper bit in mouseOn and calling DisablSprite. Applications can call **MouseOff** temporarily disable the mouse. The mouse can be reenabled to its previous state by calling **MouseUp.**

Example:

See also: MouseUp, ClearMouseMode.

MouseUp: (C64, C128) **C18A**

Function: Reenables the mouse pointer and GEOS mouse tracking.

Parameters: nothing.

Returns: nothing.

Modifies: **mobenble** sprite #0 bit cleared by **DisablSprite**.
mouseOn sets the MOUSEON_BIT

Destroys: a

Description: **MouseUp** reenables the mouse cursor and GEOS mouse tracking after a call to **MouseOff** by setting the proper bits in mouseOn and mobenble. StartMouseMode calls this routine.

Example:

See also: MouseOff, ClearMouseMode.

SetMsePic: (C64, C128) $C184

Function: Uploads and pre-shifts a new mouse picture for the software sprite handler.

Parameters: **r0** **MSEPIC** — pointer to 32 bytes of mouse sprite image data or one of the following special codes:
ARROW

Returns: nothing

Destroys: **a, x, y, r0-r15**

Description: The software sprite routines used by GEOS 128 in 80-column mode treat the mouse sprite (sprite #0) differently than the other sprites. Sprite #0 is optimized and hardcoded to provide reasonable mouse-response while minimizing the flicker typically associated with erasing and redrawing a fastmoving object. The mouse sprite is limited to a 16x8 pixel image. The image includes a mask of the same size and both are stored in a pre-shifted form within internal GEOS buffers. For these reasons, a new mouse picture must be installed with **SetMsePic** (as opposed to a normal **DrawSprite**). **SetMsePic** pre-shifts the image data and lets the soft-sprite mouse routine know of the new image.

SetMsePic accepts one parameter: a pointer to the mask and image data or a constant value for one of the predefined shapes. If a user-defined shape is used, the data that MSEPIC points to is in the following format:

16 bytes	16x8 "cookie cutter" mask. Before drawing the software mouse sprite, GEOS and's this mask onto the foreground screen. Any zero bits in the mask, clear the corresponding pixels. One bits do not affect the screen.
16 bytes	16x8 sprite image. After clearing pixels with the mask data, the sprite image is or'ed into the area. Any one bits in the sprite image set the corresponding pixels. Zero bits do not affect the screen.

Note: **SetMsePic** calls **HideOnlyMouse.**
Note[3]: **ARROW Equate - ARROW = $00**

Example:

See also: **TempHideMouse, HideOnlyMouse, DrawSprite**

TempHideMouse: (C64, C128) $C184

Function: Temporarily removes soft-sprites and the mouse pointer from the graphics screen.

Parameters: nothing.

Uses: **graphMode**

Destroys: **a, x**

Description: **TempHideMouse** temporarily removes all soft-sprites (mouse pointer and sprites 2-7) unless they are already removed. This routine is called by all GEOS graphics routines prior to drawing to the graphics screen so that software sprites don't interfere with the graphic operations. An application that needs to do direct screen access should call this routine prior to modifying screen memory.

The sprites will remain hidden until the next pass through **MainLoop.**

Note: In 40-column mode (bit 7 of **graphMode** is zero)**, TempHideMouse** exits immediately without affecting the hardware sprites.

Example:

See also: **HideOnlyMouse**

print driver

StartASCII: (C64, C128) 7912

Function: Enable ASCII text mode printing. 7912

Parameters: rl WORKBUF — pointer to a 640-byte work buffer for use by the printer driver.(word). **PrintASCII** uses this work area as an intermediate buffer, the buffer must stay intact throughout the entire page.

Returns: **x STATUS — printer error code; $00 = no error.**

Destroys: **a, y, r0-r15**

Description: **StartASCII** enables ASCII text mode printing. An application calls **StartASCII** at the beginning of each page. It assumes that **InitForPrint** has already been called to initialize the printer.

StartASCII takes control of the serial bus by opening a fake Commodore file structure and requests the printer (device 4) to enter listen mode. It then sends the proper control sequences to place the printer into text mode.

Example:

See also: PrintASCII, StopPrint, StartPrint

process

BlockProcess: (C64, C128) C103

Function: Block a processes events.

Parameters: x PROCESS — process to block (0 to n-1, where n is the number of processes in the table) (byte).

Returns: x unchanged.

Destroys: a

Description: **BlockProcess** causes **MainLoop** to ignore the runnable flag of a particular process so that if a process timer reaches zero (causing the process to become runnable) no process event is generated until the process is subsequently unblocked with a call to **UnblockProcess. BlockProcess** stops the process the **MainLoop** level. Refer to **FreezeProcess** to stop the process at the Interrupt Level.

BlockProcess does not stop the countdown timer, which continues to decrement at Interrupt Level (assuming the process is not frozen). When the timer reaches zero, the runnable flag is set and the timer is restarted. As long as the process is blocked, though, **MainLoop** ignores this runnable flag and, therefore, never generates an event. When a blocked process is later unblocked, **MainLoop** checks the runnable flag. If the runnable flag was set during the time the process was blocked, this pending event generates a call to the appropriate service routine. Only one event is generated when a process is unblocked, even if the timer reached zero more than once.

Note: If a process is already blocked, a redundant call to **BlockProcess** has no effect.

Example:

```
SuspendClock:
     ldx   #CLOCK_PROCESS    ; x <- process number of the clock
     jmp   BlockProcess      ; block that particular process
```

See also: **UnblockProcess, FreezeProcess**

FreezeProcess:	(C64, C128)	**C112**

Function: Freeze a process's countdown timer at its current value.

Parameters: x PROCESS — process to freeze (0 to n-1, where n is the number of processes in the table) (byte).

Returns: x unchanged.

Destroys: a

Description: **FreezeProcess** halts a process's countdown timer so that it is no longer decremented every vblank. Because a frozen timer will never reach zero, the process will not become runnable except through a call to **EnableProcess.** When a process is unfrozen with **UnfreezeProcess**, its timer again begins counting from the point where it was frozen.

Note: If a process is already frozen, a redundant call to **FreezeProcess** has no effect.

Example:

See also: UnfreezeProcess, BlockProcess

process

InitProcesses: (C64, C128) C103

Function: Initialize and install a process data structure.

Parameters: a NUM_PROC — number of processes in table (byte).
r0 PTABLE — pointer to process data structure to use (word).

Returns: **r0** unchanged.

Destroys: a, x, y, **rl**

Description: **InitProcesses** installs and initializes a process data structure. All processes begin as frozen, so their timers arc not decremented during vblank. Processes can be started individually with **RestartProcess** after the call to **InitProcesses.**

InitProcesses copies the process data structure into an internal area of memory hidden from the application. GEOS maintains the processes within this internal area, keeping track of the event routine addresses, the timer initialization values (used to reload the timers after they time-out), the current value of the timer, and the state of each process (i.e., frozen, blocked, runnable). The application's copy of the process data structure is no longer needed because GEOS remembers this information until a subsequent call to **InitProcesses.**

Note: Although processes are numbered starting with zero, NUM_PROC should be the actual number of processes in the table. To initialize a process table with four processes, pass a *NUM_PROC* value of $04. When referring to those processes (i.e., when calling routines such as **UnblockProcess**), use the values $00-$03. Do not call **InitProcesses** with a *NUM_PROC* value of $00 or a *NUM_PROC* value greater than **MAX_PROCESSES** (the maximum number of processes allowable).

Example:

See also: Sleep, RestartProcess

EnableProcess: (C64, C128) C109

Function: Makes a process runnable immediately.

Parameters: x PROCESS — process to enable (0 - n-l, where n is the number of processes in the table) (byte).

Returns: x unchanged.

Destroys: a

Description: **EnableProcess** forces a process to become runnable on the next pass through **MainLoop**, independent of its timer value.

EnableProcess merely sets the runnable flag in the process table. When **MainLoop** encounters an unblocked process with this flag set, it will attempt to generate an event just as if the timer had decremented to zero.

EnableProcess has no privileged status and cannot override a blocked process. However, because it doesn't depend on or affect the current timer value, the process can become runnable even with a frozen timer.

EnableProcess is useful for making sure a process runs at least once, regardless of the initialized value of the countdown timer. It is also useful for creating application-defined events which run off of **MainLoop**: a special process can be reserved in the data structure but never started with **RestartProcess**. Any time the desired event-state is detected, a call to **EnableProcess** will generate an event on the next pass through **MainLoop**. **EnableProcess** can be called from Interrupt Level, which allows a condition to be detected at Interrupt Level but processed during **MainLoop**.

Example:

See also: **InitProcesses**, **RestartProcess**, **UnfreezeProcess**, **UnblockProcess**

RestartProcess: (C64, C128) C106

Function: Reset a process's timer to its starting value then unblock and unfreeze the process.

Parameters: x PROCESS — process to restart (0 - n-1 where n is the number of processes in the table) (byte).

Returns: **r0** unchanged.

Destroys: a, x, y, **r1**

Description: **RestartProcess** sets a process's countdown timer to its initialization value then unblocks and unfreezes it Use **RestartProcess** to initially start a process after a call to **InitProcesses** or to rewind a process to the beginning of its cycle.

Note: **RestartProcess** clears the runnable flag associated with the process, thereby losing any pending call to the process.

RestartProcess should always be used to start a process for the first time because **InitProcesses** leaves the value of the countdown timer in an unknown state.

Example:

See also: **InitProcesses, EnableProcess, UnfreezeProcess, UnblockProcess.**

process

Sleep: (C64, C128) C199

Function: Pause execution of a subroutine ("go to sleep") for a given time interval.

Parameters: r0 DELAY — number of vblanks to sleep (word),

Returns: does not return directly to caller (see description below).

Destroys: a, x, y

Description: **Sleep** stops executing the current subroutine, forcing an early rts to the routine one level lower, putting the current routine "to sleep." At Interrupt Level, the DELAY value associated with each sleeping routine is decremented. When the associated DELAY value reaches zero, **MainLoop** removes the sleeping routine from the sleep table and performs a jsr to the instruction following the original jsr **Sleep**, expecting a subsequent rts to return control back to **MainLoop.** For example, in the normal course of events, **MainLoop** might call an icon event service routine (after an icon is clicked on). This service routine can perform a jsr **Sleep**. **Sleep** will force an early rts, which, in this case, happens to return control to **MainLoop.** When the routine awakes (after DELAY vblanks have occurred), **MainLoop** performs a jsr to the instruction that follows the original jsr **Sleep.** When this wake-up jsr occurs, it occurs at some later time the contents of the processor registers and GEOS **pseudo-registers** are uninitialized. A subsequent rts will return to **MainLoop.**

Sleeping in Detail:

1: The application calls **Sleep** with a jsr **Sleep.** The jsr places a return address on the stack and transfers the processor to the **Sleep** routine.

2: **Sleep** pulls the return address (top two bytes) from the stack and places those values along with the DELAY parameter in an internal sleep table.

4: **Sleep** executes an rts. Since the original caller's return address has been pulled from the stack and saved in the sleep table, this rts uses the next two bytes on the stack, which it assumes comprise a valid return address. (Note: it is imperative that this is in fact a return address; do not save any values on die stack before calling **Sleep.**)

5: At Interrupt Level GEOS decrements the sleep timer until it reaches zero.

6: On every pass, **MainLoop** checks the sleep timers. If one is zero, then it removes that sleeping routine from the table, adds one to the return address it pulled from the stack (so it points to the instruction following the jsr **Sleep**), and jsr's to this address. Because no context information is saved along with the **Sleep** address, the awaking routine cannot depend on any values on the stack, in the GEOS pseudoregisters, or in the processor's registers.

Note: A DELAY value of $0000 will cause the routine to sleep only until the next pass through **MainLoop.**

When debugging an application, be aware that **Sleep** alters the normal flow of control.

Example: BeepThrice

See also: InitProcesses.

UnblockProcess: (C64, C128) C10F

Function: Allow a process's events to go through.

Parameters: x PROCESS — number of process (0 - n-1, where n is the number of processes in the table) (byte).

Returns: **x** unchanged.

Destroys: a

Description: **UnblockProcess** causes **MainLoop** to again recognize a process's runnable flag so that if a process timer reaches zero (causing the process to become runnable) an event will be generated.

Because the GEOS Interrupt Level continues to decrement the countdown timer as long as the process is not frozen, a process may become runnable while it is blocked. As long as the process is blocked, however, **MainLoop** will ignore the runnable flag. When the process is subsequently unblocked, **MainLoop** will recognize a set runnable flag as a pending event and call the appropriate service routine. Multiple pending events are ignored: if a blocked process's timer reaches zero more than once, only one event will be generated when it is unblocked. To prevent a pending event from happening, use **RestartProcess** to unblock the process.

Note: If a process is not blocked, an unnecessary call to **UnblockProcess** will have no effect.

Example:

See also: **BlockProcess, UnfreezeProcess , EnableProcess, RestartProcess.**

UnfreezeProcess: (C64, C128) C115

Function: Resume (unfreeze) a process's countdown timer.

Parameters: x PROCESS — number of process (0 - n-1, where n is the number of processes in the table) (byte).

Returns: **x** unchanged.

Destroys: a

Description: **UnfreezeProcess** causes a frozen process's countdown timer to resume decrementing. The value of the timer is unchanged; it begins decrementing again from the point where it was frozen. If a process is not frozen, a call to **UnfreezeProcess** will have no effect.

Note: If a process is not frozen, a call to **UnfreezeProcess** will have no effect.

Example:

See also: **FreezeProcess, BlockProcess**

Sprite

DisablSprite: (C64, C128) C1D5

Function: Disable a sprite so that it is no longer visible.

Parameters: **r3L** SPRITE — sprite number (byte).

Returns: nothing.

Alters: **mobenble**

Destroys: **a, x**

Description: **DisablSprite** disables a sprite so that it is no longer visible. Although there are eight sprites available, an application should only directly disable sprite #2 through sprite #7 with **DisablSprite.** Sprite #0 (the mouse pointer) is always enabled when GEOS mouse-tracking is enabled (disable mouse-tracking with **MouseOff**), and sprite #1 (the text cursor) should be disabled with **PromptOff.**

Example:

See also: **EnablSprite, MouseOff, PromptOff, DrawSprite.**

sprite

DrawSprite: (C64, C128) C1C6

Function: Copy a 64-byte sprite image to the internal data buffer that is used for drawing the sprites.

Parameters: **r3L** SPRITE — sprite number (byte).
r4 DATAPTR — pointer to 64-bytes of sprite image data (word).

Returns: nothing.

Alters: internal sprite image.

Destroys: a, y, **r5**

Description: **DrawSprite** copies 64-bytes of sprite image data to the internal buffer that is used for drawing the sprites. **DrawSprite** does not affect the enabled/disabled status of a sprite, it only changes the image definition.

Although there are eight sprites available, an application should limit itself to sprites #2 through #7 because GEOS reserves sprite #0 for the mouse cursor and sprite #1 for the text prompt.

The 64 bytes are copied to the VIC sprite data area, which is located in memory immediately after the color matrix. The size information byte (byte 64) is unused by GEOS 64 but is copied to the data area, nonetheless. A SPRITE value of $00 can be used to change the shape of the mouse cursor.

The data is transferred to the VIC sprite area (regardless of the current graphics mode). This data is used by the VIC chip in 40-column mode and by the soft sprite handler in 80-column mode. The last byte (byte 64) of the sprite definition is used as the size information byte by the soft-sprite handler. In 80-column mode, the sprite is not visually updated until the next time the soft-sprite handler gets control. To change the mouse cursor, the application can use a SPRITE value of $00 in 40-column mode or call **SetMsePic** in 80-column mode (doing both is a simple solution: it will do no harm regardless of the graphics mode).

The data is transferred to an internal sprite area. The last byte (byte 64) of the sprite definition is used as the size information byte. The sprite is not visually updated until the next time the soft-sprite handler gets control. The soft-sprite handler will draw sprite #1 through sprite #7. In no case should the SPRITE parameter be $00; a value of $00 will most likely trample GEOS.

Example

See also: **PosSprite, EnablSprite, DisablSprite.**

EnablSprite: (C64, C128) C1D2

Function: Enable a sprite so that it becomes visible.

Parameters: **r3L** SPRITE — sprite number (byte).

Returns: nothing.

Alters: **mobenble**

Destroys: **a, x**

Description: **EnablSprite** enables a sprite so that it becomes visible. Although there are eight sprites available, an application should only directly enable sprites #2 through #7 with **EnablSprite.** Sprite #0 (the mouse pointer) is enabled through MouseOn and StartMouseMode, and sprite #1 (the text cursor) should be enabled with **PromptOn.**

Example:

See also: **DisablSprite, MouseOff, PromptOff, DrawSprite, PosSprite.**

sprite

PosSprite: (C64, C128) C1CF

Function: Positions a sprite at a new GEOS (x,y) coordinate.

Parameters:
r3L SPRITE — sprite number (byte).
r4 XPOS — x-position of sprite (word).
r5L YPOS — y-position of sprite (byte).

Returns: nothing.

Alters:
mobNxpos
msbNxpos
reqXposN
mobnypos

where N is the number of the sprite being positioned.

Destroys: **a, x, y, r6**

Description: **PosSprite** positions a sprite using GEOS coordinates (not C64 hardware sprite coordinates). PosSprite does not affect the enabled/disabled status of a sprite, it only changes the current position.

Although there are eight sprites available, an application should only directly position sprites #2 through #7 with **PosSprite.** Sprite #0 (the mouse pointer) should not be repositioned (except, maybe through mouseXPos and **mouseYPos**), and sprite #1 (the text cursor) should only be repositioned with **stringX** and **stringY.**

C64: The positions are translated to C64 hardware coordinates and then stuffed into the VIC chip's sprite positioning registers. The C64 hardware immediately redraws the sprite at die new position.

C128: The positions are translated to C64 hardware coordinates and then stuffed into the VIC chip's sprite positioning registers. This data is used by the VIC chip in 40-column mode and by the soft-sprite handler in 80-column mode. In 80-column mode, the sprite is not visually updated until the next time the soft-sprite handler gets control.

Example:

See also: **DrawSprite, EnablSprite, DisablSprite.**

Utility

Bell	N/A	Play a bell sound	177
CallRoutine	C1D8	pseudo-subroutine call. $0000 aborts call.	178
CRC	C20E	Cyclic Redundancy Check calculation.	179
DoInlineReturn	C2A4	Return from inline subroutine.	180
GetRandom	C187	Calculate new random number.	181
ToBasic	C241	Pass Control to Commodore BASIC.	182

utility

Bell: (Apple) **N/A**

Function: Makes a brief beeping sound

Parameters: none.

Returns: nothing.

Destroys: **a**

Description: **Bell** sounds a 1000 Hz signal. The sound lasts approximately 1/10th of a second.

Note: **Bell** does not exists in Commodore Geos. This code provides the behavior of the Apple Bell.

```
; Author: Dan Kaufman (w Chris Hawley)

sidBase = $D400
voicelRegs = sidBase
        freqLol         = voice1Regs
        freqHil         = voice1Regs + 1
        PWLol           = voice1Regs + 2
        PWHil           = voice1Regs + 3
        controlRegl     = voice1Regs + 4
        att_decl        = voice1Regs + 5
        sus__rell       = voice1Regs + 6

        FCLo            = voice1Regs + 7 + $07
        FCHi            = voice1Regs + 7 + $08
        res_filt        = voice1Regs + 7 + $09
        mode_vol        = voice1Regs + 7 + $0A
        pulse           = %01000001
        SOUND_ON        = $30
Bell:
        PushB     CPU_DATA            ;switch to I/O space
        LoadB     CPU_DATA,#IO_IN
        LoadB     controlReg1,#0
        sta       att_dec1
        LoadB     mode_vol, #$18
        LoadB     sus_rell,#SOUND_ON
        LoadW     PWLol,#$800
        LoadB     FCLo,#0
        sta       FCHi
        sta       res_filt
        LoadB     att_decl,#6
        LoadB     sus_rell,#0
        LoadB     freqLol,#$DF
        LoadB     freqHil/#$25
        LoadB     controlRegl,#pulse
        PopB      CPU_DATA            ;return to memory space
        rts
```

Example: **BeepThrice**

See also: **Ddec**

utility

CallRoutine: (C64,C128) C1D8

Function: Perform a pseudo-subroutine call, checking first for a null address (which will be ignored).

Parameters: a [ADDRESS — low byte of subroutine to call.
x]ADDRESS — high byte of subroutine to call.
where ADDRESS is the address of a subroutine to call.

Returns: depends on subroutine at ADDRESS.

Destroys: depends on subroutine at ADDRESS.

Description: **CallRoutine** offers a clean and simple way to perform an indirect jsr through a vector or call a subroutine with an address from a jump table. Before simulating a jsr to the address in the x and a registers, it also checks for a null address ($0000). If the address is $0000 (x=$00 and a=$00), **CallRoutine** performs rts without calling any subroutine address. This makes it easy to nullify a vector or an entry in a jump table by using a $0000 value.

GEOS frequently uses **CallRoutine** when calling through vectors. This is why placing a $0000 into **keyVector**, for example, causes GEOS ignore the vector. Other examples of this usage are **intTopVector**, **intBotVector**, and **mouseVector**.

Note: **CallRoutine** modifies the st register prior to performing the jsr. It, therefore, cannot be used to call routines that expect processor status flags as parameters (flags may be returned in the st register, however). **CallRoutine** may be called from Interrupt Level (off of routines in **IntTopVector** and **IntBotVector**). Do not use **CallRoutine** to call inline (i_) routines, as it will not return properly.

Example: **HandleCommand**, **KeyTrap**

See also:

utility

CRC: (c64,C128) C20E

Function: 16-bit cyclic redundancy check (CRC).

Parameters: **r0** DATA - pointer to start of data (word).
r1 LENGTH - of bytes to check (word).

Returns: **r2** CRC value for the specified range (word).

Destroys: **a, y, r0-r3L**

Description: **CRC** calculates a 16-bit cyclic-redundancy error-checking value on a range of data. This value can be used to check the integrity of the data at a later time. For example, before saving off a data file, and application might perform a **CRC** on the data and save the value along with the rest of the data. Later, when the application reloads the data, it can perform another **CRC** on it and compare the new value with the old value. If the two are different, the data has unquestionably been corrupted.

Note: Given the same data, **CRC** will produce the same value under all versions of GEOS.

Note[1]: This routine is called by the bootup routines to compute the checksum of GEOS BOOT. This checksum is used to create the interrupt vector address. The reason for this was to prevent piracy. This can be used to check the integrity of a memory region.

Example: **Kernal_CRC**

```
MAGIC_VALUE = $0317                ; CRC value that we're looking for
DATA_SIZE=$2434                    ; Size of data

.ramsect
       buffer    .block DATA_SIZE
.psect

Checksum:
       LoadW r0,#buffer            ; r0 <- data area to checksum
       LoadW r1,#DATA_SIZE         ; r1 <- bytes in buffer to check
       jsr   CRC                   ; r2 <- CRC value for data area
       CmpWI r2,MAGIC_VALUE        ; return status to caller
       rts
```

See also: **Ddec**

utility

DoInlineReturn: (c64,C128) C2A4

Function: Return from an inline subroutine.

Parameters: a DATABYTES — number of inline data bytes following the jsr plus one(byte).
stack top byte on stack is the status register to return (execute a php just before calling).

Returns: (to the inline jsr) x, y unchanged from the jmp **DolnlineReturn.** st register is pulled from top of stack with a plp.

Destroys: **a**

Description: **DoInlineReturn** simulates an rts from an inline subroutine call, properly skipping over the inline data. Inline subroutines (such as the GEOS routines which begin with i) expect parameter data to follow the subroutine call in memory. For example, the GEOS routine i_**Rectangle** is called in the following fashion:

```
jsr   i_Rectangle       ;subroutine call
.byte yl,y2             ;inline data
.word x1,x2
jsr   FrameRectangle    ;returns to here
```

Now if **i_Rectangle** were to execute a normal rts, the program counter would be loaded with the address of the inline data following the subroutine call. Obviously, inline subroutines need some means to resume processing at the address following the data. **DoInlineReturn** Provides this facility. The normal return address is placed in the global variable returnAddress. This is the return address as it is popped off the stack, which means it points to the third byte of the inline jsr (an rts increments the address before resuming control). The status registers is pushed onto the stack with a php, **DoInlineReturn** is called with the number of inline data bytes plus one in the accumulator, and control is returned at the instruction following the inline data.

Inline subroutines operate in a consistent fashion. The first thing one does is pop the return address off of the stack and store it in returnAddress. It can then index off of returnAddress as in Ida (returnAddress),y to access the inline parameters, where the y-register contains $01 to access the first parameter byte, $02 to access the second, and so on (not $00, $01, $02, as might be expected because the address actually points to the third byte of the inline jsr). When finished, the inline subroutine loads the accumulator with the number of inline data bytes and executes a jmp **DoInlineReturn.**

Note: **DoInlineReturn** must be called with a jmp (not a jsr) or an unwanted return address will remain on the stack. The x and y registers are not modified by **DoInlineReturn** and can be used to pass parameters back to the caller. Inline calls cannot be nested without saving the contents of returnAddress. An inline routine will not work correctly if not called directly through a jsr (e.g., **CallRoutine** cannot be used to call an inline subroutine).

Example: **i_VerticalLine**

See also: **Ddec**

utility

GetRandom: (C64,C128) C187

Function: Creates a 16-bit random number.

Parameters: none.

Uses: **random** random seed for next random number.

Alters: **random** random contains a new 16-bit random number.

Returns: depends on subroutine at ADDRESS.

Destroys: a

Description: **GetRandom** produces a new pseudorandom (not truly random) number using the following linear congruential formula:

random = (2*(**random**+1) // 65521)
(remember: // is the modulus operator)

The new random number is always less than 65221 and has a fairly even distribution between 0 and 65521.

Note: GEOS calls **GetRandom** during Interrupt Level processing to automatically keep the random variable updated. If the application needs a random number more often than random can be updated by the Kernal, then **GetRandom** must be called manually.

Example:

See also:

utility

ToBasic: (C64,C128) C187

Function: Removes GEOS and passes control to Commodore BASIC with the option of loading a non-GEOS program file (BASIC or assembly-language) and/or executing a BASIC command.

Parameters:
r0 CMDSTRING — pointer to null-terminated command string to send to BASIC interpreter.
r5 DIR_ENTRY — pointer to the directory entry of a standard Commodore file (PRG file type), which itself can be either a **BASIC** or **ASSEMBLY** GEOS-type file. If this parameter is $0000, then no file will be loaded.
r7 LOADADDR — if r5 is non-zero, then this is the file load address. For a **BASIC** program, this is typically $801. If **r5** is zero and a tokenized **BASIC** program is already in memory, then this value should point just past the last byte in the program. If **r5** is zero and no program is in memory, this value should be $803, and the three bytes at $80O-$802 should be $00.

Returns: N/A
Destroys: N/A

Description: **ToBasic** gives a GEOS application the ability to run a standard Commodore assembly-language or **BASIC** program. It removes GEOS, switches in the **BASIC** ROM and I/O bank, loads an optional file, and sends an optional command to the **BASIC** interpreter.

Once **ToBasic** has executed, there is no way to return directly to the GEOS environment unless the RAM areas from $C000 through $CO7F are preserved (those bytes may be saved and restored later). To return to GEOS, the called program can execute a jump to $C000 (**BootGEOS**).

A program in the C64 environment can check to see if it was loaded by GEOS by checking the memory starting at $C006 for the ASCII (not CBMASII) string "GEOS BOOT" If loaded by GEOS, the program can check bit 5 of $CO12: if this bit is set, ask the user to insert their GEOS boot disk; if this bit is clear, GEOS will reboot from the RAM expansion unit To actually return to GEOS, set **CPU_DATA** to $37 (KRNL_BAS_IO_IN) and jump to $C000

Example: **LoadBASIC**

See also: **BootGeos**

text

GetCharWidth: (C64, C128) C1C9

Function: Calculate the pixel width of a character as it exists in the font (in its plaintext form). Ignores any style attributes.

Parameters: a CHAR — character code of character (byte).

Uses: **curlndexTable**

Returns: **a** character width in pixels.

Destroys: y

Description: **GetCharWidth** calculates the width of the character before any style attributes are applied. If the character code is less than 32, $00 is returned. Any other character code returns the pixel width as calculated from the font data structure. The sprites will remain hidden until the next pass through **MainLoop**.

Because **GetCharWidth** does not account for style attributes, it is useful for establishing the number of bits a character occupies in the font data structure.

Note: In 40-column mode (bit 7 of **graphMode** is zero), **TempHideMouse** exits immediately without affecting the hardware sprites.

Example:

See also: **GetRealSize**

GetNextChar: (C64, C128) C2A7

Function: Retrieve the next character from the keyboard queue.

Parameters: none.

Returns: **a** keyboard character code of character or NULL if no characters available.

Alters: **pressFlag** if the call to **GetNextChar** removes the last character from the queue, then the **KEYPRESS_BIT** is cleared.

Destroys: x

Description: **GetNextChar** checks the keyboard queue for a pending keypress and returns a non-zero value if one is available. This allows more than one character to be processed without returning to **MainLoop**

Example:

See also: **GetString**

text

GetRealSize: (C64, C128) **C1B1**

Function: Calculate the printed size of a character based on any style attributes.

Parameters: **a** CHAR — character code of character (byte).

Uses: **curHeight**
baselineOffset

Returns: **y** character width in pixels (with attributes).
x character height in pixels (with attributes).
a character baseline offset (with attributes).

Destroys: nothing.

Description: **GetRealSize** calculates the width of the character based any style attributes The character code must be 32 or greater. If the character code is **USELAST**, the value in **lastWidth** is returned. Any other character code returns the pixel width as calculated from the font data structure and the MODE parameter.

lastWidth is local to the GEOS Kemal and therefore inaccessible to applications.

Example:

```
; Calculate size of largest character in current font
    lda     #'W'                        ; capital W is a good choice
    ldx     #(SET_BOLD|SET_OUTLINE)     ; widest style combo
    jsr     GetRealSize                 ; dimensions come back in x,y
```

See also: **GetCharWidth**

GetString:	(C64, C128)	**C1BA**

Function: Get a string from the keyboard using a cursor prompt and echoing characters to the screen as they are typed. Runs concurrently with **MainLoop**.

Parameters:

r0	BUFR — pointer to string buffer. When called this buffer can contain a null-terminated default string (if no default string is used, the first byte of the buffer must be NULL). This buffer must be at least MAX_CH+1 bytes long.
r1L	FLAG — $00 = use system fault routine; $80 = use fault routine pointed to by r4 (byte).
r2L	MAX_CH — maximum number of characters to accept (not including the null-terminator).
r11	XPOS — x-coordinate to begin input (word).
r1H	YPOS — y-coordinate of prompt and upper-left of characters. To calculate this value based on baseline printing position, subtract the value in **baselineOffset** from the baseline printing position (byte).
r4	FAULT — optional (see FLAG) pointer to fault routine.
keyVector	STRINGDONE — routine to call when the string is terminated by the user typing a carriage return. $0000 = no routine provided.

Uses:

*at call to **GetString** :*

curHeight	for size of text prompt.
baselineOffset	for positioning default string relative to prompt

any variables used by **PutString**.

while accepting characters:

keyVector	vectors off of **MainLoop** through here with characters.
stringX	current prompt x-position.
stringY	current prompt y-position.
string	pointer to start of string buffer.

any variables used by **PutChar**.

Returns:

*from call to **GetString** :*

keyVector	address of System String Service.
stringFaultVec	address of fault routine being used
stringX	starting prompt x-position.
stringY	starting prompt y-position.
string	BUFR (pointer to start of string buffer).

when done accepting characters:

x	length of string / index to null
string	BUFR (pointer to start of string buffer).
keyVector	$0000
stringFaultVec	$0000

Destroys:

*at call to **GetString**:*

r0-r13, **a**, **x**, **y**.

Description: **GetString** installs a character handling routine into **GetString** and returns immediately to the caller. During **MainLoop**, the string is built up a character at a time in a buffer. When the user presses [Return], GEOS calls the STRINGDONE routine with the starting address of the string in string and the length of the string in the x-register.

The following is a breakdown of what **GetString** does:

1: Variables local to the **GetString** character input routine are initialized. Global string input variables such as **string**, **stringX**, and **stringY** are also initialized.

2: **PutString** is called to output the default input string stored in the character buffer. If no default input string is desired, the first byte of the buffer should be a **NULL**.

3: The **STRINGDONE** parameter in **keyVector** is saved away and the address of the **GetString** character routine (SystemStringService) is put into keyVector.

4: If the application supplied a fault routine, install it into **StringFaultVec**, otherwise install a default fault routine.

5: The prompt is initialized by calling **InitTextPrompt** with the value in **curHeight. PromptOn** is also called.

6: Control is returned to the application.

lastWidth is local to the GEOS Kemal and therefore inaccessible to applications.

Note: String is not null-terminated until the user presses [Return]. To simulate a [Return], use the following code:

```
;Simulate a CR to end GetString
LoadB   keyData,#CR          ; load up a [Return]
lda     keyVector            ; and go through keyVector
ldx     keyVector+1          ; so SystemStringService
jsr     CallRoutine          ; thinks it was pressed
```

This will also terminate the **GetString** input.

Note: This note courtesy of Bill Coleman...Because **GetString** runs off of **MainLoop**, it is a good idea to call **GetString** from the top level of the application code and return to **MainLoop** while characters are being input. That is, while at the top level of your code you can call **GetString** like this:

```
        jsr      GetString   ; Start GetString  going
        rts                  ; and return immediately to MainLoop so
                             ; that string can be input.
```

Since the routine specified by the **STRINGDONE** value stored in **keyVector** is called when the user has finished entering the string, that is where your application should again take control and process the input.

Note[2]: If the user manages to type off the end of the screen, specifically past **rightMargin**, **GetString** will stop echoing characters although it will still enter the characters into the buffer.

See also: **PutChar, PutString, GetNextChar.**

text

InitTextPrompt: (C64, C128) C1C0

Function: Initialize sprite #1 for use as a text prompt.

Parameters: **a** HEIGHT — pixel height for the prompt (byte)

Alters: **alphaFlag** %10000011

Destroys: a, x, y

Description: **InitTextPrompt** initializes sprite #1 for use as a text prompt. The sprite image is defined as a one-pixel wide vertical line of *HEIGHT* pixels. If *HEIGHT* is large enough, the double-height sprite flags will be set as necessary. *HEIGHT* is usually taken from **curHeight** so that it reflects the height of the current font.

The text prompt will adopt the color of the mouse pointer.

Example:

See also: **PromptOn, PromptOff**

LoadCharSet:	(C64, C128)	**C1CC**

Function: Begin using a new font.

Parameters: **r0** FONTPTR — address of font header.

Returns: **r0** unchanged

Alters:

curHeight	height of font
baselineOffset	number of pixels from top of font to baseline.
cardDataPntr	pointer to current font image data.
curIndexTable	pointer to current font index table.
curSetWidth	pixel width of font bitstream in bytes.

Destroys: a, y

Description: **LoadCharSet** uses the data in the character set data structure to initialize the font variables for the font pointed at by the *FONTPTR* parameter.

Example:

See also: **LoadCharSet**

text

PromptOff: **(C64, C128)** C29E

Function: Turn off the prompt (remove the text cursor from the screen).

Parameters: none.

Alters: **alphaFlag** (($C0 & (**alphaFlag** & $40) | PROMPT_DELAY), *where PROMPT_DELAY = 60.*

Destroys: a, x, **r3L**

Description: **PromptOff** removes the text prompt from the screen. To ensure the prompt will remain invisible until a subsequent call to **PromptOn,** interrupts must be disabled before calling **PromptOff.**

Example: **KillPrompt**

See also: **InitTextPrompt, PromptOn.**

text

PromptOn: (C64, C128) C29B

Function: Turn on the prompt (show the text cursor on the screen).

Parameters: none.

Uses: **stringX** cursor x-position (word).
stringY cursor y-position (byte).

Alters: **alphaFlag** (($C0 & (**alphaFlag** & $40) | PROMPT_DELAY),*where PROMPT_DELAY = 60.*

Destroys: a, x, **r3L**

Description: **PromptOn** makes the text prompt visible and active at the position specified by **stringX** and **stringY**. The prompt will flash once every second (PROMPT_DELAY). If **stringX** or **StringY** are changed, the cursor will repositioned automatically the next time the cursor flashes. To make the update immediate, call **PromptOn**. Before **PromptOn** is called for the first time, **InitTextPrompt** should be called.

Example: **KillPrompt**

See also: **PromptOn, PromptOff**

PutChar:	(C64, C128)	**C145**

Function: Process a single character code (both escape codes and printable characters).

Parameters:
a CHAR — character code (byte).
r11 XPOS — x-coordinate of left of character (word).
r1H YPOS — y-coordinate of character baseline (word).

Uses:
dispBufferOn display buffers to direct output to.
currentMode character style.
leftMargin left margin to contain character.
rightMargin right margin to contain characters.
(following set by **LoadCharSet***)*
curHeight height of current font.
baselineOffset number of pixels from top of font to baseline.
cardDataPntr pointer to current font image data.
curlndexTable pointer to current font index table data.
curSetWidth pixel width of font bitstream in bytes.

Returns:
r11 x-position for next character.
rlH unchanged

Destroys: a, x, y, **rlL, r2-rlO, rl2, rl3**

Description: **PutChar** is the basic character handling routine. If the character code is less than 32, **PutChar** will look-up a routine address in an internal jump table to process the escape code. Only send implemented escaped codes to **PutChar**.

If the character code is 32 or greater, **PutChar** treats it as a printable character. First it establishes the printed size of the character with any style attributes (**currentMode**) then checks the character position against the bounds in **leftMargin** and **rightMargin**. If the left edge of the character will fall to the left of **leftMargin**, then the width of the character is added to the x-position in **rll** and **PutChar** vectors through **StringFaultVec**. If the right edge of the character will fall to the right of **rightMargin**, then **PutChar** vectors through **StringFaultVec** without altering the x-position. The character is not printed in either case.

Assuming no margin fault, **PutChar** will print the character to the screen at the desired position. Any portion of the character that lies above **windowTop** or below **windowBottom** will not be drawn.

PutChar cannot be used to directly process multi-byte character codes such as **GOTOX** or **ESC_GRAPHICS** unless **r0** is maintained as a string pointer when **PutChar** is called (as it is in **PutString**). See **PutString** for more information.

Example:

See also: **SmallPutChar, PutString, PutDecimal.**

text

PutDecimal: (C64, C128) C184

Function: Format and print a 16-bit positive integer.

Parameters:

a	FORMAT — formatting codes (byte) — see below.
r0	NUM — 16-bit integer to convert and print (word).
r11	XPOS — x-coordinate of leftmost digit (word).
r1H	YPOS — y-coordinate of baseline (word).

Uses: Same as **PutChar**

Returns:

r11	x-position for next character.
r1H	unchanged

Destroys: a, x, y, **r0, r1L, r2-r10, r12, r13**

Description: **PutDecimal** converts a 16-bit positive binary integer to ASCII and sends the result to **PutChar.** The number is formatted based on the FORMAT parameter bytes in the a-registers as follows:

FORMAT:

7	6	5 4 3 2 1 0
b7	b6	b0-b5

b7 justification:
  1 = left
  0 = right.
b6 leading zeros:
  1 = suppress
  0 = print.
b5-b0 field width in pixels (only used if right justifying).

The following constants may be used:

SET_LEFTJUST
SET_RIGHTJUST
SET_SUPPRESS
SET_NOSUPPRESS

Note: The maximum 16-bit decimal number is 65535 ($FFFF), so the printed number will never exceed five characters.

Example:

See also: **PutChar, PutString, SmallPutChar**

PutString, i_PutString: (C64, C128) C148, C1AE

Function: Print a string to the screen. C1AE

Parameters: Normal:

r0 STRING — pointer to string data (word).
r11 XPOS — x-coordinate of left of character (word).
r1H YPOS — y-coordinate of character baseline (word).

InLine:

data appears immediately after the jsr i_**PutString**
.word XPOS x-coordinate.
.byte YPOS y-coordinate.
.byte STRINGDATA null terminated string (no length limit)

Uses: Same as **PutChar**

Returns: **r11** x-position for next character.
rlH unchanged

Destroys: a, x, y, **rlL, r2-rlO, rl2, rl3**

Description: **PutString** passes a full string of data to **PutChar** a character at a time. PutChar maintains **r0** as a running pointer into the string and so supports multi-byte escape codes such as GOTOXY.

If a character exceeds one of the margins, **PutChar** will vector through **stringFaultVec** as appropriate. **r0**, **rll**, and **rlH** will all contain useful values (current string pointer, x-position, and y-position, respectively). For more information, refer to "String Faults (Left or Right Margin Exceeded)" in Chapter XX FIXME

Basic operation of **PutString**

```
PutString:
5$
        ldy     #0              ;use zero offset
        lda     (r0),y          ;get character
        beq     10$             ;exit if NULL terminator
        jsr     PutString       ;otherwise process char
        IncW    r0              ;move to next byte in string
        bra     5$              ;and loop through again
10$
        rts                     ;exit
```

Note: Unless a special string fault routine is placed in **stringFaultVec** prior to calling **PutString**, a margin fault will be ignored and **PutString** will attempt to print the next character.

Example:

See also: **PutChar, PutString**

SmallPutChar: (C64, C128) **C202**

Function: Print a single character without the **PutChar** overhead

Parameters:

a	CHAR — character code (byte).
r11	XPOS — x-coordinate of left of character (word).
r1H	YPOS — y-coordinate of character baseline (word).

Uses: Same as **PutChar**

Returns:

r11	x-position for next character.
r1H	unchanged

Destroys: a, x, y, **r1L, r2-r10, r12, r13**

Description: **SmallPutChar** is a bare bones version of **PutChar**. **SmallPutChar** will not handle escape codes, does no margin faulting, and does not normalize the x coordinates on GEOS 128.

SmallPutChar will assume the character code is a valid and printable character Any portion of the character that lies above **windowTop** or below **windowBottom** will not be drawn. If a character lies partially outside of **leftMargin** or **rightMargin**, **SmallPutChar** will only print the portion of the character lies within the margins. **SmallPutChar** will also accept small negative values for the character x-position, allowing characters to be clipped at the left screen edge.

Note: Partial character clipping at the **leftmargin**, including negative x-position clipping, is not supported by early versions of GEOS 64 (earlier than v1.4) — the entire character is clipped instead Full **leftmargin** clipping is supported on all other versions of GEOS: GEOS 64 v1.4 and above, GEOS 128 (both in 64 and 128 mode.

Like **PutChar**, 159 is the maximum CHAR value that **SmallPutChar** will handle correctly. Most fonts will not have characters for codes beyond 129.

Example:

128: **DOUBLE_W,ADD1_W** cannot be used on r11

See also: **PutChar, PutString**

wheels

UseSystemFont: (C64, C128) C14B

Function: Begin using default system font (BSW 9).

Parameters: none.

Alters:

curHeight	height of font
baselineOffset	number of pixels from top of font to baseline.
cardDataPntr	pointer to current font image data.
curIndexTable	pointer to current font index table.
curSetWidth	pixel width of font bitstream in bytes.

Returns: nothing.

Destroys: a, x, y, **r0**

Description: **UseSystemFont** calls **LoadCharSet** with the address of the always-resident BSW 9 font.

128: In 80-column mode a double-width BSW 9 font is substituted

Example:

See also: **LoadCharSet.**

Chapter 2 Wheels 4.4

Wheels Kernal

GetNewKernal	$9d80	Load New Kernal Group
RstrKernal	$9d83	Unload Kernal Group

wheels

GetNewKernal: (Wheels 4.4 64,128) $9D80

Function: Load Modular Kernal Group

Parameters: a GROUPNBR to load | RUNFLAG
RUNFLAG Bit 6 of a.
1 Selected Kernal Group Swapped into memory at 5000-5FFF.
0 First Routine in group executed. (Kernal Group swapped back).

Destroys: **(unknown)**

Return: varies depending on RUNFLAG and GROUPNBR.

Description: **GetNewKernal** allows access to the Extended Kernal available in 4.4.

If RUNFLAG is 0 **GetNewKernal** behaves as a far jsr to the first routine in the Kernal Group. Performing the following...

Swap the extended kernel group into memory.
Execute the first routine in the group.
Swap the kernal back out of memory.
Control is returned to the caller.

If RUNFLAG is set:

Extended Kernal is swapped into memory at 5000-5FFF.
Control is returned to the caller.

(Kernal will remain in memory until a call to **RstrKernal** to swap it back.)

Note: Kernal Groups are loaded from the Last REU bank which is reserved exclusively for the 4.4 Kernal.

Note: Caller cannot be in the Range 5000-5FFF as that address range is swapped out with the Kernal Group

Example:

```
KG_REU=$00
NO_RUN=%01000000
RUN_FIRST=%00000000

LoadREUGrp:
    lda  #KG_REU|NO_RUN
    jmp  GetNewKernal
```

See also: **RstrKernal**

wheels

RstrKernal: (Wheels 4.4 64,128) $**9D83**

Function: Unload Extended Kernal group.

Parameters: none

Destroys: **a**

Return: **nothing**

Alters: Memory area from 5000-5FFF is restored to its previous contents.

Description: **RstrKernal** is used to restore the memory area 5000-5FFF after using **GetNewKernal** to load in an extended Kernal Group.

Example:

```
GRP_REU=$00
NO_RUN=%01000000
RUN_FIRST=%00000000

.ramsect
    freeBanks: .block 1

.psect

GetBanksFree:
    lda   #GRP_REU|NO_RUN ; Select REU Group . And don't execute 1st
    jsr   GetNewKernal    ; Load in Kernal Group.
    jsr   GetRAMInfo      ; Call Kernal Group function to get
                          ; number of free REU banks.
    MoveB r4H,freeBanks   ; save the result
    jmp   RstrKernal      ; Remove Kernal Group, restoring 5000-5FFF
                          ; to its previous contents.
```

See also: **GetNewKernal**

GetRAMBam: (Wheels 4.4 64,128) $5000

Function: Creates a copy of Wheels' expansion RAM's 'BAM'.

Parameters: none

Uses: Expansion RAM's BAM homespace at $5025-44 and workspace at $5045-64. RAM BAM's checksum at $5024.

Return: Y = $FF.

Destroys: **a,y**

Description: **GetRAMBam** copies 32 bytes from $5025-44 to $5045-64. I assume that the 32-byte buffer at $5025-44 contains the Wheels' expansion RAM BAM (referred to as 'homespace') and a copy is made on $5045-64 (referred to as 'workspace') for further operations. That way, if something goofs up, $5045-64 is trashed, but the original BAM for the Wheels' expansion RAM is largely untouched at $5025-44. Any application that needs to modify Wheels' expansion RAM makeup needs to call this routine first to ensure that they are working off of a copy of the RAM BAM.

Secondly, the routine creates a checksum byte value (stored at $5024), computing the entire memory range from $5025-$5105. This entire memory range is swapped in/out along with the entire Group 0 module from Wheels' expansion RAM. Still unclear is the purpose of the checksum value.

Example:

Note[4]:

See also: **PutRAMBam**

PutRAMBam: (Wheels 4.4 64,128) $5003

Function: Writes back the copy of Wheels' expansion RAM's 'BAM' to its original buffer.

Parameters: none

Uses: Expansion RAM's BAM homespace at $5025-44 and workspace at $5045-64. RAM BAM's checksum at $5024.

Return: Y = $FF.

Destroys: **a,y**

Description: **PutRAMBam** copies 32 bytes from $5045-64 to $5025-44. I assume that the 32-byte buffer at $5045 contains a copy of the Wheels' expansion RAM BAM and this copy is written back to the original 32-byte buffer at $5025-44. Essentially the opposite of the **GetRAMBam** routine. An application can ensure that whatever changes are made to this copy of the Wheels' expansion RAM BAM are confirmed and written back to its original buffer.

Secondly, the routine creates a checksum byte value (stored at $5024), computing the entire memory range from $5025-$5105. This entire memory range is swapped in/out along with the entire Group 0 module from Wheels' expansion RAM.

Example:

Note[4]:

See also: **GetRAMBam**

wheels

AllocRAMBlock: (Wheels 4.4 64,128) $**5009**

Function: Allocates a 64Kb bank of Wheels' expansion RAM for a program to use.

Parameters: Bank number ($01-$FE) in r6L.

Uses: ramExpSize ($88C3).
bit masking tables at $522d-34.
expansion RAM's BAM workspace at $5045-64.
RAM BAM checksum at $5024.

Return: X = $00 (no error.)
X=$06 (BAD_BAM).

Destroys: a,x,y

Description: **AllocRAMBlock** allocates memory in 64Kb banks in Wheels' expansion RAM by marking the RAM BAM workspace wherever appropriate. You cannot pass a value (in r6L) of $00 and the last bank of Wheels' expansion RAM as they are already allocated by the operating system. Similarly, you cannot pass a value exceeding the last bank of Wheels' expansion RAM, i.e., passing a bank number of $80 when the Wheels' expansion RAM consists of a 2Mb REU. You cannot allocate an already allocated bank. In those cases of errors, the routine will return X with a BAD_BAM error value.

Once a valid bank number has been passed, the routine will clear a bit in the corresponding RAM BAM entry to '0', indicating that this bank is now allocated for a program use. The RAM BAM's checksum value is recomputed and stored at $5024.

Example:

Note[4]:

See also: AllocAllRAM, FreeRAMBlock, RamBlkAlloc.

AllocAllRAM: (Wheels 4.4 64,128) $**5006**

Function: Allocates all banks in expansion RAM.

Parameters: Nothing.

Uses: Expansion RAM's BAM workspace at $5045-$5064. RAM BAM's checksum at $5024.

Return: Y = $FF

Destroys: a,y

Description: **AllocAllRAM** allows a program to allocate all banks in expansion RAM for their own use. Then a new checksum is computed for the expansion RAM's BAM and is stored at $5024. Make sure that **GetRAMBam** has been called and that there is a Wheels' expansion RAM BAM already in place in Wheels' local RAM workspace.

Example:

Note[4]:

See also: AllocRAMBlock, FreeRAMBlock, RamBlkAlloc.

wheels

FreeRAMBlock:	(Wheels 4.4 64,128)	$**500C**

Function: Frees up a 64Kb bank in expansion RAM for the Wheels OS.

Parameters: Bank number ($01-$FE) in r6L.

Uses: ramExpSize ($88c3).
bit masking tables at $522d-34.
expansion RAM's BAM workspace at $5045-64.
RAM BAM checksum at $5024.

Return: X = $00 (no error)
X=$06 (BAD_BAM).

Destroys: a,x,y

Description: **FreeRAMBlock** frees up Wheels' expansion RAM memory in 64Kb chunks by marking the RAM BAM (copy) wherever appropriate. The routine merely uses the bank value ($01-$FE) passed via r6L and calls up the toggle routine, marking the copy of the RAM BAM's corresponding entry to a value of '1', freeing it. This toggle routine is shared by the AllocRAMBlock routine. It then recomputes the RAM BAM's checksum value and stores the value back onto $5024.

Example:

Note[4]:

See also: AllocAllRAM, AllocRAMBlock, RamBlkAlloc.

GetRAMInfo: (Wheels 4.4 64,128) $**500F**

Function: Gets a snapshot of available RAM from Wheels' expansion RAM.

Parameters: Nothing.

Uses: **r2, r3, r4, r6, r9L.**
ramExpSize ($88c3).
bit masking tables at $522d-34.
expansion RAM's BAM homespace at $5025-44.
expansion RAM's BAM workspace at $5045-64.
RAM BAM checksum at $5024.
temporary space at $52c7.

Calls: **GetRAMBam.**
RamBlkAlloc.

Return: **r2L**=# of consecutive free 64Kb banks. (If **r2L** contains a zero value, then there are no banks in Wheels' expansion RAM available.)
r3L=# of starting bank pointing to the largest free area.
r4H=# of free 64Kb banks.

Destroys: a,x,y

Description: **GetRAMInfo** Gives you a snapshot of available RAM that a program can use in accessing Wheels' expansion RAM. Basically what it does is that it loads **r2L** with the value (minus one) located at ramExpSize and loads **r3L** with $00 and then calls the **RamBlkAlloc** routine.

The **RamBlkAlloc** routine will allocate the largest available contiguous memory area and pass its parameters upon return. Upon returning from the **RamBlkAlloc** routine, it then calls the **GetRAMBam** routine to undo any changes that **RamBlkAlloc** routine may have made. Next, it recomputes the RAM BAM's checksum value and stores it back onto $5024. The resulting parameters are then returned back to the calling program.

Example:

Note[4]:

See also:

wheels

RamBlkAlloc: (Wheels 4.4 64,128) $**5012**

Function: Allocates banks in Wheels' expansion RAM for program use.

Parameters: r2L=# of contiguous 64Kb banks needed.
r3L=0 for the Wheels OS to select a bank or # of desired starting bank in Wheels' expansion RAM.

Uses: r2, r3, r6, r9L
bit masking tables at $522d-34.
expansion RAM's BAM homespace at $5045-64.
RAM BAM checksum at $5024.
temporary space at $52c7.

Calls: **GetRAMBam**.
RamBlkAlloc.

Return: r3L=Start # of 64Kb bank that was just allocated.
X = $00 (no error)
INSUFF_SPACE if ram allocation was not successful.

Destroys: a,x,y

Description: **RamBlkAlloc** is similar to the **AllocRAMBlock** routine, except that this can allocate a bunch of contiguous 64Kb banks in succession in Wheels' expansion RAM memory. Make sure that the program will free these same banks upon exiting, because the Wheels OS will leave these banks alone and will not allow other programs to allocate these same banks.

While you can specify a starting 64Kb bank of expansion RAM prior to calling this routine, the routine will search for the next available 64Kb bank of expansion RAM if the desired starting bank is already in use. Just like **AllocRAMBlock**, it will mark a contiguous area in Wheels' expansion RAM memory as allocated by marking a value of '0' in the corresponding BAM entries in the expansion RAM BAM workspace. The RAM BAM's checksum value is recomputed and stored at $5024.

Example:

Note[4]**:**

See also: **AllocAllRAM, AllocRAMBlock, FreeRAMBlock.**

RemoveDrive: (Wheels 4.4 64,128) $**5015**

Function: Removes a RAM drive from the Wheels OS system.

Parameters: Nothing.

Uses:

r4L,	
driveType	type of drive to open
numDrives	Number of Drives in the system
curDrive	currently active disk drive.
curType	Currently Active Drive Type.
curDevice	currently active device.
ramBase,	

Calls: **SetDevice**
PurgeTurbo.

Return: **Nothing.**

Destroys: a,y

Description: **RemoveDrive** checks **numDrives** to ensure that there are at least two drives running. No sense in deleting the only drive in a system! Using the drive number passed in **r4L**, it calls **SetDevice** & **PurgeTurbo**. Next, it zeroes out the corresponding **driveType** entry and the ramBase entry in these two tables. It then zeroes out **curType**, **curDrive**, **curDevice** and finally decreases the value found in **numDrives** by one.

This has the effect of removing a RAM drive from the Wheels OS system. It does not actually remove the RAMdisk in a physical sense! It is just that some pointers indicating the existence of a RAMdrive is simply wiped out.

See Also: Interestingly enough, there is no corresponding AddDrive entry. Maybe this routine is contained in the Toolbox instead and is not in the Group 0 section of the Wheels OS Kernal.

Example:

Note[4]:

See also:

SvRamDevice: (Wheels 4.4 64,128) $**5018**

Function: Create a partition in Wheels' expansion RAM.

Parameters: **r0** = pointer to a 16 byte null-terminated partition name.
r2L = # of contiguous 64Kb banks needed.
r3L = Starting Bank Number.
(0 = Let the Kernal decide which starting 64Kb bank to use).
r7L = ID number, can be any number less than 128.
(Any number higher than 128 designates a RAMdisk).
Y = Partition Nbr. 1-8 (Max of 8)
0 Let the Kernal decide the partition #.

Uses: **r0, r1, r2L, r3L, r7L,**
Temporary address space at $5373 indicating the current partition number.
RAM BAM checksum at $5024.
RAM BAM homespace ($5025-44).
RAM BAM workspace ($5045-64).
ID# table for each partition at $5065-6c.
Start bank table for each partition at $506d-74.
# of 64Kb banks table for each partition at $5075-7c.
Partition names table for each partition from $507d-$5104.

Calls: **GetRAMBam**
RamBlkAlloc.

Return: **Nothing.**

Destroys: a,x,y

Description: **SvRamDevice** can permanently set up a partition in Wheels' expansion RAM by creating a partition in the Wheels' extended kernal. This way, the partition will be reserved by a program and have it survive various computing sessions. Once created, a program can simply reuse that partition over and over instead of individually allocating and freeing up expansion RAM memory every time it boots.

This routine makes heavy use of the four tables mentioned above, from $5065-$5104. This way, ram partitions have their own corresponding entries and are preserved for use in future computing sessions. The RAM BAM workspace at $5045 is written back to the homespace ($5025-44) and the checksum is generated and stored at $5024.

Example:

Note[4]:

See also: **DelRamDevice, RamDevInfo.**

wheels

DelRamDevice: (Wheels 4.4 64,128) $**5018**

Function: Remove a partition from Wheels' expansion RAM.

Parameters: Y = Partition Nbr. 1-8 (Max of 8)
0 Let the Kernal decide the partition #.

Uses: **r1, r3H, r6L**
Temporary address space at $5373 indicating the current partition number.
RAM BAM workspace ($5045-64).
ID# table for each partition at $5065-6c.
Start bank table for each partition at $506d-74.
# of 64Kb banks table for each partition at $5075-7c.
Partition names table for each partition from $507d-$5104.

Calls: **GetRAMBam**
PutRAMBam

Return: X = $00 (no error)
X != 0 an Error Occurred

Destroys: a,x,y

Description: **DelRamDevice** removes a partition from the Wheels' expansion RAM by modifying pointers indicating its existence. The partition in question really isn't removed in a physical sense, but rather, its pointers are zeroed out.

First, it calls **GetRAMBam** to load the RAM BAM workspace, and frees up the corresponding BAM (64Kb bank) entries occupied by the partition as available for future use. Next, it zeroes out the corresponding entries in the four tables mentioned above, from $5065-$5104. Finally, it calls **PutRAMBam** to save the RAM BAM workspace.

Example:

Note[4]:

See also: SvRamDevice, RamDevInfo.

RamDevInfo: (Wheels 4.4 64,128) $501E

Function: Get stats on a partition residing in Wheels' expansion RAM.

Parameters: Y = Partition Nbr. 1-8 (Max of 8)

Uses: **r1, r2L, r3L, r7L.**
ID# table for each partition at $5065-6c.
Start bank table for each partition at $506d-74.
# of 64Kb banks table for each partition at $5075-7c.
Partition names table for each partition from $507d-$5104.

Return:
r2L contains the # of contiguous 64Kb banks value extracted from the corresponding entry at $5075-7c.
r3L contains the start bank value extracted from the corresponding entry at $506d-74.
r7L contains the ID# value extracted from the corresponding entry at $5065-6c.
r1 points to the corresponding name for the partition located at $507d-$5104.

Destroys: a,x,y

Description: **RamDevInfo** gets stats about a particular partition, i.e., its ID#, its size in # of 64Kb banks, its starting bank value and its name.

Example:

Note[4]:

See also: SvRamDevice, RamDevInfo.

Examples

atoms

atoms

KeyTrap:

```
.psect
.include _upper

      T_Action:
            'A','B','C','D'          ; Keyboard commands to act on. Case insensitive
      T_ActL:                        ; Low Pointer table to Action Handlers
            .byte [SetDrv8
            .byte [SetDrv9
            .byte [SetDrv10
            .byte [SetDrv11
      T_ActH:                        ; High Pointer table to Action Handlers
            .byte ]SetDrv8
            .byte ]SetDrv9
            .byte ]SetDrv10
            .byte ]SetDrv11
      T_ACTCNT=*-T_ActH

      Init:
            LoadW keyVector, KeyTrap
            rts

      KeyTrap:                       ; Routine hooked into keyVector
            lda   keyData            ; Get Keypress and
            jsr   Upper              ; Convert it to Uppercase
            ldy   #T_ACTCNT-1        ; Search action table for a hit
      10$
            cmp   T_Action,y
            beq   20$
            dey
            bpl   10$
            rts                      ; No Action found for press. Exit
      20$
            lda   T_ActL,y           ; Action Found.
            ldx   T_ActH,y
            jmp   CallRoutine        ; Execute the Handler


      SetDrv8:
            lda   #8
cldaI SetDrv9,    #9
cldaI SetDrv10,   #10
cldaI SetDrv11,   #11
            jsr   SetDevice          ; Set Device to user selected number
            jsr   OpenDisk           ; open the disk
            jmp   ErrHndlr           ; Generic Error Handler.
                                     ; Displays error dialog or
                                     ; does nothing on no error.
```

hardware

GetFPS:

```
;Author PBM
;PASS: Nothing
;Return: a = fps
;        minus flag set if known model was not found
;        minus return should never happen without a bug in C64Model


models: .byte %00,%01,%10,%11
NBR_MODELS=*-models
frates: .byte 50,60,60,50

GetFPS:
    jsr C64Model
10$
    ldx #NBR_MODELS-1
    cmp    models,x
    beq    90$
    dex
    bpl    10$
    lda    [TRUE
    rts
90$
    lda    frates,x
    rts
```

C64Model:

```
;~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
; Detect PAL/NTSC
; Original Name: DetectC64Model
; Author: TWW
; ~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
; 312 rasterlines -> 63 cycles per line PAL
;                 => 312 * 63 = 19656 Cycles / VSYNC  => #>76  %00
; 262 rasterlines -> 64 cycles per line NTSC V1
;                 => 262 * 64 = 16768 Cycles / VSYNC  => #>65  %01
; 263 rasterlines -> 65 cycles per line NTSC V2
;                 => 263 * 65 = 17095 Cycles / VSYNC  => #>66  %10
; 312 rasterlines -> 65 cycles per line PAL DREAN
;                 => 312 * 65 = 20280 Cycles / VSYNC  => #>79  %11
C64Model:
    ;-- Use CIA #1 Timer B to count cycled in a frame
    lda #$FF
    sta $DC06
    sta $DC07           ; Latch #$FFFF to Timer B
10$
    bit $D011
    bpl 10$             ; Wait until Raster > 256
20$
    bit $D011
    bmi 20$             ; Wait until Raster = 0
    ldx #%00011001
    stx $DC0F           ; Start Timer B (One shot mode
                        ;(Timer stops automatically when underflow))
30$
    bit $D011
    bpl 30$             ;Wait until Raster > 256
40$
    bit $D011
    bmi 40$             ;Wait until Raster = 0
    sec
    sbc $DC07           ;Hibyte number of cycles used
    and #%00000011
    rts
```

DetectC64Model Source from CodeBase64
https://codebase64.org/doku.php?id=base:detect_pal_ntsc

Note[3]: I believe this will also work on a 128 in 40 column mode. Need to test FIXME.

math

8BitMultiply:

```
;*****************************************************
;   8BitMultiply-    8 Bit unsigned multiply.
;
;   pass:    x - zpage address of multiplicand
;            y - zpage address of multiplier
;
;   returns: unsigned result in address pointed to by x
;            x, y unchanged
;
; Multiply rlL by rlH and store the word result in r2

8BitMultiply:
    MoveB    rlL,r2L          ; r2L <- rlL copy of OPERAND1
    ldx      #r2L             ; x <- source register address
    ldy      #r1H             ; y <- destination register address
    jsr      BBMult           ; r2 <- r2L * r2H do multiplication
    rts
```

16x8Multiply:

```
;**************************************************************************
;   16x8Multiply -   16x8 Bit unsigned multiply.
;
;   pass:    x - zpage address of multiplicand
;            y - zpage address of multiplier
;
;   returns: unsigned result in address pointed to by x
;            x, y unchanged
;
; Multiply the value in r9 by 87 and store the result back in r9
; (rl is destroyed)
;
;**************************************************************************

16x8Multiply:
    ldx      #r9              ; point to OPERAND1 in r9
    LoadB    rlL,#87          ; rl <- 87 (OPERAND2)
    ldy      #rl              ; point to OPERAND2 in rl
    jsr      Bmult            ; r9 <- r9 * rlL
    rts
```

ConvToUnits:

```
;*************************************************************************
;   This routine converts a pixel measurement to inches or, optionally,
;   centimeters, at the rate of 80 pixels per inch or 31.5 pixels per
;   centimeter.
;
;   pass:    r0 - number to convert (in pixels)
;
;   return:  r0 - inches / centimeters
;            rlL - tenths of an inch / millimeters.
;   destroys: a, x, y, r0-rl, r8-r9
;*************************************************************************
; Assembler time decision on whether inches or centimeters is to be used.
.if AMERICAN
    INCHES = TRUE
.else
    ;Metric
    INCHES = FALSE
.endif

ConvToUnits:                  ; First, convert r0 to length in 1/20 of
                              ; standard units

.if INCHES
                              ; For Inches, need to multiply by
                              ;      20          1
                              ; ------------- = ---
                              ; 80 dots/inch     4
                              ; which amounts to a divide by four
    ldx      #r0
    ldy      #2
    jsr      DShiftRight
.else
                              ; For Centimeters, need to multiply by
                              ;      20          1
                              ; ------------- = ---
                              ; 31.5 dots/cm     63
                              ;
    LoadB    rl,#40           ; First multiply by 40
    ldx      #r0              ; (word value)
    ldy      #rl              ; (byte value)
    jsr      Bmult            ; r0 * r0*40 (byte by word multiply)
    LoadW    rl,#63           ; then divide by 63
    ldx      #r0              ;
    ldy      #rl              ;
    jsr      Ddiv             ; r0- r0/63
.endif
    ;-- Start of Common Code   ; r0 * result in 1/20ths
    IncW     r0                           ; add in one more 1/20th, for rounding
    LoadW    rl,#20           ; now divide by 20 (to move decimal over one)
    ldx      #r0              ; dividend
    ldy      #rl              ; divisor
    jsr      Ddiv             ; r0 = r0 /20 (r0 = result in proper unit)
    MoveB    r8L,rlL          ; rlL - 1/20ths
    lsr      rlL              ; and convert to 1/10ths (rounded)
    rts                       ; exit
```

Kernal_CRC:

```
;**********************************************************************
;   This is the actual Kernal Code for CRC.
;
;   pass:    r0 - pointer to start of data
;            r1 - # of bytes to check
;
;   return:  r2 - CRC Checksum
;
;   destroys: a, x, y, r0, r1, r3L
;**********************************************************************

Kernal_CRC:
            ldy   #$FF
            sty   r2L
            STY   r2H
            iny
10$
            lda   #$80
            sta   r3L
20$
            asl   r2L
            rol   r2H
            lda   (r0),y
            and   r3L
            bcc   30$
            eor   r3L
30$
            beq   40$
            lda   r2L
            eor   #%00100001
            sta   r2L
            lda   r2H
            eor   #%00010000
            sta   r2H
40$
            lsr   r3L
            bcc   20$
            iny
            bne   50$
            inc   r0H
50$
            ldx   #r1
            jsr   Ddec
            lda   r1L
            ora   r1H
            bne   10$
            rts
```

DecCounter:

```
;****************************************************
;

zCounter     = $70
COUNT        = $FFF0

DecCounter:
             LoadW   zCounter,COUNT
   10$
             Jsr     DoSomething
             ldx     #zCounter
             jsr     Ddec
             bne     10$
             rts
```

DdecvsDecW:

Size in Bytes vs Speed in Cycles of **Ddec** and **DecW**

Ddec represents a maximum of 7 byte savings over **DecW** every time it is used in your code. If Not needing a zero result after **DecW** then only a 3 byte savings.

DecW takes roughly ½ the time to execute. In and Inner loop executed 1 Million times. **DecW** will save roughly 20 seconds off the time vs **Ddec**

```
zCounter=$70
.macro DecW  dest
    lda      dest
    bne      dolow
    dec      dest+1
dolow:
    dec      dest
.endm
```

Ddec code block.

```
Op Code         Instruction           Bytes    Cycles
---------       ---------------       ------   --------
A2  70          ldx #zCounter         2        2
20  0E C2       jsr Ddec              3        6
                (Kernal Routine)      0        27 - 32
-----------------------------------------------------------
                Total                 5        35 - 40
```

DecW macro code block.

```
Op Code         Instruction           Bytes    Cycles
---------       ---------------       ------   --------
A9  70          lda zCounter          2        3
D0  02          bne 10$               2        2 or 3 or 4
C6  71          dec zCounter+1        2        5
            10$
C6  70          dec zCounter          2        5
-----------------------------------------------------------
                Total                 8        11 Worst Case 15
                if branch crosses page         12

    ;-- When using DecW on a counter, Add check for word=0 after the DecW macro
A9  70          lda zCounter          2        2
05  70          ora zCounter+1        2        3
-----------------------------------------------------------
                Total                 12       16 - 20
```

```
Kernal Ddec     ;Actual Kernal Code for Ddec
Op Code         Instruction                    Cycles
---------       ---------------       ------   --------
B5 00           lda zpage,X                    4
D0 02           bne 10$               (1/256ish chance 2) or 3 or Worst case:4
D6 01           dec zpage+1,X                  6
            10$
D6 01           dec zpage,X                    6
B5 00           lda zpage,X                    4
D6 01           ora zpage+1,X                  4
60              rts                            6
                                      =====================
                    Total             Best Case: 27          Worst Case:32
            if branch crosses Page               28          (1/256 chance)
```

DSmult:

```
;****************************************************
;   DSMult - double-precision signed multiply.
;
;   pass:    x - zpage address of multiplicand
;            y - zpage address of multiplier
;
;   returns: signed result in address pointed to by x
;            word pointed to by y is absolute-value of the
;                   multiplier passed
;            x, y unchanged
;
;   Strategy:
;            Establish the sign of the result: if the signs of the
;            multiplicand and the multiplier are different, then the result
;            is negative; otherwise, the result is positive. Make both the
;            multiplicand and the multiplier positive, do unsigned
;            multiplication on those, then adjust the sign of the result
;            to reflect the signs of the original numbers.
;
;   destroys:       a, r6 - r8
;....................................................................

DSmult:
    lda      zpage+1,x         ;get sign of multiplicand (hi-byte)
    eor      zpage+1,y         ;and compare with sign of multiplier
    php                        ;save the result for when we come back
    jsr      Dabs              ;multiplicand = abs(multiplicand)
    stx      r6L               ;save multiplicand index
    tya                        ;put multiplier index into x
    tax                        ;for call to Dabs
    jsr      Dabs              ;multiplier = abs(multiplier)
    ldx      r6L               ;restore multiplier index
    jsr      DMult             ;do multiplication as if unsigned
    plp                        ;get back sign of result
    bpl      90$               ;ignore sign-change if result positive
    jsr      Dnegate           ;otherwise, make the result negative
90$
    rts
```

memory

CopyBuffer

CopyBuffer:

```
SrcBuff: .byte "Any Values can be in the buffer",NULL,CR
         .byte $0C,”NULLS are just zeros here”,CR
LENBUFF = (*-SrcBuff)

.ramsect
      DestBuff .block LENSTRING

•psect

CopyBuffer:
      LoadW r5,#SrcBuff       ; point to start of source buffer
      LoadW rll,#DestBuff     ; point to start of destination buffer
      ldx   #r5               ; x <- source register address
      ldy   #rll              ; y <- destination register address
      lda   #LENBUFF          ; a <- length of buffer
      jsr   CopyFString       ; DestBuff <- SrcBuff (copy)
      rts



SrcStr: .byte "Any values but null can be in the string",NULL
LENSTRING = (*-SrcStr)

.ramsect
      DestBuff .block LENSTRING

.psect


CopyStr:
      LoadW r0,#SrcStr        ; point to start of source String
      LoadW rl,#DestBuff      ; point to start of destination buffer
      ldx   #r0               ; x <- source register address
      ldy   #rl               ; y <- destination register address
      jsr   CopyString        ; DestBuff <- SrcStr (copy)
      rts
```

memory

Find:

```
REC__SIZE = 5 ;size of each record
.ramsect
      Data: .block      1024  ;Table of Zip Code Locations.

.psect
      Key:  .byte “65803”     ;Zip Code to Find

Find:
      LoadW r2,#NUM_RECS      ; r2 <- total number of records
      LoadW r0,#Key           ; r0 <- pointer to keyword
      LoadW rl,#Data          ; rl <- pointer to start of search list
10$                           ; DO
      ldx   #r0               ;     x <- source string - key ,
      ldy   #rl               ;     y <- destination string - list
      lda   #REC_SIZE         ;     a <- length of each record
      jsr   CmpFString        ;     compare key with current record
      beq   20$               ;     if they match, branch to handler
      AddVW #REC_SIZE,rl      ;     otherwise point to the next record
      DecW  r2                ;     r2— (decrement counter)
      bne   10$               ; WHILE (r2 > 0)
      ;---
      jmp   NotMatched        ; jmp to no match handler
20$   jmp   Matched           ; jmp to match handler
```

Find2:

```
Find2:
      LoadW r0,#original      ; r0 <- pointer to original string
      LoadW r1,#copy          ; r1 <- pointer to copy
      ldx   #r0               ; x  <- source string =* key
      ldy   #r1               ; y  <- destination string - list
      jsr   CmpString         ;
      beq   20$
      jmp   NotMatched        ; jmp to no match handler
20$   jmp   Matched           ; jmp to match handler


original:
      .byte "Mark Charles Heartless",NULL

Copy:
      .byte "Mark Charlie Heartless",NULL
```

InitBuffers:

```
;*****************************************************
; initialize buffers and variables to zero
InitBuffers:
    LoadW    r0,#varStart              ; clear variable space
    LoadW    r1,#(varEnd-varStart)
    jsr      ClearRam
    LoadW    r0,#heapStart             ; clear heap
    LoadW    r1,#(heapEnd-heapStart)
    jmp      ClearRam


;*****************************************************
Alternate version. Using more space efficient i_FillRam
InitBuffers:
    jsr      i_FillRam                 ; clear variable space
    .word    varStart
    .word    varEnd-varStart
    .byte    $AA                       ; With any value you choose

    jsr      i_FillRam                 ; clear heap
    .word    heapStart
    .word    heapEnd-heapStart
    .byte    $00                       ; Heap set to zero's
    rts
```

disk

CheckDiskSpace

CheckDiskSpace:

```
;*****************************************************
;DESCRIPTION: Ensures that the current disk has a enough space for a
;             minimum number of bytes. Does not take into account any
;             index blocks or other blocks needed to maintain the file
;             structure. Works with GEOS 64, GEOS 128
;
;Pass:        r2    number of bytes we need
;Returns:     x =   If not enough space, returns an
;                   INSUFFICIENT_SPACE error.
;             x =   0 Is there is enough space.
;             Z Flag follows value of X.
;
;Destroyed: a, y, r2, r3, r8, r9
;*****************************************************

; Number of bytes that can be stored in each block on the disk. Accounts for
; two-byte track/sector link on Commodore versions of GEOS.

NO_ERROR     = 0
BLOCK_SIZE   = $100
BLOCK_BYTES  = BLOCK_SIZE - 2

.macro bgt raddr
   beq      label
   bcs      raddr
label:
.endm

CheckDiskSpace:
   lda    r2L                  ; r2 - # of BYTES to check for
   ora    r2H                  ; check if zero bytes requested
   beq    80$                  ; if so, exit with no error
   LoadW  r3,BLOCK_BYTES       ; r3 <- number of bytes per block.
   ldx    #r2                  ; divide r2 by r3 to get number of
   ldy    #r3                  ; blocks to hold BYTES
   jsr    Ddiv                 ; r2 <- r3/r2
   lda    r8L                  ; r8L <- remainder
   ora    r8H                  ; Any remainder bytes?
   beq    10$                  ; if not, OK
   IncW   r2                   ; otherwise 1 more block needed
                               ; r2 = BLOCKS needed to hold BYTES
10$                            ; get number of free blocks on disk
   LoadW  r5,#curDirHead       ; point to directory header
   jsr    CalcBlksFree         ; r4 <- free blocks on disk
   CmpW   r2,r4                ; are there enough free blocks?
   bgt    99$                  ; if not, assume. correct, branch.
90$
   ldx    #NO_ERROR            ; otherwise, no error
   rts
99$
   ldx    #INSUFFICIENT_SPACE ; not enough space
   rts                         ; exit
```

DeleteDirEntry:

```
;Pass: r0 pointer to filename
.ramsect
    rFileName:                  .block 17

DeleteDirEntry:
    LoadW   r0,rFileName
    LoadW   r3,#NullTrScTable   ;pass dummy table
    jmp     FastDelFile
```

This will also work correctly with a VLIR file. For freeing (deleting) all the blocks in a file without removing the directory entry refer to **FreeFile**.

ReadAndDelete:

```
                                                                    .if COMMENT
  Read sequential file into memory and then delete it from disk
  Pass:     r6 pointer to filename
            r7 where to put data
            r2 size of buffer (max size of file)

  Returns: x error code

  Destroys: a, y, rO-r9

  Implementation:
       Call FindFile to get the directory entry of the  file to load/delete. We pass
       the directory entry  to GetFHdrlnfo to get the GEOS header block. We  check
       the header to ensure we're not trying to read in a VLIR file. After
       GetFHdrlnfo, the parameters are already set up correctly to call ReadFile
       (fileTrScTab+O,fileTrScTab+1 contains  header block and rl contains first
       data block). ReadFile reads in the file's blocks, building out  the remainder
       of the fileTrScTab, which we pass to FastDelFile to free all blocks in the
       file  (including the file header block, which is the  first entry in the
       table).
                                                                    .endif

ReadAndDelete:
    MoveW   r6,r0                           ; save pointer for FastDelFile
    Jsr     FindFile                        ; find file on disk
    txa                                     ; set status flags
    bne     99$                             ; branch on error
    LoadW   r9,dirEntryBuf                  ; get directory entry
    jsr     GetFHdrlnfo                     ; get GEOS file header
    txa                                     ; set status flags
    bne     99$                             ; branch on error
    lda     fileHeader+OFF_GSTRUCT_TYPE     ;
    cmp     #VLIR                           ; check filetype
    bne     10$                             ; branch if not VLIR
    ldx     #STRUCT_MISMAT                  ; can't load VLIR
    bne     99$                             ; branch always for error
10$
    jsr     ReadFile                        ; read in file
    txa                                     ; else set status flags
    bne     99$                             ; branch on other error
20$
    LoadW   r3,#fileTrScTab                 ; track/sector table
    jsr     FastDelFile                     ; file read OK, delete it!
99$
    rts                                     ; error in x
```

GrabSomeBlocks:

```
;****************************************************
;
;   GrabSomeBlocks — allocate enough disk blocks to hold
;                    data in buffer.
;
;   pass:    Nothing
;
;   returns: Carry flag. 1 = Error, 0 = success.
;            X = Error Nbr if Carry is set or 0.
;
;****************************************************

    K = 1024                  ;one kilobyte

.ramsect
    buffer:  .block 5*K -1    ; 5K buffer .'
    bufferE: .block 1         ; End of 5k Buffer

    BUF_SIZE = (bufferE - buffer)+l ; size of buffer
.psect

GrabSomeBlocks:
    LoadW    r2,BUF_SIZE      ; number of bytes to allocate
    LoadW    r6,fileTrSecTab  ; buffer to build out table
    jsr      BlkAlloc         ; allocate the blocks
    txa                       ; check status
    bne      99$              ; and exit on error
    ; more code here
90$
    ldx      #0
    clc                       ; Success exit
    rts
99$
    sec                       ; Error Exit
    rts
```

MyFreeBlock:

```
;*****************************************************
;
;   MyFreeBlock — allocate specific block in BAM
;   with any CBM device driver. And any GEOS Version
;
;   pass:
;            r6L = track #
;            r6h = sector #
;
;   Note:
;            FreeBlock was not added to the
;            GEOS jump table until v1.3
;*****************************************************

MyFreeBlock:
    lda      version           ;check GEOS version number
    cmp      #$13              ; version Less then 1.3?
    bcc      10$               ;
    Jmp      FreeBlock         ; if not, go through jump table

10$
    jsr      FindBAMBit        ; Returns   r8H = mask for BAM byte
                               ;           r7H = offset to track
                               ;           x = offset into bam
                               ;           a = masked value
    bne      99$               ; if 1, then not allocated, give error
    txa
    bne      99$
    lda      r8H               ; get mask
    eor      curDirHead,x      ; flip BAM bit to make available
    sta      curDirHead,x      ;
    ldx      r7H               ; one more free block
    inc      curDirHead,x      ;
    ldx      #0                ; NO_ERROR
    rts

99$
    ldx      #BAD_BAM          ;
    rts
```

MySetGDirEntry:

```
                                                        .if COMMENT
This routine duplicates the function of the Kernal's SetGDirEntry for demonstration
purposes. It shows examples of the following routines:

    BldGDirEntry
    GetFreeDirBlk
    PutBlock

Pass:        Same as SetGDirEntry
Destroys:    Same as SetGDirEntry

                                                        .endif
DIRCOPYSIZE=30                    ; Size of directory entry for copy
TDSIZE=5                          ; number of bytes in time/date entry

MySetGDirEntry:
    jsr      BldGDirEntry         ; build directory entry for GEOS file
    jsr      GetFreeDirBlk        ; get block with free directory entry
                                  ; r3 = 1st byte of free entry
                                  ; block number of block in rl
    txa                           ; test for error code
    bne      99$                  ; if error, exit...
    tya                           ; get offset into diskBlkBuf for dir entry
    clc
    adc      #[diskBlkBuf         ; and get absolute address in buffer
    sta      r5L
    lda      #]diskBlkBuf
    adc      #0                   ; (propagate carry)
    sta      r5H
    ldy      #DIRCOPYSIZE         ; copy over some bytes
10$
    lda      dirEntryBuf,y        ; get byte from directory entry built
    sta      (r5),y               ; store new entry into block buffer
    dey
    bpl      10$                  ; loop till copied
    jsr      TimeStampEntry       ; stamp the dir entry with time & date
    LoadW    r4,#diskBlkBuf       ; write out the new directory entry
    jsr      PutBlock
    txa                           ; get error status
    bne      99$                  ; if error, exit
    clc
    rts                           ; Success exit
99$
    sec
    rts                           ; Error exit

TimeStampEntry:
    ldy      #(OFF_YEAR+TDSIZE)-1         ; offset to time/date stamp
10$
    lda      dirEntryBuf,y        ; get the year/month/day/hour/minute
    sta      (r5),y               ; store in dir entry
    dey                           ;
    bpl      10$                  ; Loop until done
    rts
```

MyPutBlock:

```
;****************************************************
;   MyPutBlock - Write diskBlkBuf to disk
;
;   pass:
;           r1L = track #
;           r1H = sector #
;           r4  = Address of block to write.
;           verify =  FALSE (0) Do Not Verify
;                   <> 0 Verify after Write
;
;   Note:   If you have multiple blocks to write you should
;           write the entire chain and then verify the chain.
;           See WriteBlock description for more information
;****************************************************
    .ramsect
            nextTrack:      .block      1
            nextSector:     .block      1
            outbuffer:      .block      $FE
            track           .block      1
            sector          .block      1
            verify:         .block      1
    .psect

CallMyPutB:
    LoadW   r4,outBuffer-2
    MoveB   track,r1L,
    MoveB   sector,r1H
    LoadB   verify,[#TRUE
    jsr     MyPutBlock
    bcs     99$
    rts                     ;return good status in carry
99$
    ...                     ;Error Handler or let caller handle error
    rts

MyPutBlock:
    jsr     EnterTurbo      ; go into turbo mode
    txa                     ; check for error in X
    bne     99$             ; branch if error found
    jsr     InitForIO       ; prepare for serial I/O
    jsr     WriteBlock      ; primitive write block
    txa                     ; set status flags
    bne     99$             ; branch if error found
    lda     verify          ; check verify flag
    beq     80$             ; branch if not verifying
    jsr     VerWriteBlock   ; verify block we wrote
    txa                     ; set status flags
    bne     99$             ; branch if error found
80$
    jsr     DoneWithIO      ; restore after I/O done
    clc
    rts                     ; No Errors
99$
    jsr     DoneWithIO      ; restore after I/O done
    sec
    rts                     ; Error Status exit
```

MyReadBlock:

```
;****************************************************
;
;   MyReadBlock — Read sector from disk into diskBlkBuf
;   Demonstrates use of very-low level disk primitives
;   pass:
;            r1L = track #
;            r1H = sector #
;            r4  = Address of block to read into.
;
;****************************************************
.ramsect
             nextTrack:       .block      1
             nextSector:      .block      1
             outbuffer:       .block      $FE
             inbuffer         .block      $100
             track            .block      1
             sector           .block      1
    verify:          .block    1

    .psect

CallMyPutB:
    LoadW    r4,inBuffer
    MoveB    track,r1L,
    MoveB    sector,r1H
    jsr      MyReadBlock
    bcs      99$
    rts                       ;return good status in carry
99$
    ...                       ;Error Handler or let caller handle error
    rts

MyReadBlock:
    jsr      EnterTurbo       ; go into turbo mode
    txa                       ; check for error in X
    bne      99$              ; branch if error found
    jsr      InitForIO        ; prepare for serial I/O
    jsr      ReadBlock        ; primitive read block
    jsr      DoneWithIO       ; restore after I/O done (x is preserved in DoneWithIO)
    txa                       ; get error result of ReadBlock
    bne      99$              ; branch if error found
80$
    clc
    rts
99$
    sec
    rts
```

NewAllocateBlock:

```
;****************************************************
; NewAllocateBlock — allocate specific block in BAM
; with any CBM GEOS device driver.
;
; Pass: r6L,r6H track, sector to allocate
;
; Uses: BAM in curDirHead
;
; Returns: x  error status ($00 = success, BAD_BAM = block already in use, etc.)
;
; Destroys: a,y,r7, r8H.
;****************************************************
BAD_BAM=$0B
DRV_1571=2
NO_ERROR=0

NewAllocateBlock:
    ldy      curDrive           ; get current drive
    lda      driveType-8,y      ; get drive type
    and      #%00001111         ; keep only drive format
    cmp      #DRV_1571          ; see if 1571 or above
    bcc      1541$              ; branch if 1541
    jmp      AllocateBlock      ; else, use driver routine
1541$
    jsr      FindBAMBit         ; get BAM bit info
    beq      110$               ; if zero, then it's not free
    lda      r8H                ; get bit mask for BAM
    eor      #$FF               ; convert to clearing mask
    and      curDirHead,x       ; and with BAM byte to clear
                                ; bit and show as allocated
    sta      curDirHead, x      ; and store back.
    ldx      r7H                ; get base of track9s entry
    dec      curDirHead,x       ; dec # free blocks this track
    ldx      #NO_ERROR          ; show no error
    rts                         ; exit
99$
    ldx      #BAD_BAM           ; show error — already in use
    rts                         ; exit

Example Caller Routine;
.ramsect
    diskBlock .block 2
.psect

CallNewAlloc:
    MoveW    diskBlock,r6       ; block to allocate
    jsr      NewAllocateBlock   ; (see above)
    cpx      #BAD_BAM           ; BAD_BAM means block in use
    beq      95$                ; branch if block already in use
    txa                         ; check for other error
    bne      99$                ; branch if error
             ; code to handle newly allocated block goes here
95$                             ; block was not free...
             ; code to handle block already allocated goes here
99$
    jmp      MyDiskError        ; call error handler with error in x
```

SaveRecord:

```
;*****************************************************
;
;   SaveRecord — Append new record into am existing VLIR
;
;   pass:    appendPoint = Already set to the last VLIR Record
;            Filename    = Buffer populated with VLIR's filename
;
;*****************************************************
NAME_LENGTH=17

.ramsect
      appendPoint:      .block 1               ; record to append to
      filename:         .block NAME_LENGTH     ; hold null-terminated filename
      bufStart:         .block 1023            ; data buffer
      bufEnd:           .block 1               ; length of buffer
      BUFLENGTH         = (BufEnd - BufStart)+1

.psect

SaveRecord:
      LoadW r0, #filename     ; pointer to filename
      jsr   OpenRecordFile    ; open VLIR file
      txa                     ; check open status
      bne                     ; exit on error
      lda   appendPoint       ; get record to append to
      jsr   PointRecord       ; go to that record
      txa                     ; check point status
      bne   99$               ; exit on error
      jsr   AppendRecord      ; append a record at this point
      LoadW r7, #bufStart     ; point at data buffer
      LoadW r2, #BUFLENGTH    ; bytes in buffer (bufEnd-bufStart)
      jsr   WriteRecord       ; write buffer to record
      txa                     ; get write status
      bne   99$               ; exit on error
      jsr   CloseRecordFile   ; close VLIR file
      txa                     ; check point status
      bne   99$               ; exit on error
90$                           ; Clean Exit
      clc                     ; clear carry for all ok
      rts
99$:                          ; Error handler
      sec                     ; Set carry to show returning with an error
      rts
```

NOTE: geoProgrammer1.1 does not support the * counter in .ramsect. The method above must be used when the assembler needs to calculate the size of a ramsect field.

internal

internal

internal

FatalError:

```
                                                        .if COMMENT
Purpose: use Panic to send a fatal error message to the user
Pass:           r0
                                                             .endif
.ramsect
   GEOS_save .block BYTESTOSAVE ; save area for GEOS restart block

.psect

FatalError:
      IncW  r0    ; add 2 to error number
      IncW  r0    ; to compensate for Panic
.if C64
      PushW r0    ; push error number onto stack

.else             ; 128, expects all kinds of internal
      ; machine-state information (10 bytes total) on the
      ; stack. It ignores all but the bottommost word.
      ldx   #5-1  ; place 5 words (10 bytes) total onto stack
$10
      PushW r0    ; push error number onto stack
      dex         ; (use error number repeatedly as dummy value)
      bne   10$   ; loop until all done.
.endif
      jmp   Panic ; go put up the Panic dialog box
```

Alternate Version with live detection of 64/128 and a more efficient setting of the stack pointer.

```
FatalError:
      IncW  r0    ; add 2 to error number
      IncW  r0    ; to compensate for Panic
      bit   c128Flag
      bpl   10$   ; C64. Just push once.
                  ; 128, expects all kinds of internal
                  ; machine-state information (10 bytes total) on the
                  ; stack. It ignores all but the bottommost word.
      tsx         ; Set Stack Pointer down 8 bytes to prepare for r0
      txa         ; push for the last word
      sec
      sbc   #8
      txs         ; Save the new stack pointer
10$               ; Now put final Word onto stack
      PushW r0    ; push error number onto stack
      jmp   Panic ; go put up the Panic dialog box
```

internal

RoadTrip:

```
;Show Leaving GEOS to use all of the resources of the machine and returning again via
;rebooting by either REU or disk.

BYTESTOSAVE=$80                 ; no. of bytes to save at BootGeos
RBOOT_BIT=5                     ; bit in sysFlgCopy to check
CKRNL_BAS_IO_IN=$40
config=$FF00

.ramsect
    GEOS_save .block BYTESTOSAVE ; save area for GEOS restart block

.psect

RoadTrip:
    jsr  OnEntry                ; Save Kernal Boot strap
    jsr  HaveAFunTrip           ; Do anything ... Use all of kernal ram
                                ; just no kernal calls while you are gone
    jmp  OnExit                 ; Reboot the Kernal

OnEntry:
    ldx  #BYTESTOSAVE           ; save bytes GEOS needs so we can use area
10$                             ; STARTLOOP
    lda  BootGeos-1,x           ;   copy a byte
    sta  GEOS_save -l,x         ;
    dex                         ;    count--
    bne  10$                    ; if (count != 0), then loop
    rts                         ; ENDLOOP

OnExit:
    lda  version                ; Get. version of GEOS
    cmp  #$13                   ;
    bcc  64$                    ; If version < 1.3, then branch
    lda  cl28Flag               ; else, test for GEOS 128
    bpl  64$                    ; If GEOS64, then branch
128$                            ;
    lda  CKRNL_BAS_IO_IN        ; load 128 memory mapping
    sta  config                 ;
    bra  200$                   ;
64$                             ;
    lda  #KRNL_BAS_IO_IN        ; load 64 memory mapping
    sta  CPU_DATA               ;
$200
    ldx  #BYTESTOSAVE           ; restore bytes GEOS needs to restart
10$                             ; STARTLOOP
    lda  GEOS_save-l,x          ;   copy a byte
    sta  BootGeos-l,x           ;
    dex                         ;    count--
    bne  10$                    ; if (count != 0), then loop
                                ;ENDLOOP
    lda  #(%1<<RBOOT_BIT)       ; check for Rboot flag
    and  sysFlgCopy             ;
    bne  90$                    ; if flag is clear, branch to rboot
    jsr  AskForBootDisk         ; else, get user to insert boot disk
90$
    jmp  BootGeos
```

internal

graphics

ChangeMode

BitCompact:

```
.if (0)
*************************************************************************************
BitCompact
*************************************************************************************

DESCRIPTION:
      Converts linear bitmap data into compacted bitmap format, suitable for
      passing to routines such as BitmapUp.

      When compacting bitmaps directly from screen memory, the data must tsx first
      be converted from the internal screen format to linear bitmap format. The
      left edge of the source bitmap must start on a card boundary and the right
      edge must extend to the end of another, card boundary.

      This bitmap data must then be converted to a linear format, where the first
      byte represents the first eight pixels of the upper-left corner of the
      bitmap, the next byte represents the next eight pixels and so on to the right
      edge of the bitmap. The byte following the last byte in a single line of a
      bitmap is the first byte of the next line. (The actual dimensions of the
      bitmap will be reconstructed from the WIDTH and HEIGHT parameters passed to
      the bitmap display routine.

      To convert from internal screen format to linear bitmap format:

      C64: Set dispBufferOn appropriately (to reflect which
      screen buffer to grab data from) and...

      Cnvrt40:
            ldx   yPos              ; get y coord of top of bitmap
            jsr   GetScanLine       ; use it to calc screen ptrs
            lda   xPos              ; get x pixel coord (lo byte)
            and   #%11111000        ; strip off 3 bits for card x-position
            clc                     ; Add card offset to
            adc   r5L               ; base pointer (lo byte first)
            sta   r5L
            lda   xPos+1            ; (hi byte also)
            adc   r5H
            sta   r5H
            ;At this point, (r5) points to the first byte in
            ;the bitmap (upper-left corner).

            Now step through each byte in this scanline by adding 8 to the pointer
            in r5 (compensating for the card architecture) to get to the next byte,
            and repeat this process for each line in the bitmap (incrementing yPos
            appropriately for each scanline).

      C128: (40-column, same as C64; 80-column, read on...)
            Conveniently, the 80-column data is already in linear bitmap format.
            The data, will probably be coming from the background buffer because
            the foreground screen is entirely contained on the VDC chip's internal
            RAM and is difficult to access...
```

```
Cnvrt80:
      bit   graphMode         ; make sure in 80-col mode
      bpl   Cnvrt40           ; handle 40 like C64
      PushB dispBufferOn      ; save current dispBufferOn
      LoadB dispBufferOn,#ST_WR_BACK ;force use of back buffer
      ldx   yPos              ; get y coordinate
      jsr   GetScanLine       ; use it to calc screen ptrs
      MoveW xPos,r0           ; copy x-position to zp work reg
      ldx   #r0               ; divide r0 by 8
      ldy   #3                ; (shift right 3 times)
      jsr   DShiftRight       ; this gives us the card offset
      AddW  r0,r6             ; add card (byte) offset to scanline addr

      ;At this point (r6) points to the first byte of the bitmap.
      Now step each byte in this scanline by adding
      1 to the pointer in r6 to get to the next byte,
      and repeat this process for each line in the
      bitmap (incrementing yPos appropriately).
```

CALLED BY:

PASSED:

r0 Pointer to destination buffer to store compacted data (this buffer must be at least 1 and 1/64 of size of the uncompacted data because it is possible, but unlikely, that the compacted data will actually be larger than the uncompacted data).

r1 Pointer to linear bitmap data to compact.

r2 # if bytes to compact

RETURNS:

r0 Points to byte following last byte in compacted data.

DESTROYED: a,x,y,**rl-r6**

PSEUDO CODE / STRATEGY:

Starts with the first source byte and counts the number of identical bytes following it to determine whether to generate a UNIQUE or REPEAT packet. If there are three or less identical bytes in a row, a UNIQUE packet is generated, four or more generates a REPEAT packet. The packet is placed in the destination buffer and this process is then repeated until all bytes in the source buffer have been compressed.

KNOWN BUGS / SIDE EFFECTS / IDEAS:

Only uses the UNIQUE and REPEAT compaction types. The BIGCOUNT compaction type is such that it is difficult to determine the compaction payoff point. BIGCOUNT could be used to compress adjacent scanlines that are identical because this type of check would be trivial. The basic scanline could be compressed with UNIQUE and REPEAT, then duplicated by placing it inside a BIGCOUNT.

This routine is not limited to compressing bitmap data. In fact, it works quit well on any data where strings of identical bytes are common (e.g., fonts). It does not, for example, compress text very efficiently. A Huffman-based algorithm yields better results.

```
*******************************************************************************
.endif
```

```
MAX_REPEAT  =     127          ; maximum repeat COUNT value
MAX_UNIQUE  =     191          ; maximum unique COUNT value
UNIQ_THRESH =     3            ; byte count threshold, beyond which a REPEAT type
                               ; should be used instead of UNIQUE.

BitCompact:
10$                            ; rl = current addr in source buffer
                               ; r0 = current addr in destination buffer
                               ; r2 = # bytes left in source
      jsr   CountRepeat        ; count the # of identical bytes here
      cmp   #UNIQ_THRESH       ; Enough repeats to justify REPEAT type?
      ble   20$                ; No, go use UNIQUE
                               ; yes, use REPEAT (A = # to repeat)
      sta   r5L                ; store repeat # for later
      ldy   #0                 ; init. index into buffers
      sta   (r0),y             ; store repeat # to destination
      lda   (rl),y             ; get repeat value
      iny                      ; point to next byte in dest buffer
      sta   (r0),y             ; store to destination buffer
      AddVW 2,r0               ; move up dest. pointer
      bra   100$               ; exit.
20$
                               ; use UNIQUE
      jsr   GetUnique          ; Calc # of unique bytes to use
                               ; (A = number of unique)
      ldy   #0                 ; init. index into buffers.
      ora   #$80               ; convert unique count to packet count value
      sta   (r0),y             ; store to dest buffer
30$
      lda   (rl),y             ; get first unique value
      iny                      ; increment pointer
      sta   (r0),y             ; store to destination buffer
      cpy   r5L                ; done yet? (r5L - repeat #)
      bne   30$                ; loop till done copying
      inc   r5L                ; convert to # to add to dest pointer
      AddBW r5L,r0             ; move up destination pointer
      dec   r5L                ; correct back to # done
                               ; fall through to-exit
100$                           ;
      AddBW r5L,rl             ; move up source pointer
      SubBW r5L,r2             ; subtract off # left in source buffer
      lda   r2L                ; check for zero bytes left
      ora   r2H                ; more to do?
      bne   10$                ; if so, loop
      rts                      ; else, exit.

CountRepeat:
                               ; rl = current pointer into source buffer
                               ; r0 = current pointer into destination buffer
                               ; r2 = number of bytes left in source
      ldy   #0                 ; initialize relative buffer index
      ldx   #0                 ; initialize current repeat count
                               ;
```

```
      lda    (rl),y            ; get first byte
      sta    r6L               ; keep in r6L. This is the byte we're trying
                               ; to match.
10$                            ;
      lda    r2H               ; more than 255 bytes left in source?
      bne    20$               ; if so, ignore # check
      cpx    r2L               ; else, are we at the last byte?
      beq    90$               ; if so, exit
20$                            ;
      cpx    #MAX__REPEAT      ; check repeat count with max # of repeats
      beq    90$               ; if at maximum, branch to exit.
      lda    (rl),y            ; does it actually match?
      cmp    r6L               ; check against 1st byte
      bne    90$               ; if no match, exit.
      inx                      ; else, we found a match, increment repeat count
      iny                      ; move to next byte in source
;NOTE -- following branch changed to save a byte, y is never incremented to $00.
;     bra    10$               ; and loop to check it
      bne    10$               ; branch always... iny above will always clear z flag
90$
      txa                      ; return repeat count in A
      rts                      ; exit

GetUnique:
      PushW rl                 ; Save orig pointer
      LoadB r5L,#0             ; start none unique
10$
      inc    r5L               ; do one more unique
      ldx    r5L               ; get # unique so far
      lda    r2H               ; lots left?
      bne    20$               ; if so, skip end check
20$
      cpx    r2L               ; all of them?
      beq    90$               ; if yes, then that many
      cpx    #MAX UNIQUE       ; max # unique
      beq    90$               ; if full, do them
      AddVW #1,rl              ; move up a byte
      jsr    CountRepeat       ; how many of the following bytes are repeats?
      cmp    #UNIQ_THRESH      ; Enough to warrant a REPEAT packet?
      ble    10$               ; No, go stuff them in this UNIQUE packet
30$                            ; Yes, close this UNIQUE packet.
      PopW   rl                ; retrieve start pointer
      lda    r5L               ; get # to do unique
      rts
```

ChangeMode:

```
GREYPAT=2

ChangeMode:
       jsr   GreyScreen                ; grey out old screen
       lda   graphMode                 ; switch mode by flipping
       eor   #%10000000                ; 40/80 bit
       jsr   SetNewMode                ; and calling SetNewMode
       jsr   GreyScreen                ; grey out new screen
       rts                             ; exit


GreyScreen:
     jsr     i_GraphicsString          ;
     .byte   NEWPATTERN,GREYPAT        ; set to grey pattern
     .byte   MOVEPENTO                 ;Put pen in upper left
     .word   0                         ; x
     .byte   0                         ; y
     .byte   RECTANGLETO               ; grey out entire screen
     .word   (SC_PIX_WIDTH-1) | DOUBLE_W | ADD1_W
     .byte   SC_PIX_HEIGHT-1
     .byte   NULL
     rts
```

Check128:

```
                                                          .if COMMENT
*****************************************************
Check for GEOS 128.

Pass:
nothing

Returns:
    st minus flag set if running under GEOS 128.
*****************************************************
                                                  .endif
Check128:
    lda      #$12        ; cl28Flag not valid until version 1.3
    cmp      version     ; first see if version <= 1.2
    bpl      10$         ; if so; branch and say C64.
    lda      c128Flag    ; else set minus based on high bit cl28Flag.
10$
    rts

Example usage:
    jsr Check128
    bpl 10$              ;ignore if under GEOS 64
    jsr DoDeDoubling     ;else, patch x-coordinates to remove doubling bits
10$
    .
    .
    .
```

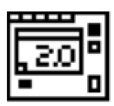
DblDemo1:

```
;Will assemble differently depending on the status of the C64 and C128 assembly
;constants. If assembling for GEOS 64, doubling constants will be set to zero so
;that they will not affect the x-positions. If assembling for GEOS 128, doubling
;constants will be set according to geosConstants file so that graphic operations
;will double automatically in 128 mode.

.if !(C128 ^^ C64)                  ; C64/C128 flags must be mutually exclusive!
      .echo "DblDemo not designed to assemble for both GEOS 64 and GEOS 128!"
.else
      .if    !C128                  ; if not assembling for GEOS 128, force
                                    ; doubling constants to harmless values so
                                    ; GEOS 64 graphics routines
                                    ; don't get confused.
            DBLE_B=0                ; Note3: geoAssembler1.x cannot do reassignment
            DBLE_W=0                ; Need a new equate to hold the conditional
            AD1_W =0                ; value.
      .else                         ;
            DBLE_B=DOUBLE_B         ; If this logic block was in the CONSTANTS
            DBLE_W=DOUBLE_W         ; file it could set DOUBLE_B,DOUBLE_W,ADD1_W as
            AD1_W=ADD1              ; needed and then all of the code base would
      .endif                        ; use those values.

BM_XPOS      = (32/8)               ; byte x-position of bitmap (40-col)
BM_YPOS      = 20                   ; y-position of bitmap

Bitmap:
```



```
BM_WIDTH = PicW                     ; byte bitmap width (40-col)
BM HEGHT = PicH                     ; byte bitmap width (40-col)

FPATTERN = %11111111                ; pattern for surrounding frame

DoBMap:
;Place the bitmap on the screen, loading the registers with
;inline data (note double-width settings).
      jsr    i_BitmapUp             ; inline call
       .word Bitmap                 ; bitmap address
       .byte (BM_XPOS|DBLE_B)       ; xpos
       .byte (BM_YPOS               ; yPos
       .byte (BM_WIDTH|DBLE_B)      ; width
       .byte BM_HEIGHT              ; height
90$
      rts /exit
      ;(both C128 & C64 constants were both true or both false)
.endif
```

DisplayImage:

```
                                                                        .if COMMENT
******************************************************
DisplayImage -- General purpose routine to display a portion of compacted bitmap
                image in a window

Pass:        pixBuf   compacted bitmap image in pseudo-photoscrap format. Byte 0 is
                      card width of image. Byte 1 and 2 is the pixel height (word).
                      The compacted image data starts at byte 3.
             xOffset  card index into bitmap to display
             yOffset  pixel index into bitmap to display

Destroys:   a,x, y, r0-rl2

******************************************************
                                                                             .endif
.ramsect
      xoffset:    .block 1          ; card x index into bitmap (byte)
      yoffset:    .block 2          ; pixel y index into bitmap (word)

      ;--- 2K picture buffer
      PixWidth:   .block 1          ; width of picture in cards (byte)
      PixHeight:  .block 2          ; height of picture in pixels (word)
      PixImage:   .block $800-3     ; start of bitmap image

.psect

WINDOW X       = 4                  ; card x-position of window
WINDOW_Y       = 30                 ; pixel y-position of window
WINDOW_WIDTH   = 5                  ; card width of window
WINDOW__HEIGHT = 60                 ; pixel height of window

DisplayImage:
      ;--- set up initial parameters
      LoadW    r0,#PixImage         ; r0 <- compacted picture data (DATA)
      LoadB    rlL,#WINDOW_X        ; rlL <- left edge of window (XPOS)
      LoadB    rlH,#WINDOW_Y        ; rlH <- top edge of window (Y)
      LoadB    r2L,#WINDOW_WIDTH    ; r2L <- width of window (W_WIDTH)
      LoadB    r2H,#WINDOW_HEIGHT   ; r2H <- height of window (W_HEIGHT)
      MoveB    xOffset,rllL         ; rllL <- x offset into bitmap (DX1)
      MoveW    yOffset,rl2          ; rl2 <- y offset into bitmap (DY1)

      ;--- clip x to window
      lda      PixWidth             ; get bitmap width
      sec                           ;
      sbc      #WINDOW_WIDTH        ;
      sbc      rllL                 ; now we have the right edge clip distance
      sta      rllH                 ; rllH <- right edge clip (DX2)
      bpl      10$                  ; if we're >0, branch to skip x clipping
      adc      #WINDOW WIDTH        ; add back the window width
      sta      r2L                  ; make that the new clip window
      LoadB    rllH,#0              ; rllH <- $00 (fixes underflow of DX2)
```

```
10$   ;--- clip y to window
      lda      PixHeight            ; subtract window height from bitmap height
      sec                           ; (two byte subtraction)
      sbc      #WINDOW_HEIGHT
      sta      r3L                  ; store intermediate result in r3
      lda      PixHeight+1
      sbc      #0
      sta      r3H

      lda      r3L                  ; now subtract y index into bitmap
      sec
      sbc      rl2L                 ; (rl2 = yOffset)
      sta      r3L
      lda      r3H
      sbc      r12H                 ; (rl2 = yOffset)
;     sta      r3H                  ; value in r3H never used after this
      bpl      20$                  ; branch if no underflow
      lda      r3L
      adc      #WINDOW HEIGHT       ; correct for underflow
      sta      r2H
20$   jsr      BitmapClip           ; display the bitmap with clipping
      rts                           ; exit
```

FilledRect:

```
X1    = 35   ; left edge
X2    = 301  ; right edge
Y1    = 40   ; top edge
Y2    = 100  ; bottom edge

FilledRect:
;--- Draw a filled rectangle using the current pattern
      jsr    i_Rectangle.         ;inline call
       .byte Y1,Y2                ;y1,y2
       .word (X1|DOUBLE_W|ADD1_W) ;x1 with doubled width + space on left for frame
       .word (X2|DOUBLE_W)        ;x2 with doubled width

      jsr    i_FrameRectangle
       .byte Y1,Y2                ;y1,y2
       .word (X1|DOUBLE_W)        ; x1 with doubled width
       .word (X2|DOUBLE_W|ADD1_W) ; x2 with doubled width + offset for frame
       .byte $FF
      rts


;--- Size Optimized Version.
;--- Saves 7 bytes over the original version of FilledRect
;--- While achieving the same result.

FilledRect:
      jsr    i_Rectangle.         ;inline call
       .byte Y1,Y2                ;y1,y2
       .word (X1|DOUBLE_W)        ;Fill Full Size of final Rectangle
       .word (X2|DOUBLE_W)        ;
                                  ;X (r3,r4) and Y (r2L,r2H) are set and returned
                                  ;unchanged by i_Rectangle

      lda    #$FF                 ;Set Line Pattern
      jmp    i_FrameRectangle     ;Frame Full Size of rectangle
```

MseToCardPos:

```
                                                                  .if COMMENT
*****************************************************

MseToCardPos
    converts current mouse positions to card position

    pass:            Nothing

    uses:            mouseXPos, mouseYPos

    Returns:         r0L       mouse card x-position (byte)
                     r0H       mouse card y-position (byte)

    Destroys:        a,x,y

*****************************************************
                                                                  .endif

MseToCardPos:
    php                        ; save current interrupt disable status
    sei                        ; disable interrupts so mouseXPos doesn't change*
    MoveW    mouseXPos, r0     ; copy mouse x-position to zp work reg (r0)
    lda      mouseYPos         ; get mouse y-position
    plp                        ; reset interrupt status asap.
    ldx      #r0               ; divide x-position (r0) by 8
    ldy      #3                ; (shift right 3 times)
    jsr      DShiftRight       ; this gives us the card x-position in r0L
    lsr      a                 ; shift y-position in a right 3 times
    lsr      a                 ; which is a divide by 8
    lsr      a                 ; and gives us the card y-position in a
    sta      r0H               ; set card y-position
    rts                        ; exit
```

Note: If you do not disable interrupts prior to getting the value of **mouseXPos** you could get **r0H** with Lydia/site and before getting really an interrupt occurs and the mouse position is updated during the interrupt. Now when you do/star for **r0L** it is for a different **mouseXPos** reading giving unpredictable results.

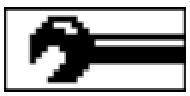
Note[3]: By also getting the Y value while interrupts are disabled, you are guaranteed to also get a consistent reading for all three parts of the mouse position.

ShowBitmap

```
;******************
;*** ShowBitmap ***
;******************
; For C64 and C128

.if C128
   DOUBLE_B=%10000000
.else
   DOUBLE_B=NULL
.endif

BM_XPOS     = (32/8)              ; card x-position of bitmap
BM_YPOS     = 20                  ; y-position of bitmap
                                  ;
Bitmap:
```



```
BM_WIDTH    = picW                ; card width of bitmap
BM_HEGHT    = picH                ; bitmap height
                                  ;
                                  ; Place the bitmap on the screen,
                                  ; loading the registers with
                                  ; inline data (note double-width
ShowBitmap:
    LoadB   dispBufferOn,#(ST_WR_FORE | ST_WR_BACK)

.if (C128)                        ; bug fix for 128 release 1. (Not needed for 2.0+)
    jsr     TempHideMouse         ; remove sprites
.endif

    jsr     i_BitmapUp            ; inline bitmap call
     .word  Bitmap                ;      *bitmap address
     .byte  BM_XPOS | DOUBLE_B    ;      *Xpos
     .byte  BM_YPOS               ;      *Ypos
     .byte  BM_WIDTH | DOUBLE_B   ;      *width
     .byte  BM HEIGHT             ;      *height

90$ rts                           ; exit
```

StopMenus:

```
                                                                .if COMMENT
Example of how to temporarily disable menus and then restart them at
a later time.
jsr StopMenus        will stop menu processing.
jsr RestartMenus     will return menu processing to it's prior state.
                                                                .endif
oldMouseOn:
    .byte   $00                    ; temp save area for mouseOn variable

StopMenus:
    MoveB   mouseOn,oldMouseOn     ; save current enable status for later
    rmbf    MENUON_BIT,mouseOn     ; disable menus temporarily
    rts

RestartMenus:
    lda     oldMouseOn             ; get old menu enable status
    and     #(%1 << MENUON_BIT)    ; ignore all but menu bit
    ora     mouseOn                ; restore old menu bit
    sta     mouseOn                ; in current mouseOn byte
    rts                            ; exit
```

Sta80Fore:

```
                                                                    .if COMMENT
Sta80Fore    — stores byte to 128 80-column foreground screen
Lda80Fore    — loads byte from 128 80-column foreground screen

Pass:        r5 = address in foreground memory
             A = data value (for Sta80Fore)
Returns:     A data value (for Lda80Fore)
Destroyed:   x
Note:        Call TempHideMouse to disable software sprites before accessing
             foreground screen directly.
                                                                    .endif
VDC_UAH=$12                    ; update hi-byte of VDC pointer
VDC_UAL=$13                    ; update lo-byte of VDC pointer
VDC_DA=$1F                     ; data byte at current VDC pointer
VDC_cr=$D600
VDC_dr=$D601

Sta80Fore:
; Send data byte to the VDC chip
       jsr     NewVDCAddress  ; Update VDC address with fg screen pointer (r5)
       ldx     #VDC_DATA      ; request VDC data register
       stx     VDC_cr         ;
10$    bit     VDC_cr         ; test VDC status
       bpl     10$            ; loop till VDC ready for data byte
       sta     VDC_dr         ; store data byte
       rts                    ; exit
```

Lda80Fore:

```
; Get data byte to the VDC chip
       jsr     NewVDCAddress  ; Update VDC address with fg screen pointer (r5)
       ldx     #VDC_DATA      ; request VDC data register
       stx     VDC_cr         ;
10$    bit     VDC_cr         ; test VDC status
       bpl     10$            ; loop till VDC ready for data byte
       lda     VDC_dr         ; get data byte
       rts                    ; exit

; Transfer value in r5 to VDC internal hi/lo address register.
; Destroys:    x
NewVDCAddress:
       ldx     #VDC_HI_UPDATE ;
       stx     VDC_cr         ; ask VDC for high byte
10$    bit     VDC_cr         ; check VDC status
       bpl     10$            ; and loop till VDC ready
       ldx     r5H            ; store hi-byte of address
       stx     VDC_cr         ; to VDC chip

       ldx     #VDC_LO_UPDATE ; ask VDC for low-byte
       stx     VDC_cr         ;
20$    bit     VDC_cr         ; check VDC status
       bpl     20$            ; and loop till VDC ready
       ldx     r5L            ; store lo-byte of address
       stx     VDC_dr         ; to VDC chip
       rts                    ; exit
```

i_VerticalLine:

```
;**************************************************
; Inline version of VerticalLine.
; Pass:
;   .word    x1
;   .word    x2
;   .byte    y1
;**************************************************
IVERT_BYTES = 5     number of inline bytes in call

i_VerticalLine:
      ;---  Save away the inline return address
      PopW  returnAddress

      ;---  Load up VerticalLine's parameters
      ldy   #VJBYTES
      lda   (returnAddress),y       ; get yl parameter first
      sta   rllL
10$
      dey                           ; load other params in a loop
      lda   (returnAddress),y       ; They occupy consecutive GEOS
      sta   r3L-l,y                 ; pseudoregisters, so this will,
      cpy   #1                      ; work correctly
      bne   10$

      ;---  Now call VerticalLine with registers loaded
      jsr   VerticalLine

      ;---  and do an inline return
      php                           ; save st reg to return
      lda   #IVERT_BYTES +1         ; # of bytes + 1
      jmp   DoInlineReturn          ; jump to inline return. Do not jsr!
```

icons/menu

ChangeMode

IconsUp:

```
IconsUp:
    LoadB    dispBufferOn,#(ST_WR_FORE | ST_WR_BACK)        ;draw to both buffers
    LoadW    r0,#IconTable
    jsr      DoIcons ;exit
    rts
```

Important: Due to a limitation in the icon-scanning code, the application must always install an icon table with at least one icon. If the application is not using icons, create a dummy icon table with one icon (see below).

```
;*****************************************************************
; NoIcons: Install a dummy icon table. For use in applications that
; aren't using icons. Call early in the initialization of the
; application, before returning to MainLoop.
;*****************************************************************

DummylconTable:
       .byte 1            ; one icon
       .word NULL         ; dummy mouse x (don't reposition)
       .byte NULL         ; dummy mouse y
       .word NULL         ; bitmap pointer to $0000 (disabled)
       .byte NULL         ; dummy x-pos
       .byte NULL         ; dummy y-pos
       .byte 1,1          ; dummy width and height
       .word NULL         ; dummy event handler

Nolcons:
      LoadW r0,#DummyIconTable      ; point to dummy icon table
      jmp DoIcons               ; install. Let DoIcons rts
```

mouse/sprite

ChangeMode

ArrowUp:

```
;****************************************************
; Put up a new mouse picture
;****************************************************

ArrowUp:
    LoadW r0, #DnArrow          ;point at new image
    jsr      SetMsePic          ;install it
    rts


;macro to store a word value in high/low order

.macro HILO word
    .byte ]word,[word
.endm


;Mouse picture definition for down-pointing arrow

DnArrow:
    HILO %1111111110000000     ;mask
    HILO %1111111001111110
    HILO %0001100111111001
    HILO %0110011111100111
    HILO %0111111110011111
    HILO %0111111110011111
    HILO %0111111111101111
    HILO %0000000000001111

    HILO %0000000000000000     ;image
    HILO %0000000001111110
    HILO %0000000111111000
    HILO %0110011111100000
    HILO %0111111110000000
    HILO %0111111110000000
    HILO %0111111111100000
    HILO %0000000000000000
```

utility

NewIsMseInRegion:

```
;****************************************************
; Replacement for IsMseInRegion.
; Handles the disabling of interrupts so return status registers
; are not effected by plp.
;****************************************************
NewIsMseInRegion:
      php                           ; disable interrupts around coordinate checks
      sei                           ; so it doesn't change while we're looking
      lda   mouseYPos               ; get mouse y-position
      cmp    r2L                    ; compare to top edge
      blt    20$                    ; branch if outside
      cmp    r2H                    ; compare to bottom edge
      bgt    20$                    ; branch if outside
      CmpW   mouseXPos,r3           ; compare mouseX with left edge
      blt    10$                    ; branch if outside
      CmpW   mouseXPos,r4           ; compare mouseX with right edge
      bgt    10$                    ; branch if outside
      plp                           ; (restore interrupts before setting st reg)
      lda    #TRUE                  ; return outside region status
      rts                           ; exit
10$
      plp                           ; (restore interrupts before setting st reg)
20$
      lda                           ; return inside region status
      rts                           ; exit

      LoadW  r3,windowX1            ; get coordinates of window's rectangle
      LoadW  r2L,windowY1
      LoadW  r4,windowX2
      LoadW  r2H,windowY2
      jsr    NewIsMseInRegion       ; check for mouse inside region
      beq    MouseOutsideWindow     ; branch if outside window area
```

text

Text

text

Placeholder

Keyboard

KillPrompt:

```
.if COMMENT
*****************************************************
Purpose:      Safely turn off Text Prompt.

Pass:         nothing

Returns:      nothing

Alters:       alphaFlag

Destroyed:    a, x, r3L

DESCRIPTION:  Disables Interrupts and then Turns Text Prompt off
*****************************************************
.endif

KillPrompt:
      php                    ; save interrupt status
      sei                    ; disable interrupts
      jsr    PromptOff       ; prompt - off
      LoadB  alphaFlag,0     ; clear alpha flag
      plp                    ; restore interrupt status
      rts
```

utility

BeepThrice	Beep three times. Runs off the **MainLoop** by using **Sleep**
HandleCommand	Given a command number this routine handles dispatching control to the appropriate routine.
LoadBASIC	Loads a Commodore BASIC program and starts it running.

BeepThrice:

```
;****************************************************
; Beep three times
; Runs off the MainLoop by using Sleep
;****************************************************

.if TARGET_NTSC
    FRAME_RATE=60
.else
    FRAME_RATE=50
.endif

BELL_INTERVAL = (FRAME_RATE/10) ;approximately. 1/10 second.

BeepThrice:
      jsr    Bell                ; sound the bell
      LoadW  r0,BELL_INTERVAL    ;
      jsr    Sleep               ; pause a bit
      jsr    Bell                ; sound the bell again
      LoadW  r0,BELL_INTERVAL
      jsr    Sleep               ; pause a bit
      jmp    Bell                ; sound the bell again and let bell rts
```

Note[3]: see **GetFPS** for detecting Frame Rate for portability between hardware.

HandleCommand:

```
;**************************************************
; HandleCommand
; DESCRIPTION: Given a command number this routine handles dispatching
; control to the appropriate routine.
;
; Pass: y command number
; Returns: depends on command
; Destroyed: depends on command
;**************************************************
UNIMPLEMENTED = $0000

HandleCommand:
    cpy      #TOT_CMDS        ; check command # against last cmd num
    bcs      99$              ; exit if command is invalid
    ldx      CMDtabH,y        ; get high byte routine address
    lda      CMDtabL,y        ; get low byte of routine address
    jsr      CallRoutine      ; call the routine
99$
    rts                       ; exit

; The table below is a collection of the the high/low bytes of the routine
; associated with each command number. If a command is not yet implemented
; use the UNIMPLEMENTED constant
CMDtabL:
    .byte    [UNIMPLEMENTED   ; Low Byte of command 0
    .byte    [Cmdl            ; Low Byte of command 1
    .byte    [Cmd2            ; etc...
    .byte    [Cmd3
    .byte    [Cmd4
    .byte    [Cmd5
CMDtabH:     ;low bytes
    .byte    ]UNIMPLEMENTED   ; High Byte of command 0
    .byte    ]Cmdl            ; High Byte of command 1
    .byte    ]Cmd2            ; etc...
    .byte    ]Cmd3
    .byte    ]Cmd4
    .byte    ]Cmd5

TOT_CMDS = (CMDtabH-CMDtabL)  ; Total Number of commands

Cmd1:
    ;Perform some action here.
    rts
Cmd2:
    ;Perform some action here.
    rts
Cmd3:
    ;Perform some action here.
    rts
Cmd4:
    ;Perform some action here.
    rts
Cmd5:
    ;Perform some action here.
    rts
```

LoadBASIC:

```
; Loads a Commodore BASIC program and starts it
; running. Assumes that the program is a standard BASIC
; file that loads at $801. This example does little
; error checking.
;
; Pass:      Nothing
;
*****************************
UNIMPLEMENTED = $0000
basicProg:
    .byte    "GodZilla",NULL

runCommand:
    .byte "RUN",NULL

LoadBASIC:
    LoadW    r6,basicProg     ; Find Basic Program to run
    jsr      FindFile         ; r5 will now point to programs DIR Entry
    txa
    bne      99$              ; If FILE_NOT_FOUND or other Disk Errors exit.
    LoadW    r0,runCommand    ; point at command string
    LoadW    r7,#$801         ; assume standard address
    jmp      ToBasic
99$
    sec
    rts
```

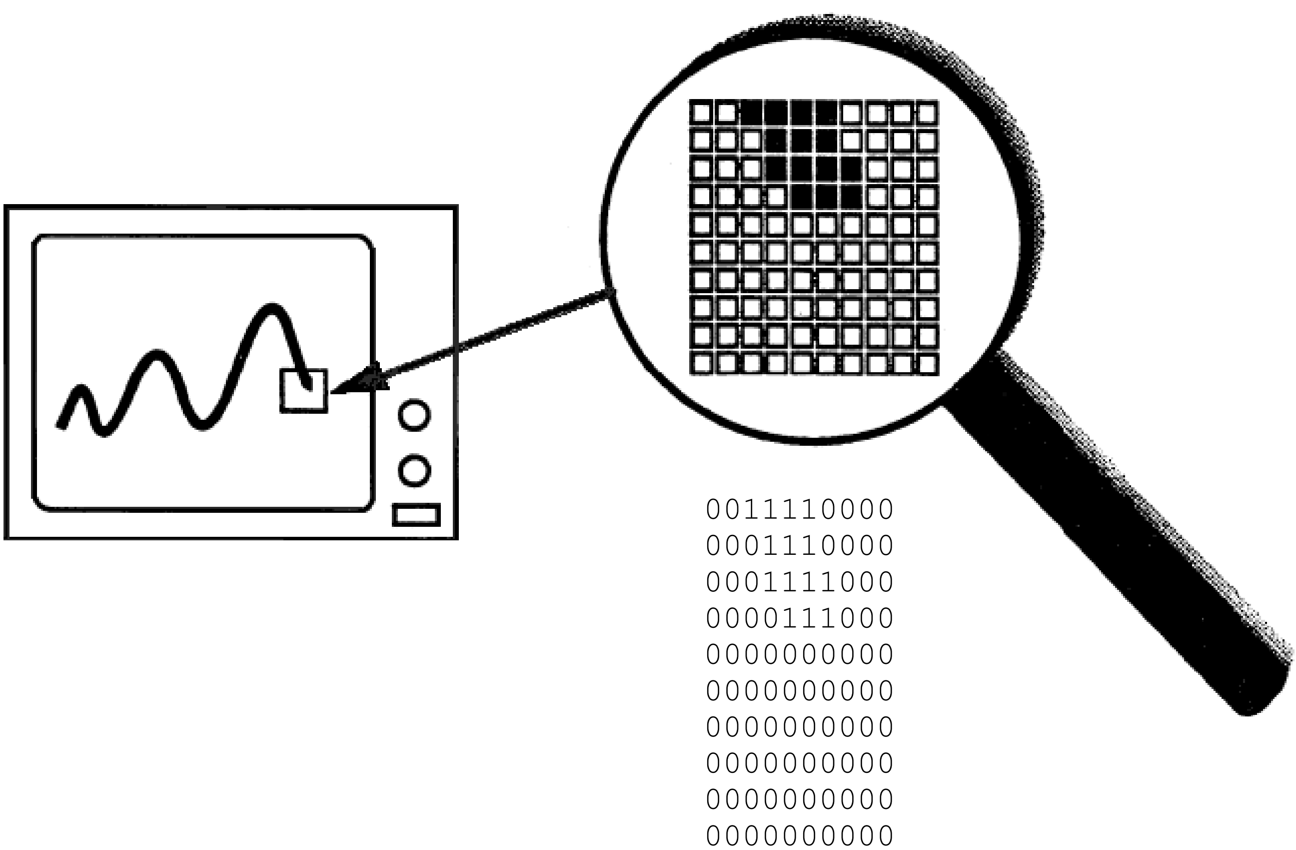
Graphic Routines

As the name GEOS (Graphics Environment Operating System) implies, screen graphics are central to both the operating system and its applications. GEOS provides a number of graphic primitives ("primitive" because they are the basis of more complex objects) for drawing points, lines, rectangles, and other objects, as well as displaying bitmap images such as those cut from geoPaint. GEOS also provides graphic support routines for undoing regions, inverting areas, scrolling, and directly accessing the screen memory.

Drawing with the built-in GEOS routines increases program portability by making much of the internal, machine-dependent screen architecture transparent to the application. When you draw a line, for example, you merely supply the two endpoints. GEOS takes care of calculating the proper pixel locations and modifying the screen memory. This allows an application to use the same code to draw lines on machines with very different graphics hardware and spares the programmer from dealing directly with screen memory.

Introduction to GEOS Graphics

If you look closely at a monitor or television screen, you will notice that the image is made up of many small dots. These small dots, called pixels, can be either on or off and are represented in memory by l's and 0's, respectively. A pixel with a value of one is considered set and a pixel of value zero is considered clear. This binary, or bitwise, representation of images is referred to as bitmapped graphics, and a bitmap is a picture or image created in this way.



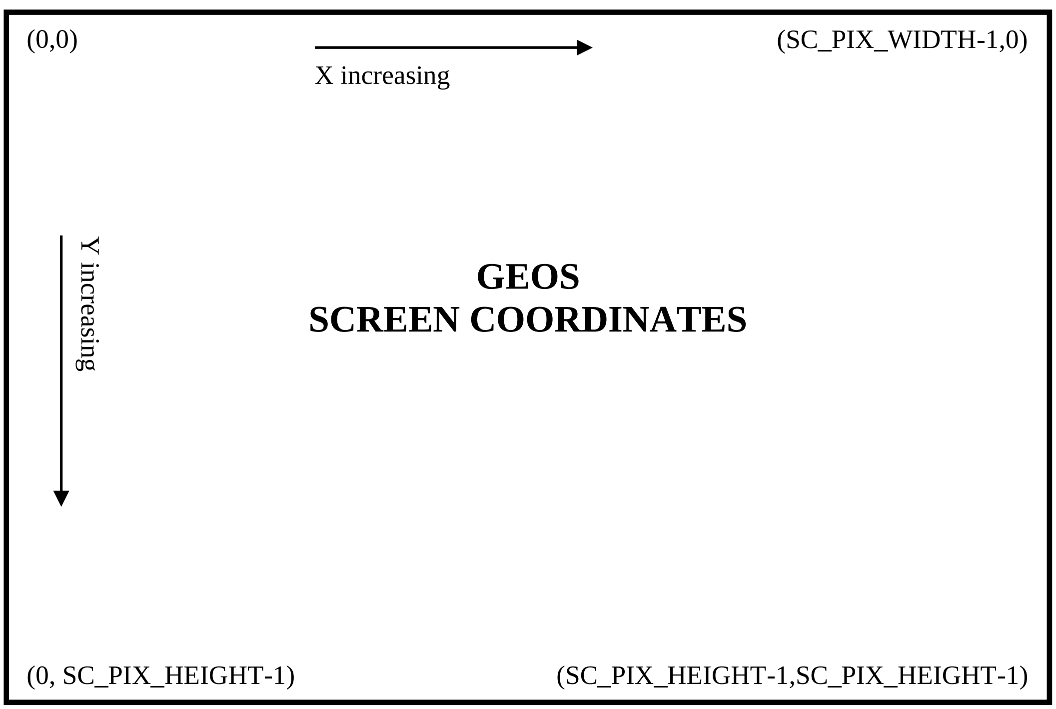
Color

Although some hardware configurations support color graphics, GEOS assumes that the screen is a monochromatic device; that is, GEOS only deals with one drawing color and one background color. Typically, the drawing color is black, like ink and the background color is white, like a piece of paper. Depending on the monitor being used and the Preference Manager settings, the actual displayed colors may be different. We will refer to the color displayed by a zero-pixel as the background color and the color displayed with a one-pixel as the drawing color. Applications that support multiple drawing colors, such as the Commodore 64 version of geoPaint, must do so on their own, bypassing GEOS (at the expense of portability) to provide multiple colors on the screen.

The GEOS Virtual Screen

The GEOS screen is often referred to as virtual screen, one whose layout and internal storage characteristics exist independent of any underlying graphics hardware. For this reason, the GEOS screen is fundamentally identical under all versions of the operating system.

The GEOS screen is a rectangular array of pixels arranged like a sheet of graph paper. Each pixel on the screen has a corresponding (x,y) coordinate. The x-axis begins with zero and runs horizontally (left to right) across the screen, and the y-axis begins with zero and runs vertically (top to bottom) down the screen. The maximum x- and y-positions, because they differ from machine to machine, are calculated by subtracting one from the GEOS constants SC_PIX_WIDTH and SC_PIX_HEIGHT.



Important: GEOS does no clipping or range-checking on coordinates passed to it. If you pass it invalid data or coordinates, the results are unpredictable and will often crash the application.

GEOS 128 40/80-Column Support

Because applications that run under GEOS 128 may want to take advantage of both the 40- and 80-column screen modes, the following conventions have been adopted for the screen width and height constants:

- *The following constants can be used to access the dimensions of the 40- or 80-column screen specifically:*

SC_40_WIDTH	320	Pixel width of 40-column screen.
SC_40_HEIGHT	200	Pixel height of 40-column screen.
SC_80_WIDTH	640	Pixel width of 80-column screen.
SC_80_HEIGHT	200	Pixel height of 80-column screen.

- *If the application is designed to run under GEOS 128 only and not run under GEOS 64 (the C64 constant is set to $00 and the C128 constant is set to $01), then the standard SC_PIX_WIDTH and SC_PIX_IEIGHT constants take on the following values:*

SC_PIX_WIDTH	Pixel width of 80-column screen.
SC PIX_HEIGHT	Pixel height of 80-column screen.

- *If the application is designed to run under GEOS 64 and GEOS 128 (both the C64 constant and the C128 constant set to $01), then the standard SC_PIX_WIDTH and SC_PIX_HEIGHT constants take on the following values:*

SC_PIX_WIDTH	Pixel width of 40-column screen.
SC PIX_HEIGHT	Pixel height of 40-column screen.

This is because the application (typically) will be written with the 40-column screen in mind. At runtime, the application can check to see which version of GEOS it is running under and add doubling bits to the appropriate coordinate values so that the 40-column coordinates will be normalized automatically when GEOS 128 is in 80-column mode.

An application can use the following subroutine to determine whether it is running under GEOS 128 or GEOS 64: **`Check128`**

When running under GEOS 128, the **graphMode** variable may be checked to determine whether GEOS is in 40- or 80-column mode:

```
bit   graphMode   ; check 40/80 mode bits
bpl   C64Mode     ; branch if in 40-column mode
                  ; else, handle as 80-column...
```

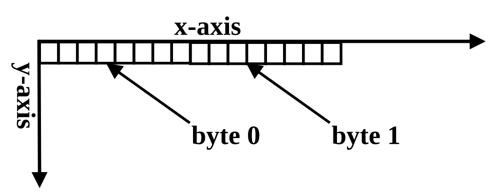
For more information, refer to "GEOS 128 X-position and Bitmap Doubling11 in this chapter. Also see **NormalizeX** in the Routine Reference Section.

Inclusive Dimensions

All dimensions and GEOS coordinates are inclusive: a line contains the endpoints which define it, and a rectangle includes the lines that make up its sides. For example, a rectangle defined by an upper-left corner of (10,10) and a lower-right corner of (20,20) would include the lines around its perimeter defined by the points (10,10), (10,20), (20,10), and (20,20).

Linear Bitmap

For the purpose of bitmap compaction and patterns, the GEOS screen is treated as a *linear bitmap*, a contiguous block of bytes with each bit controlling an individual pixel. The bytes are lined up end-to-end for each screen line! The high-order bit (bit 7) of each byte controls the leftmost pixel and the low-order bit (bit 0) controls the rightmost pixel.



GEOS Virtual Screen

Keep in mind that this is a conceptual organization of the screen; the actual in-memory storage of the screen and bitmap data may be very different.

Dividing the Screen Into Cards

Many GEOS routines subdivide the GEOS virtual screen into 8x8-pixel blocks called cards. A card is a two-dimensional unit of measurement eight pixels on each side. The first card begins in the upper-left corner of the screen (0,0) and extends to (7,7). The next card is just to the right of the first and extends from (8,0) to (15, 7).

Cards are always aligned to eight pixel boundaries called *card boundaries* (pixel positions 0, 8,16, 24, etc.). Aligning an object to a card boundary is called *card alignment*, and the position of an object expressed in cards is called its *card position*. Pixel position (32, 72), for example, would correspond to card position (4, 9) because 32/8 =.4 and 72/8 = 9). The *card width* of an object is its width in cards, and the card height is its height in cards. An entire row of cards is called a *cardrow*.

The card is a convenient unit of measurement because its dimensions, 8x8, which is a power of 2, lend themselves to simple binary arithmetic. For example, converting a pixel position to a card position is merely a matter shifting right three times. See **MseToCardPos** in Examples / Graphic Routines

Example: **MseToCardPos**

Cards are also convenient because they map directly to the internal storage format of the Commodore 40-column graphics screen. (Converting to other formats, such as the Commodore 128 80-column screen or the Apple II double hi-res screen, requires additional translation. This translation is handled automatically by the GEOS graphics routines.)

Display Buffering

Normally the application has control of the screen. But when an item such as a dialog box or a menu is displayed, GEOS overwrites the screen. When the dialog box is removed or the menu is retracted, GEOS needs to restore the portion of the screen it destroyed. For this purpose, GEOS maintains a background screen buffer. Most of the time, the background buffer contains an exact copy of the foreground screen (the screen that is displayed) because GEOS normally sends graphics data to both screen buffers. When a temporary object is displayed, however, it is only drawn to the foreground screen. Removing the object, or recovering the original area of the screen, is then simply a matter of copying pixels from the background buffer to the foreground screen. The GEOS dialog box and menu routines-handle this sort of recovery automatically.

dispBufferOn

Usually the application will want to draw to both buffers so that GEOS can properly recover the foreground screen after menus and dialog boxes. If graphics are only drawn to one buffer and a menu is brought down or a dialog box is displayed, the subsequent recover may restore the wrong data.
However, sometimes an application may want to limit drawing to only the foreground or background screen buffer. GEOS graphics and text routines use the global variable **dispBufferOn** to determine whether to draw to the foreground screen, the background buffer, or both simultaneously. Bits 6 and 7 of **dispBufferOn** determine the writing and reading mode:

bit 7:	1	— use foreground screen.
	0	— do not use foreground screen.
bit 6:	1	— use background buffer.
	0	— do not use background buffer.
bits	5-0:	*reserved for future use* — should always be zeros

There are some constants which allow you to gain access to these bits:

ST_WR_FORE use foreground.
ST_WR_BACK use background.

and they can be used in following manner:

;Use both foreground screen and background buffer (normal).
```
LoadB dispBufferOn, # (ST_WR_FORE | ST_WR_BACK)
```

;Use foreground screen only.
```
LoadB dispBufferOn, #ST_WR_FORE
```

;Use background buffer only.
```
LoadB dispBufferOn, #ST_WR_BACK
```

Important: If bits 6 and 7 of **dispBufferOn** are both zero, GEOS considers this an undefined state and will not produce useful results. In most cases, the internal address calculations will force your graphic objects to appear in the center of the drawing area where they can do little harm. If the center line on the screen becomes garbled, **dispBufferOn** probably contains a bad value.

Using dispBufferOn

Typically applications leave **dispBufferOn** set to draw to both screens, whereas most desk accessories will only draw to the foreground screen. In some situations, an application may want to limit drawing to the foreground screen so that it may recover from the background buffer at a later time. Internally this is what GEOS does when it opens a menu or dialog box: the object is only drawn to the foreground screen, and when it needs to be erased, the original data is recovered from the background buffer. **dispBufferOn** can also be used to pre-draw complex objects in the background buffer (**ST_WR_BACK**) and make them instantly appear on the foreground screen by doing a recover.

An application must take special precautions when using **dispBufferOn** to draw selectively to one buffer or the other. For example, when GEOS automatically recovers from a menu or a dialog box, it recovers the data from the background buffer. If the background buffer has not been updated (the application has been drawing with the **ST_WR_BACK** bit cleared, for example), then the menu or dialog may recover the wrong data.

Since dialog boxes are only displayed when the application calls **DoDlgBox** and menus are only opened while GEOS is in **MainLoop**, the application has some control over GEOS's automatic recovering. The application can postpone displaying dialog boxes and returning to **MainLoop** until the foreground screen and background buffer contain the same data. If an application must return to **MainLoop** while the buffers contain different data (to let processes run, for example), it can always disable menus by clearing the **MENUON_BIT** bit of **mouseOn**. The menus may be reenabled again by restoring the **MENUON_BIT** bit of **mouseOn**:

Example: **`StopMenus`**

Using the Background Buffer as Extra Memory

Some applications are so starved for memory that they opt to use the background buffer for program code or data. To do this, they must always keep the ST_WR BACK bit of dispBufferOn clear so that the background buffer is not corrupted with graphic data.

If you disable the background buffer, GEOS cannot automatically recover after menus and dialog boxes. The application must provide its own routine for restoring the foreground screen. There is a GEOS vector called RecoverVector, which normally points to the RecoverRectangle routine. Whenever GEOS needs to recover from a menu, dialog box, or desk accessory, it sets up parameters as if it were going to call RecoverRectangle and jsr's indirectly through the address in RecoverVector. If the application is using the background buffer, it must place the address of its own screen recover routine in RecoverVector. When GEOS needs to recover a portion of the screen, it will jsr to the application's recover routine with the following register values describing the rectangular area to recover:

r3 X1 — x-coordinate of upper-left (word).
r2L Y1 — y-coordinate of upper-left (byte).
r4 X2 — x-coordinate of lower-right (word).
r2H Y2 — y-coordinate of lower-right (byte).

where (X1,Y1) is the upper-left corner and (X2,Y2) is the lower-right corner of the rectangular area to recover. The rectangle's coordinates are inclusive. The application must then use these values to restore the portion of the screen that lies within the rectangle's boundaries and return with an rts. This recovery can be as simple as filling with a halftoned pattern or as involved as redrawing graphic and text objects that fall within the rectangular recover area.

Most of the larger Berkeley Softworks GEOS applications use a technique called saveFG/recoverFG (short for "save foreground" and "recover background") to save and recover the foreground screen when displaying menus and dialog boxes. Basically, saveFG will save a rectangular subregion of the foreground screen to a special buffer just before GEOS displays a menu or a dialog box. When GEOS tries to recover from the background buffer, recoverFG restores the data from the special buffer. Although the size of the buffer varies from application to application, it will seldom be larger than 5.5K (just large enough to hold the largest standard dialog box).

Transferring data to and from the buffer is fairly straightforward. With the Commodore 40-column screen, it is mostly a matter of calculating the proper address offsets and copying bytes. With the GEOS 128 80-column screen, the process is complicated a bit because the bytes must be read from the VDC chip's RAM.

The real trick is knowing how to intercept the normal GEOS menu and dialog box drawing and recovering mechanisms. Dialog boxes are the easiest because they are always called by the application. The program only needs to save the foreground screen area prior to calling **DoDlgBox**. The size of the dialog box can be calculated from its table (be sure to account for any shadow) and the foreground data can be copied into the saveFG buffer. When the dialog box is finished, GEOS will jsr through **RecoverVector**. The application installs its own recoverFG routine into **RecoverVector** and restores the foreground area from the saveFG buffer. The GEOS dialog box recovery does have one quirk that concerns shadowed dialog boxes. GEOS shadowed dialog boxes consists of two overlapping rectangular areas: the actual dialog box and the slightly offset shadow rectangle. GEOS first calls through **RecoverVector** once for the region bounded by the shadow box, then again for the region bounded by the dialog box. When saving the foreground area, the entire dialog box region (the area bounded by the union of all eight corner points) should be saved and a special flag should be set so that the area is only recovered once. Under Apple GEOS, the recovery of dialog box shadows can be suppressed by setting recoverOnce to a non-zero value. When recoverOnce is non-zero, GEOS only vectors through **RecoverVector** once with the bounding rectangle of the dialog box. The application's recover routine will need to compensate for the shadow box. For more information on dialog boxes, refer to Chapter ?@DLG@. FIXME

Saving the foreground area before a menu is displayed is a bit tougher because GEOS displays menus at **MainLoop**, the application has little notice that a submenu is opening up. Fortunately, there is a workaround: GEOS supports a special type of sub-menu called a dynamic sub-menu. Just before a dynamic sub-menu opens, GEOS calls a subroutine whose address is stored in the menu data structure. This opportunity can be used to save the foreground screen area before GEOS draws the menu by calculating the bounding rectangle from the menu structure. When GEOS recovers a menu, it calls through **RecoverVector** as it does with dialog boxes. With multiple sub-menus, the menus are always recovered in the reverse order they were drawn. For more information on menus, refer to Chapter "Icons, Menus, and Other Mouse presses"

Manual Imprinting and Recovering

Within an application, data can be moved between the foreground screen and background buffer with GEOS routines that copy data to and from the two areas. Copying data from the foreground screen to the background buffer is called imprinting, and copying data from the background buffer to the foreground screen is called recovering. There are GEOS routines for imprinting and recovering points, lines, and rectangular regions.

Some Possible dispBufferOn Complications

When drawing with both buffers enabled (with both foreground and background bits set in **dispBufferOn**), GEOS requires that the foreground screen and the background buffer contain exactly the same data. If they are different, the results of graphic operations may be unpredictable. If you need to draw to the foreground screen and the background buffer when they contain different data, you must perform the graphic operation once by writing only to the foreground screen, and then a second time, writing only to the background buffer — you cannot write to both of screen areas simultaneously if they contain different data.

Machine Dependencies

The GEOS graphics routines hide much of the underlying hardware from the application. This allows the same code to run under a variety of different environments with very few changes. However, it is sometimes necessary to optimize graphic routines for a specific machine. This can be as simple as taking advantage of color display capabilities or as complex as direct screen memory manipulation. Either way, an application should only resort to such tactics when the desired effect cannot be achieved through the standard graphics routines. Be aware that circumventing the GEOS Kernal will very likely increase your development time and that there is no guarantee that the techniques will be compatible with future versions of GEOS.

Commodore 64

The Commodore 64 version of GEOS uses the standard high-resolution bitmap mode (not multi-color bitmap mode), which is 320 pixels wide by 200 pixels high. Memory is mapped to the screen in eight-byte stacks called *cards*: byte 0 controls pixels (0,0) through (7,0), with bit 7 on the left and bit 0 on the right, and byte 1 controls the same pixels on the line below, which is pixels (0,1) through (7,1). This stacking continues through byte 7, which controls pixels (0,7) through (7,7) and completes the 8x8-pixel card. Byte 8 begins the next card, controlling pixels (8,0) through (15,0). The screen memory begins at **SCREEN_BASE** and occupies 8,000 bytes, extending to **SCREEN_BASE**+7999. The background buffer begins at **BACK_SCR_BASE** and extends to **BACK_SCR_BASE**+7999.

GEOS does not directly support the foreground and background color options of the standard high-resolution bitmap mode. The color matrix, located from COLOR_MATRIX to COLOR_MATRIX+999, is set to a constant foreground and background color as determined by the Preference Manager. If an application wants to support color like geoPaint), it must manage the color matrix itself. Each byte in the color matrix sets the foreground and background colors of a card (8x8 pixel block): color byte 0 sets the colors for card 0 (bitmap bytes 0-7) and color byte 1 sets the colors for card 1 (bitmap bytes 8-15). Before the application exits, it must restore the original color matrix. This best done by saving the first byte and then filling the color matrix before calling EnterDeskTop, as the following code fragments illustrate:

Example:

```
;On entry, save off the first byte of the color matrix
        MoveB   COLOR_MATRIX, saveColor
        .
        .
;On exit, fill the color matrix with the saved value
        LoadW   r0,1000             ; color matrix is 1000 bytes
        LoadW   r1,COLOR_MATRIX
        MoveB   saveColor,r2L       ; fill with original color
        jsr     FillRam
```

Commodore 128

In 40-column mode, GEOS 128 screen memory is identical to the Commodore 64. In 80-column mode, GEOS 128 uses the high-resolution 640x200 mode supported by the 8563 VDC (Video Display Controller) chip.. The foreground screen memory is not stored in the normal Commodore memory but on the VDC chip instead. The VDC RAM is accessed indirectly through the VDC control registers. The screen occupies 16,000 bytes, and each byte is accessed one at time by its address within the VDC display RAM(the first screen byte is at 0, the last at 15999). Bits are mapped sequentially from memory to the screen pixels: bits 7 through 0 of byte 0 (in that order) control the first seven pixels, (0,0) through (7,0). The following byte controls the next seven pixels, (8,0) through (15,0). And so on for the remainder of the screen. The following two subroutines will access bytes in the VDC screen RAM when GEOS 128 is in 80-column mode: See **Sta80Fore, Lda80Fore** in Examples.

For more information on controlling the 8563 VDC chip, refer to the Commodore 128 Programmer's Reference Guide.

Before writing directly to the 80-column foreground screen, be sure to call **TempHideMouse** to temporarily disable the virtual sprites (for more information, refer to **TempHideMouse** in Chapter XX).

Because the 80-column screen requires a 16,000-byte background buffer, GEOS 128 (when in 80-column mode) uses the 8,000-byte 40-column screen foreground buffer (**SCREEN_BASE** to **SCREEN_BASE**+7999) for store the first 100 scanlines of background buffer data and the 8,000-byte foreground screen buffer (**BACK_SCR_BASE** to **BACK_SCR_BASE**+7999) to store the last 100 scanlines of background buffer data. Because these data areas are not contiguous, an application that directly accesses the background screen must compensate for this break.

FIXME. Back Buffer for VDC mode starts at BACK_SCR_BASE for the first half of the screen. Second half starts at roughly $A032. Need to get the exact address on Second half of back screen start address and fix the above paragraph. Possible this paragraph was correct for GEOS 128 V1.3? But not for 2.0

Porting Considerations and Techniques

Outside of the normal considerations for porting a GEOS application from one machine to another, there are a few additional elements which pertain specifically to graphics.

GEOS 128 Virtual Sprites
GEOS 128 (in 80-column mode) renders sprites entirely in software by modifying the actual bitmap screen. (GEOS 64 and GEOS 128 in 40-column mode, use the hardware sprite capabilities of the VIC chip.) In order to properly treat these virtual sprites as if they were apart from the bitmap screen, they must be erased before any graphic operation, whether drawing, testing, imprinting, or recovering, is done. To do this, Apple GEOS and GEOS 128 provide the **TempHideMouse** routine to temporarily remove all sprites. The sprites are not redrawn until the application returns to MainLoop. Normal GEOS graphics and text routines will automatically call **TempHideMouse**; only applications that are directly accessing the foreground screen area need call **TempHideMouse**. For more information, refer to **TempHideMouse** in the Routine Reference Section "Software Sprites11 in Chapter®SPRITE®. FIXME

GEOS 128 X-position and Bitmap Doubling
Because the GEOS 128 80-column bitmap screen has a horizontal resolution exactly twice that of GEOS 64 (640 vs. 320), GEOS 128 supports the ability to automatically double the Xcoordinate (s) of graphic and text objects, and the width of bitmap objects, by setting special bits in the x-position and width calling parameter(s). This allows the visual elements of a GEOS 64 application to run in 80-column mode under GEOS 128 with a minimum of effort. The special bits can also be added at run-time to dynamically configure a program to run correctly under both GEOS 64 and GEOS 128. X-position and bitmap doubling is supported by nearly every GEOS 128 routine that writes to the screen (including text, dialog box, and icon routines). The following constants may be bitwise or'ed into GEOS 128 x-coordinates and bitmap widths to take advantage of the automatic 80-column doubling features:

DOUBLE_W	For doubling word-length values. Normal Xcoordinates, such as those passed to Rectangle and DrawPoint.
DOUBLE_B	For doubling byte-length values. A byte-length value is either a card x-position or a card width, both of which apply almost exclusively to bitmap routines, such as BitmapUp and **`BitmapClip`**.
ADD1_W	Used in conjunction with DOUBLE_W; adds one to a doubled word-length value. This allows addressing odd-coordinates, as when drawing a one-pixel frame around a filled rectangle.

These doubling bits have no effect when GEOS 128 is in 40-column mode but come to life when GEOS 128 is in 80-column mode. For example, the following code fragment will frame a filled rectangle. It will appear similarly in both 40- and 80-column modes. See **FilledRect** in Examples:

Important:	GEOS 128 filters all word-length x-coordinates (but not widths or byte-length Xcoordinates) through the routine **NormalizeX** to process the doubling. For more detailed information on how this routine works, refer to its documentation in this chapter. **`NormalizeX`** will also double signed x-coordinates. If the x-coordinate is a signed number (like you might pass to SmallPutChar), then the double bits must be exclusive-or'ed into the x-coordinate parameters rather than simply or'ed.

The graphic elements of existing GEOS 64 applications can be ported to run under GEOS 128 with a minimum of effort by taking advantage of the GEOS 128 doubling bits. However, once the doubling bits have been installed, the application will no longer run under GEOS 64. The simplest approach to this problem is to have two entirely different applications. One designed to run under GEOS 64 and the other designed to run under GEOS 128. The doubling bits may be controlled at assembly-time with conditional assembly, as the following example illustrates.

Example: DblDemo1

Designing an application so that it runs well under both GEOS 64 and GEOS 128 is a more difficult task. It usually involves using self-modifying code: part of the initialization code for each module can check the version of GEOS it is running under (use the Checkl28 subroutine illustrated in "GEOS 128 40/80-Column Support" in this chapter) and add the proper doubling-bits to all relevant x-coordinates.

Note[3]: A More efficient method is to build the application with all doubling in place. Then if the program detects it is on a C64 it will remove the doubling bits with a simple and #%00011111. If you are trying to add doubling instead then you have to have additional logic to handle when an ADD1_W gets applied.

Note[3]: The best correct solution has not been created yet as of this writing. If the C64 kernal was updated to be able to use NormalizeX in the same way 40 Column GEOS on the 128 does, then all applications could be written with no need for self modification and would work the same on C64/C128 40/80.

Points and Lines

Points

The simplest graphic operation involves setting, clearing, or testing the state of an individual pixel, or point, on the screen. GEOS provides two routines for working with points:

DrawPoint	Set or clear a single point.
TestPoint	Test a single point: is it set or clear?

Horizontal and Vertical Lines

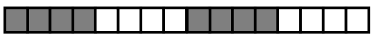
Due to the rectangular nature of bitmapped graphics, horizontal and vertical lines are inherently fast and easy to create and manipulate. GEOS provides five routines for working with horizontal and vertical lines:

• HonzontalLine	Draw a horizontal line with a repeating bit pattern.
• VerticalLine	Draw a vertical line with a repeating bit pattern.
• InvertLine	Invert the pixels in a horizontal line.
• ImprintLine	Imprint a horizontal line to the background buffer.
• RecoverLine	Recover a horizontal line from the background buffer.

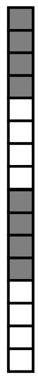
Line Patterns.

Both **HorizontalLine** and **VerticalLine** use a byte-sized bit pattern when creating the line. Each bit in the pattern byte represents a pixel in the line: wherever a one appears in the pattern byte, the corresponding pixel will be set, and wherever a zero appears , the corresponding pixel will be cleared. This allows lines which vary from solid (all l's) to dashed (a mixture of Ts and O's) to clear (all O's). Note: this concept of a line-pattern is different from the 8x8 GEOS fill patterns used for rectangles.

Bits in the pattern byte are used left-to-right for horizontal lines and top-to-bottom in vertical lines, where bit 7 is at the left and the top, respectively. A bit pattern of %11110000 would create a horizontal line like:



and a vertical line like:

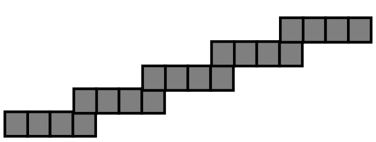


The pattern byte is always drawn as if aligned to an eight-pixel boundary. If the endpoints of a line do not coincide with eight-pixel boundaries, then bits are masked off the appropriate ends. The effect of this is that a pattern is always aligned to specific pixels, regardless of the endpoints and that adjacent lines drawn in the same pattern will line up. That is, positions 0, 8, 16, 24, etc. will always depend on pattern bit 7, and positions 1,9,17,25, etc. will always depend on pattern bit 6.

> **Note:** Because of the internal memory layout of screen memory, horizontal lines will often draw up to eight times faster than vertical lines.

Diagonal Lines

For the same reason that bitmap displays are well-suited for displaying horizontal and vertical lines, they are ill-suited for displaying diagonal lines. A smooth, even-density line cannot be drawn diagonally between two points (except at 45-degree angles) — the points on the line must be approximated in a stairstep fashion:



GEOS provides one routine for drawing and recovering a line between two arbitrary points:

DrawLine	Draw or recover a line between any two points.

DrawLine does not utilize a pattern byte; it will either set or clear all pixels between the two endpoints.

> **Note:** **DrawLine** is the most general-purpose drawing routine. It can be used to draw single points (both endpoints the same), horizontal and vertical lines, or lines at arbitrary angles. However, it is burdened by this flexibility> making it appreciably slower than the other plotting routines.

Patterns and Rectangles

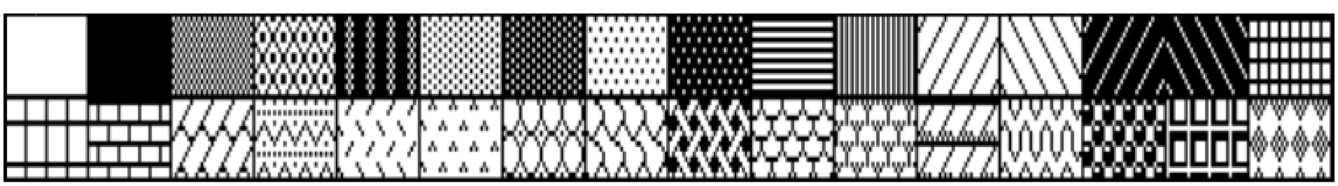
Fill Patterns
GEOS uses two types of patterns: line patterns and fill patterns. A line pattern is a one-byte repeating pixel pattern used by routines like HorizontalLine and VerticalLine, and a fill pattern is an 8x8 pixel block represented by eight bytes in memory and used by routines like Rectangle. Line patterns are discussed in "Points and Lines11 earlier in this chapter. Fill patterns are discussed here.

A 50% fill pattern might be defined by the following:

```
.byte %10101010
.byte %01010101
.byte %10101010
.byte %01010101
.byte %10101010
.byte %01010101
.byte %10101010
.byte %01010101
```

The pattern has alternating set and clear pixels. Drawing a filled rectangle in this pattern would produce a medium-dark block.

All versions of the GEOS Kernal contains the following predefined patterns:



Fills occur in the current pattern. The current pattern can be changed with the following routine:

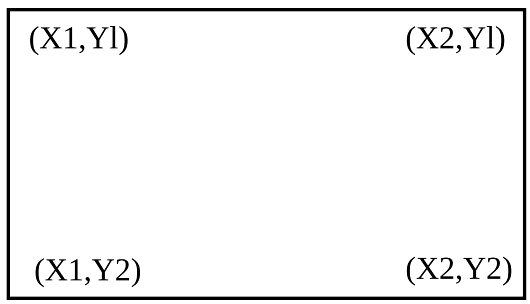
• **SetPattern**	Set the current pattern.

To use one of the system patterns, the application would first call **SetPattern** with the appropriate pattern number. **SetPattern** calculates the proper pattern address, the address of the eight-byte block, and places it in the GEOS variable **curPattern**. Any subsequent call to a routine which uses a system pattern will index off of the address in **curPattern** to access the 8x8 block. Some applications, finding the need to define their own patterns, modify either the address in **curPattern** to point to their own eight-byte pattern or use the address in **curPattern** (after a valid call to **SetPattern**) to modify the GEOS system patterns directly.

> **Note:** GEOS does not restore the system patterns when an application exits. If an application modifies the patterns, it should restore them when it exits unless it is desirable for the next application to inherit the redefined patterns (as with the GEOS Pattern Editor).

Rectangles

Rectangles in GEOS are defined by their upper-left and lower-right corners. The upper-left is usually referred to as (X1,Y1) and the lower-right as (X2,Y2), where XI, X2, Yl, and Y2 are valid x and y screen positions. From these two coordinates, the rectangle routines can determine the coordinates of the other two corners:



GEOS provides five routines for dealing with rectangular regions:

• Rectangle	Draw a solid rectangle using the current fill pattern.
• FrameRectangle	Draw an unfilled rectangle (bounding frame).
• InvertRectangle	Invert the pixels in a rectangular area.
• ImprintRectangle	Imprint a rectangular area to the background buffer.
• RecoverRectangle	Recover a rectangular area from the background buffer.

Bit-mapped Images

All graphic picture objects, such as icons and Photo Scrap images cut from geoPaint, are stored internally in GEOS Compacted Bitmap Format to save space. When you paste an image or icon into a geoProgrammer source file, it is in compacted bitmap format, and when you read a geoPaint image, it too is in compacted bitmap format If a compacted image were to be copied directly to the screen, it would very likely be unrecognizable. GEOS bitmap routines first decompact the image and then transfer it to the screen area.

Standard Bitmap Routines

All versions of GEOS support the following bitmap routines:

• BitmapUp	Place a full compacted bitmap on the screen.
• BitmapClip	Place a rectangular subset of a compacted bitmap on the screen.
• BitOtherClip	Special version of BitMapClip which uses an application-defined routine to collect the compacted bitmap data a byte at a time, allowing the image to come from disk or other I/O device.

GEOS bitmaps are compacted from the GEOS virtual screen format rather than the internal machine format. Because the standard bitmap routines deal with byte-sized chunks (eight-pixels at a time), the following apply:

- Horizontally, the bitmap occupies pixels up to the nearest eight-pixel (byte) boundary. That is: a bitmap of five pixels is extended to eight and a bitmap of 30 pixels is extended to 32 pixels. Bitmaps which are not evenly divisible by eight (in the horizontal direction) are usually padded with zero bits.

- Bitmaps can only be placed at eight-pixel intervals on the x-axis (0, 8, 16...). This limitation does not apply to the y-axis.

GEOS Compacted Bitmap Format

The GEOS compacted bitmap format relies on the observation that pixel patterns in bitmap images are frequently repetitive. If you were to examine a rectangular area of the screen (in GEOS linear bitmap format) it would often be the case that adjacent bytes would be identical. The compacted bitmap format encodes this redundancy into groups of bytes called packets. Each packet can decompress to a large number of bytes in the actual bitmap.

Packet Format

Each packet in a GEOS compacted bitmap follows a specific format. The first byte of each packet is called the count byte and is part of the packet header. Depending on its value, it has the following significance:

COUNT	(HEX)	SIGNIFICANCE
0	($00)	*reserved for future use.*
1-127	($00 - $7F)	***repeat***: repeat the following byte count times. The total length of this packet is two bytes and decompresses to count bytes in the actual bitmap.
128	($80)	*reserved for future use.*
129-219	($81-$DB)	***unique***: use the next count-128 bytes literally. The total length of this packet is (count-128)+l or count-127 bytes and decompresses to count-128 bytes.
220	($DC)	*reserved for future use.*
221-255	($DD - $FF)	***bigcount***: the next byte is a bigcount value in the range 2 through 255. The following count-220 bytes comprise data in repeat and unique format that should be repeated bigcount times. The total length of this packet depends on the decompacted size of the ***repeat*** and ***unique*** packets. A bigcount cannot contain another ***bigcount***.

Decompaction Walkthrough

Given the following compacted data:

.byte 25, 0, 133, 240, 220, 10, 0, 7, 224, 4, 3, 10, 5, 3

The decompaction routine would interpret it like this:

25, 0

repeat: the decompaction routine encounters the count value 25. Since it is in the range 1-127, the following byte (0), is repeated 25 times:

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

133, 240, 220, 10, 0, 7

unique: the next packet begins with a count of 133,which is in the range 129-219. The next 133-128 = 4 bytes are used once each:

240, 220, 10, 0, 7

224, 4, 3, 10, 5, 3

bigcount: the final packet begins with a count of 24 which is in the range 221-255. This signals a two byte header and the following byte, the bigcount, is 4. These two bytes are interpreted to mean repeat the next 224-220 = 4 bytes four times. The next four bytes, however, are expected to be in the unique and repeat compacted formats. In this case, its 3,10 (repeat: 10 three times) and 5,3 (repeat: 3 five times), which in turn are repeated four times:

10, 10, 10 ,3, 3, 3, 3, 3, 10, 10, 10 ,3, 3, 3, 3, 3, 10, 10, 10 ,3, 3, 3, 3, 3, 10, 10, 10 ,3, 3, 3, 3, 3

Compacting Strategy
The easiest way to compact a bitmap image is to let geoPaint do it for you by cutting the image out as a photo scrap and pasting it directly into your geoProgrammer source code. Sometimes this method is impractical and you will want to compress images directly from within an application. The following subroutine can be used to compact bitmap data:

Example: **BitCompact**

Direct Screen Access and Block Copying

Direct Screen Access

One purpose of an operating system such as GEOS is to insulate the application from the peculiarities of the machine it is running on, allowing the programmer to worry more about how the application will function than how it will interact with the hardware. However, because of the complexity of GEOS graphics routines, it is sometimes necessary, for performance reasons, to bypass the operating system and manipulate the screen memory directly. Although this practice is not recommended — it increases portability problems, defeating much of the purpose of a GEOS — it is a reality. And with that in mind, Berkeley Softworks built routines into GEOS to facilitate direct screen access. The following routine exists in all versions of the Kernal:

• GetScanLine	Calculate the address of the first byte of a particular screen line.

Special Graphics Related Routines
GEOS provides a few graphics-related routines which don't fit nicely into any other category:

• GraphicsString	Execute a string of graphics commands.
• NormalizeX	Adjust an x-coordinate (under GEOS 128 only) to compensate for the higher-resolution 80-column mode
• SetNewMode	Change GEOS 128 graphics mode (40/80-column).

Icons, Menus, and Other Mouse Presses

When the user clicks the mouse button, GEOS determines whether the mouse pointer was positioned over an icon, a menu item, or some other region of the screen. GEOS has a unique method of handling a mouse press for each of these cases. If the user pressed on an icon, GEOS calls the appropriate icon event routine. If the user pressed on a menu, GEOS opens up a sub-menu or calls the appropriate menu event routine, whichever is applicable. And if the user pressed somewhere else, GEOS calls through **otherPressVector**, letting the application handle (or ignore) these "other" mouse presses.

Icons

When you open a disk by clicking on its picture, delete a file by dragging it to the trash can, or click on the CANCEL button in a dialog box, you are dealing with *icons*, small pictorial representations of program functions. A GEOS icon is a bitmapped image, whether die picture of a disk or a button-shaped rectangle, that allows the user to interact with the application. When the application enables icons, GEOS draws them to the screen and then keeps track of their positions. When the user clicks on an icon, an icon event is generated, and the application is given control with information concerning which icon was selected.

Icon Table Structure

The information for all active screen icons is stored in a data structure called the *icon table*. GEOS only deals with one icon table at a time. The icon table consists of an *icon table header* and a number of *icon entries*. The whole table is stored sequentially in memory with the header first, followed by the individual icon entries.

Icon Table Header

The icon table header is a four byte structure which tells GEOS how many icons to expect in the structure and where to position the mouse when the icons are enabled. It is in the following format:

Icon Table Header:

Index	Constant	Size	Description
+0	OFF_NM_ICNS	byte	Total number of icons in this table.
+1	OFF_IC_XMOUSE	word	Initial mouse x-position. If $0000, mouse position will not be altered.
+3	OFF_IC_YMOUSE	byte	Initial mouse y-position.

This first byte reflects the number of icon entries in the icon table (and, hence, the number of icons that can be displayed). The table can specify up to MAX_ICONS icons.

The next word (bytes 1 and 2) is an absolute screen x-coordinate and the following byte (byte 3) is an absolute screen y-coordinate. The mouse will be positioned to this coordinate when the icons are first displayed. If you do not want the mouse positioned, set the x-coordinate word to $0000, which will signal **DoIcons** to leave the mouse positions alone.

Icon Entries

Following the icon table header are the icon entries, one for each specified in the OFF_I_NUM byte in the icon table header. Each icon entry is a seven-byte structure in the following format:

Icon Entries:

Index	Constant	Size	Description
+0	**OFF_I_PIC**	word	Pointer to compacted bitmap picture data for this Icon. If set to $0000, icon is disabled.
+2	**OFF_I_X**	byte	Card x-position for icon bitmap.
+3	**OFF_I_Y**	byte	Y-position of icon bitmap.
+4	**OFF_I_WIDTH**	byte	Card width of icon bitmap.
+5	**OFF_I_HEIGHT**	byte	Pixel height of icon bitmap.
+6	**OFF_I_EVENT**	word	Pointer to icon event routine to call if this icon is selected.

Note: OFF_I_NEXT=8 Offset to Next Icon in structure if it exists.

The first word (**OFF_I_PIC**) is a pointer to the compacted bitmap data for the icon. The icon can be of any size (up to the full size of the screen). If this word is set to **NULL** ($0000), the icon is disabled.

The third byte (**OFF_I_X**) is the x byte-position of the icon. The x byte-position is the x-position in bytes. Icons are placed on the screen by **`BitmapUp`** and so must appear on an eight-pixel boundary. The byte-position can be calculated by dividing the pixel-position by eight (x_byte_position = x_pixel_position/8).

The fourth byte (**OFF_I_WIDTH**) is the pixel position of the top of the icon. The icon will be placed at (x_byte_position*8, y_pixel_position).

The next two bytes (**OFF_I_WIDTH** and **OFF_I_HEIGHT**) are the width in bytes and height in pixels, respectively. These values correspond to the geoProgrammer internal variables **picW** and **picH** when they are assigned immediately after a pasted icon image.

The final word (**OFF_I_EVENT**) is the address of the icon event handler associated with this icon.

Sample Icon Table

The following data block defines three icons which are placed near the middle of the screen. The mouse is positioned over the first icon:

```
***********************************************************************
; SAMPLE ICON TABLE
***********************************************************************
; Icon positions and bitmap data
I_SPACE = 1 ; space between our icons (in cards)
PaintIcon:
```

```
PAINTW = picW
PAINTH = picH
PAINTX = 16/8
PAINTY = 80
```

```
Writelcon:
```


```
WRITEW = picW
WRITEH = picH
WRITEX = PAINTX + PAINTW + I_SPACE
WRITEY = PAINTY

PublishIcon:
```

```
PUBLISHW    = picW
PUBLISHH    = picH
PUBLISHX    = WRITEX + WRITEW + I_SPACE
PUBLISHY    = WRITEY

;The actual icon data structure to pass to DoIcons follows
IconTable:

I_header:
      .byte NUMOFICONS                    ; number of icon entries
      .word (PAINTX*8) + (PAINTW*8/2)     ; position mouse over paint icon
      .byte PAINTY + PAINTH/2

I_entries:
PaintIStruct:
      .word Paintlcon                     ; pointer to bitmap
      .byte PAINTX, PAINTY                ; icon position
      .byte PAINTW, PAINTH                ; icon width, height
      .word PaintEvent                    ; event handler

WriteIStruct:
      .word Writelcon                     ; pointer to bitmap
      .byte WRITEX, WRITEY                ; icon position
      .byte WRITEW, WRITEH                ; icon width, height
      .word WriteEvent                    ; event handler

PublishIStruct:
      .word Publishlcon                   ; pointer to bitmap
      .byte PUBLISHX, PUBLISHY            ; icon position
      .byte PUBLISHW, PUBLISHH            ; icon width, height
      .word PublishEvent                  ; event handler

NUMOFICONS = (*-I_entries)/IRECSIZE       ; number of icons in table

;Dummy icon event routines which do nothing but return
PaintEvent:
WriteEvent:
PublishEvent:
      rts
```

Installing Icons

When an application is first loaded, GEOS will not have an active icon structure. GEOS must be given the address of the applications icon table before **`MainLoop`** can display and track the user's interaction with them. GEOS provides one routine for installing icons

- **DoIcons** Display and activate an icon table.

DoIcons draws the enabled icons and instructs **`MainLoop`** to begin watching for a single- or double-click on one. The icon table stays activated and enabled until the ICONS_ON_BIT of mouseOn is cleared or another icon table is installed by calling **`DoIcons`** with the address of a different icon structure. In either case, the old icons are not erased from the screen by GEOS.

`DoIcons` will draw to the foreground screen and background buffer depending on the value of **`dispBufferOn`**. Icons are usually permanent structures in a display and so often warrant being drawn to both screens. If icons are only drawn to the foreground screen, they will not be recovered after a menu or dialog box.

Example: **IconsUp**

Important: Due to a limitation in the icon-scanning code, the application must always install an icon table with at least one icon. If the application is not using icons, create a dummy icon table with one icon (see below).

```
;*****************************************************************
; NoIcons   Install a dummy icon table. For use in applications that
;           aren't using icons. Call early in the initialization of
;           the application, before returning to MainLoop.
;*****************************************************************
NoIcons:
      LoadW   r0,#DummyIconTable    ; point to dummy icon table
      jmp     DoIcons               ; install. Let DoIcons rts

DummyIconTable:
      .byte 1                       ; one icon
      .word $0000                   ; dummy mouse x (don't reposition)
      .byte $00                     ; dummy mouse y
      .word $0000                   ; bitmap pointer to $0000 (disabled)
      .byte $00                     ; dummy x-pos
      .byte $00                     ; dummy y-pos
      .byte 1,1                     ; dummy width and height
      .word $0000                   ; dummy event handler
```

`MainLoop` and Icon Event Handlers

When the user clicks the mouse button on an active icon, GEOS **`MainLoop`** will recognize this as an icon event and call the icon event handler associated with the particular icon. The icon event handler is given control with the number of the icon in r0L (the icon number is based on the icon's position in the table: the first icon is icon 0). Before the event handler is called, though, **`MainLoop`** might flash or invert the icon depending on which of the following values is in **`iconSelFlag`**:

Constants for **`iconSelFlag`**:

ST_NOTHING	$00	The icon event handler is immediately called; the icon image is untouched
ST_FLASH	$80	The icon is inverted for selectionFlash vblanks and then reverted to its normal state before the event handler is called.
ST_INVERT	$40	The icon is inverted (foreground screen image only) before the event handler is called. The event handler will usually want to revert the image before returning to **`MainLoop`** by calculating the bounding rectangle of the icon, loading **`dispBufferOn`** with **ST_WR_FORE**, and calling **InvertRectangle**.

Detecting Single- and Double-clicks on Icons

When the user first clicks on an icon, GEOS loads the global variable **`dblClickCount`** with the GEOS constant **CLICK_COUNT**. GEOS then calls the icon event handler with **r0H** set to **FALSE**, indicating a single-click. **`dblClickCount`** is decremented at interrupt level every vblank. If the icon event handler returns to **`MainLoop`** and the icons user again clicks on the icon before **`dblClickCount`** reaches zero, GEOS calls the icon event handler a second time with **r0H** set to **TRUE** to indicate a double-click.

Checking for a double-click or a single-click (but not both) on a particular icon is trivial: merely check **r0H**. If **r0H** is **TRUE** when you're looking for a single-click or its **FALSE** when you're looking for a double-click, then return to **`MainLoop`** immediately. Otherwise, process the click appropriately. This way, if the user single-clicks on an icon which requires double-clicking or double-clicks on an icon which requires single-clicking, the event will be ignored.

However, checking for both a double- or a single-click on the same icon (and performing different actions) is a bit more complicated because of the way double-clicks are processed: during the brief interval between the first and second clicks of a double-click, the icon event handler will be called with **r0H** set to **FALSE**, which will appear as a single-click; when the second press happens before **`dblClickCount`** hits zero, the icon event handler is called a second time with **r0H** set to **TRUE**, which will appear as a double-click. There is no simple way (using the GEOS double click facility) to distinguish a single-click which is part of a double-click from a single-click which stands alone.

There are two reliable ways to handle single- and double-click actions on icons: the additive function method and the polled mouse method. The additive function method relies on a simple single-click event which toggles some state in the application and a double-click event (usually more complicated) which happens in addition to the single-click event. The GEOS deskTop uses the additive function method for selecting (inverting) file icons on a single-click and selecting and opening them on a double-click. The icon event handler first checks the state of **r0H**. If it is **FALSE** (single-click) then the icon (and an associated selection flag) is inverted. If it is **TRUE** (double-click) then the file is opened. If the user single-clicks, the icon is merely inverted. If the user double-clicks, the icon is inverted (on the first click) and then processed as if opened (on the second click).

Example:

```
; ****************************************************************
; Icon double-click handler
; additive function method
; ****************************************************************
IconEvent1:
      lda   r0H                ; check double-click flag .
      bne   10$                ; branch if second click of a double-click
                               ; else, this is a single-click or the
                               ; first push of a double-click,
      jsr   InvertIcon         ; so just invert the selection
      bra   90$
10$
      jsr   OpenIcon           ; double-click detected, go process it
90$
      rts                      ; exit
```

The polled-mouse method can be used when the single-click and double-click functions are mutually exclusive. When a single-click is detected the icon event handler, rather than returning to **MainLoop** and letting GEOS manage the double-click, handles it manually by loading **dblClickCount** with a delay and watching **mouseData** for a release followed by a second click.

Example:

```
; ****************************************************************
; Icon double-click handler
; polled mouse method Open Icon
; ****************************************************************
IconEvent2:
 ;--- User pressed mouse once, start double-click counter going
      LoadB dblClickCount,#CLICK_COUNT    ; start delay

 ;--- Loop until double-click counter times-out or button is released
10$
      lda   dblClickCount                 ; check double-click timer
      beq   40$                           ; If timed-out, no double-click
      lda   mouseData                     ; Else, check for release
      bpl   10$                           ; loop until released

 ;--- mouse was released, loop until double-click counter times-out or
 ;--- button is pressed a second time.
20$
      lda   dblClickCount                 ; check double-click timer
      beq   30$                           ; If timed-out, no double-click
      lda   mouseData                     ; Else, check for second press
      bmi   20$                           ; loop until pressed
 ;--- Double-click detected (no single-click)
30$   jmp   DoDoubleClick                 ; do double-click stuff
 ;--- Single-click detected (no double-click)
40$   jmp   DoSingleClick                 ; do single-click stuff
```

> Note: These techniques for handling single- and double-clicks are described here as they pertain to icons; they are not directly applicable to applications that detect mouse clicks through otherPressVector. When control vectors through otherPressVector, the value in r0H is meaningless. For more information on otherPressVector, refer to "Other Mouse Presses" in this chapter.

Other Things to Know About Icons

Icon Releases and otherPressVector
When the user clicks on an active icon, **`MainLoop`** will call the proper icon event routine rather than vectoring through otherPressVector. However, the routine pointed to by otherPressVector will get called when the mouse is released. Applications that aren't using otherPressVector can disable this vectoring by storing a. $0000 into OtherPressVector ($0000 is actually its default value). Applications that depend on otherPressVector, however, can check mouseData and ignore all releases.

Example:

```
;OtherPressVector routine that ignores releases (high bit of mouseData is set on
releases)
MyOtherPress:                          ; control comes here from otherPressVector
            lda   mouseData            ; check state of the mouse button
            bmi   90$                  ; ignore it if it's a release
            jsr   PressDown            ; otherwise process the press
90$:
            rts                        ; exit
```

For more information on **otherPressVector**, refer to "Other Mouse Presses" in this chapter.

Icon Precedence
GEOS draws icons sequentially. Therefore, if icons overlap, the ones which are drawn later will be drawn on top. When the user clicks somewhere on the screen, GEOS scans the icon table in this same order, looking for an icon whose rectangular boundaries enclose the coordinates of the mouse pointer. If more than one icon occupies the coordinate position, the icon that is defined first in the icon table (and therefore drawn on bottom) will be given the icon event. If an active menu and an icon overlap, the menu will always be given precedence.

Disabling Icons
An application can disable an icon in the current icon structure by clearing the **OFF_I_PIC** word of the icon (setting it to $0000). If an icon is disabled prior to a call to **DoIcons**, the icon will not be drawn. If an icon is disabled after the call to **DoIcons**, the icon will remain on the screen but will be ignored during the icon scan. The application can reenable the icon by restoring the **OFF_I_PIC** word to its original value. (Actually, any non-zero value will do because reenabling an icon does not redraw it, it only restores the coordinates to **`MainLoop`**'s active search list.)

GEOS 128 Icon Doubling

As with bitmaps, special flags in the icon data structure can be set to automatically double the xposition and/or icon width when GEOS 128 is running in 80-column mode. To have an position icon's x-position automatically doubled in 80-column mode, bitwise-or the **OFF_I_X** parameter with **DOUBLE_B**. To double an icon's width in 80-column mode, bitwise-or the **OFF_I_WIDTH** parameter with **DOUBLE_B**. These bits will be ignored when GEOS 128 is running in 40-column mode. Do not, however, use these doubling bits when running under GEOS 64. GEOS 64 will try to treat the doubling bit as part of the coordinate or width value rather than a special-case flag. For more information, refer to "GEOS 128 X-position and Bitmap Doubling" in Chapter "Graphics Routines".

Example:

```
; ******************************************************************
; SAMPLE GEOS 128 ICON TABLE THAT USES AUTOMATIC DOUBLING FEATURE
; using compiler flags for conditional assembly between C128 and C64
; Note: You can build programs that work on both the 128 in 80cols
; and the 64.
; ******************************************************************

C128=TRUE
C64=FALSE
.if !C128
      .echo Error: cannot assemble GEOS 128 specific code without C128 flag set
.else
PaintIcon:
```

```
PAINTW = picW
PAINTH = picH
PAINTX = 16/8
PAINTY = 80

;The actual icon data structure to pass to DoIcons follows
IconTable:
I_header:
   .byte NUMOFICONS
   .word ((PAINTX*8) + (PAINTW*8/2)) | DOUBLE_W  ; position mouse over paint icon
   .byte PAINTY + PAINTH/2

I_entries:
PaintIStruct:
   .word PaintIcon                       ; pointer to bitmap
   .byte PAINTX | DOUBLE_B               ; x card position (dbl in 80-column mode)
   .byte PAINTY                          ; y-position
   .byte PAINTW | DOUBLE_B               ; icon width (dbl in 80-column mode)
   .byte PAINTH                          ; icon height
   .word PaintEvent                      ; event handler
NUMOFICONS - (*-I_entries)/ OFF_I_NEXT   ;number of icons in table

;Dummy icon event routines which do nothing but return
PaintEvent:
   rts
.endif
```

Menus

Menus, one of the most common and powerful user-interface facilities provided by GEOS, allow the application to offer lists of items and options to the user. The familiar menus of the GEOS desktop, for example, provide options for selecting desk accessories, manipulating files, copying disks, and opening applications. Virtually every GEOS-based program will take advantage of these capabilities, providing a consistent interface across applications.

GEOS menus come in two flavors: horizontal and vertical. The main menu, the menu which is always displayed, is usually of the horizontal type and is typically placed at the top of the screen. Each selection in the main menu usually has a corresponding vertical sub-menu that opens up when an item in the main menu is chosen. These sub-menus can contain items that trigger the application to perform some action. They can also lead to further levels of sub-menus. For example, a horizontal main menu item can open up to a vertical menu, which can have items which then open up other horizontal sub-menus, which can then lead to other vertical menus, and so on.

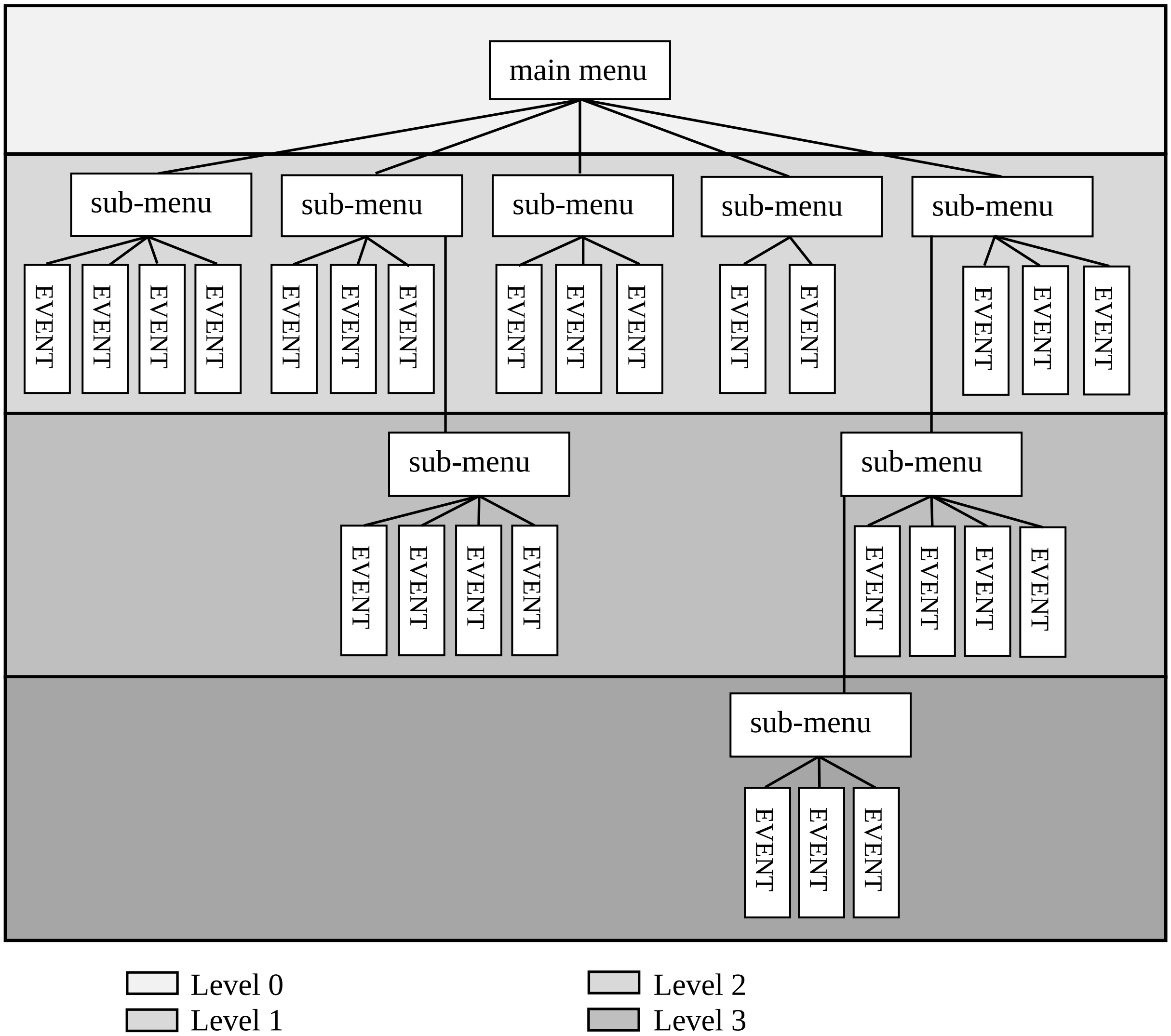
Division of Labor with Menus

GEOS divides the labor of handling menus between itself and the application. The GEOS Kernal handles all of the user's interaction with the menus. This includes drawing the menu items, opening up necessary sub-menus, and restoring the Screen area from the background buffer when the menus are retracted. **`MainLoop`** manages the menus, keeping track of which items the user selects. If the user moves off of the menu area without making a selection, GEOS automatically retracts the menus without alerting the application.

If the user selects a menu item which generates a menu event, the application's menu event handler is called with the menus left open. Leaving the menus open allows the application to choose when and how to retract them: all the way back to the main menu, up one or more levels (for multiple sub-menus), or up no levels (keeping the current menu open). This lets the application choose the menu level which is given control upon return, thereby allowing multiple selections from a sub menu without forcing the user to repeatedly traverse the full menu tree for each option.

Menu Data Structure

The main menu, all its sub-menus, their individual selectable items, and various attributes associated with each menu and each item are all stored in a hierarchical data structure called the menu tree. Conceptually, a menu tree with multiple sub-menus might have the following layout:

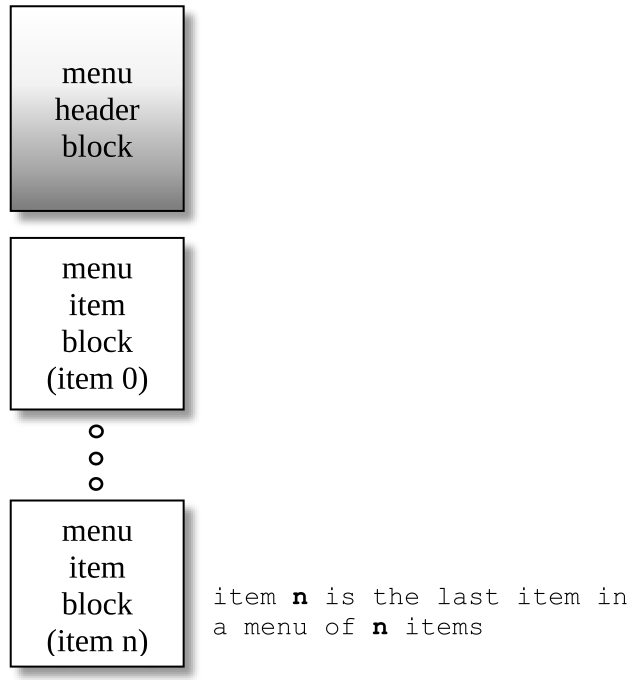


Sample Menu Tree

The main menu (or level 0) is the first element in the tree; it is the menu that is always displayed while menus are enabled. Each item in a main menu will usually point to a secondary menu or submenu. Items in these submenus can point to events (alerts to the application that an item was selected) or they can point to additional submenus. Menus are linked together by address pointers.

Sub-menus are sometimes referred to as child menus, and the menu which spawned the sub-menu as its parent. Sub-menus can be nested to a depth determined by the GEOS constant MAX_M NESTING, which reflects the internal variable space allocated to menus. The depth or level of the current menu can be determined by the GEOS variable menuNumber, which can range from 0 to (MAX_M_NESTING-1)

.In memory, all menus, whether the main menu or its children, are stored in the same basic menu structure format Each menu is comprised of a single menu header block followed by a number of menu item blocks (one for each selectable item in the menu):



Menu/Sub-menu structure

Menu/Sub-menu Header

The menu header is a seven-byte structure that specifies the size and location of the menu (How big is the rectangle that surrounds the menu and where should the menu be drawn?), any attributes that affect the entire menu (Is it a vertical or horizontal menu?), and the number of selectable items in the menu. The header is in the following format:

Menu/Sub-menu Table Header:

Index	Constant	Size	Description
+0	OFF_M_Y_TOP	byte	Top edge of menu rectangle (yl pixel position).
+1	OFF_M_Y_BOT	byte	Bottom edge of menu rectangle (y2 pixel position).
+2	OFF_M_X_LEFT	word	Left edge of menu rectangle (xl pixel position).
+4	OFF_M_X_RIGHT	word	Right edge of menu rectangle (x2 pixel position).
+6	OFF_NUM_M_ITEMS	byte	Menu type bitwise-or'ed with number of items in this menu/sub-menu.

The first six bytes specify the screen location and size of the menu with the positions of the bounding rectangle in pixel positions. The x-positions are word (two-byte) values and the y positions are byte values. These values are absolute screen pixel positions. The size of the bounding rectangle depends on the number of menu items and the size of text strings within the menu. The height of the rectangle can be calculated with the constant M_HEIGHT: a horizontal menu is always a height of M_HEIGHT, and a vertical menu is a height of the number of menu items multiplied by M_HEIGHT. For example, the height of a vertical menu with seven items would be 7*M_HEIGHT. The width of a menu is more difficult to calculate because it depends on the length of the individual text strings. It is best to use a large number for this dimension and adjust it to a smaller size if necessary.

Important: GEOS 64 and GEOS 128 before version 2.0 do not correctly handle menus that extend beyond an x-position of 255.

All menus and sub-menus are positioned independently. This means that the main menu need not be at the top of the screen (it can be inside a window, for example), and sub-menus need not be adjacent to their parent menus (although that is where you will usually want them). You can experiment with the flexibility of menu positioning to customize your applications.

The seventh byte is the attribute byte. It is the number of selectable items in the menu bitwise-or'ed with any menu type flags. A menu can have as many as MAX_M_ITEMS selectable menu items.

Menu/Sub-menu Types (use in attribute byte):

Constant	Description
HORIZONTAL	Arrange menu items in this menu/sub-menu horizontally.
VERTICAL	Arrange menu items in this menu/sub-menu vertically.
CONSTRAINED	Constrain the mouse to the menu/sub-menu. If the menu is a sub-menu, the mouse can still be moved off to the parent menu (off the top of a vertical sub-menu or off the left of a horizontal menu).
UNCONSTRAINED	Do not constrain the mouse to the menu/sub-menu. If the user moves off of the menu, GEOS will retract it.

Bitwise Breakdown of the Attribute Byte:

7	6	5	4	3	2	1	0
b7	b6	b5-b0					

b7	orientation:	1= vertical;	0 = horizontal,
b6	constrained:	1 = yes;	0 = no.
b5-b0	number of items in menu/sub-menu (up to MAX_M_TEMS).		

Some of the menu types are obviously mutually exclusive: you can't, for example, make a menu both vertical and horizontal, nor simultaneously constrained and unconstrained.

A vertical, unconstrained menu with seven selectable items would have an attribute byte of:

```
.byte (7 | VERTICAL | UN_CONSTRAINED)
```

A horizontal, constrained menu with 11 selectable items would have an attribute byte of:

```
.byte (11 |  HORIZONTAL | CONSTRAINED)
```

Most sub-menus are unconstrained: if the user moves the pointer off the sub-menu, all opened menus are retracted as if **`GotoFirstMenu`** had been called. A constrained menu, on the other hand, restricts the pointer from moving off the menu area from all but one side. A constrained menu will only allow the pointer to move off the side leading back to where it expects the parent menu to be: off the top for a vertical sub-menu and off the left for a horizontal sub-menu. If the user moves off of a constrained menu (in the only available direction), the current sub-menu is retracted and the parent menu becomes active as if **DoPreviousMenu** had been called.

NOTE:	The constrain option is only applicable to sub-menus — if the **CONSTRAINED** flag is set in the main menu (level 0), the option will have no effect.

Menu Item Structure

For each selectable item in a menu (the number items is specified in the header) there is a five-byte item structure. These item structures follow the menu header in memory. The first item represents the first menu selection (top- or leftmost), the second, the second, and so on. Each item structure specifies the text that will appear in the menu, what happens when the item is selected (Will it generate an event or a sub-menu?), and the appropriate event routine or sub-menu. Each menu item is in the following format:

Menu Item:

Index	Constant	Size	Description
+0	OFF_TEXT_ITEM	word	Pointer to null-terminated text string for this menu item.
+2	OFF TYPE ITEM	byte	Selection type (sub-menu, event, dynamic sub-menu).
+3	OFF_POINTER_ITEM	word	Pointer to sub-menu data structure, event routine, or dynamic sub-menu routine, depending on selection type.

The first word of the item is a pointer to the text that will be placed in the menu. The text is expected to be null-terminated (the last byte should be $00 or NULL). If the menu rectangle specified in the header is not wide enough to contain the entire text string, the text will be clipped at the right edge when the menu is drawn.

The byte following the text pointer (the third byte) is an item type indicator. Each selectable item can either be an action, a sub-menu, or a dynamic sub-menu selection. An action8type item generates a menu event from **MainLoop**. A sub-menu type item automatically opens up a sub menu structure. And a dynamic sub-menu type selection opens up a sub-menu, but before it does, it calls an application's routine. Dynamic sub-menus arc useful for modifying a menu structure on the fly. For example, a point size sub-menu, such as those used in geoWrite, can be changed dynamically when a new font is selected. When the user chooses the font item, the dynamic sub menu routine checks the list of available point sizes and builds out the point size sub-menu based on its findings. The following table summarizes the three menu item types:

Types of Menu Items (for use in item type byte):

Constant	Description
SUB_MENU	This menu item leads to a sub-menu. The OFF_POINTER_ITEM is a pointer to the sub-menu data structure (points to first byte of "a menu/sub-menu header).
DYN_SUB_MENU	This menu item is a dynamic sub-menu. The OFF_POINTER_ITEM is a pointer to a dynamic sub-menu routine that is called *before* the menu is actually drawn. The dynamic sub-menu routine can do any necessary preprocessing and return with r0 containing a pointer to a sub-menu data structure or $0000 to ignore the selection.
MENU_ACTION	This menu item generates an event. The OFF_POINTER_ITEM is a pointer to the event routine that will to call.

structures
dialog/Icons/Menus/Graphics

DIALOG:

Note[2]: The first entry in a DB table is a command byte defining its position. This can either be a byte indicating a default position for the DB, **DEF_DB_POS (%10000000)**, or a byte indicating a user defined position, **SET_DB_POS (%00000000)** which must be followed by the position information.

The position command byte is or'ed with a system pattern number to be used to fill in a shadow box. The shadow box is a rectangle of the same dimensions as the DB and is filled with one of the system patterns. The shadow box appears underneath the Dialog Box, Offset 1 card right and 1 card down.

```
Start of Default Dialog      Start of Custom Size Dialog
--------------------------   ----------------------------------
.byte DEF_DB_POS | pattern   .byte SET_DB_POS | pattern
                             .byte top        ;(0-199)
                             .byte bottom     ;(0-199)
                             .word left       ;(0-319 or 0-639)
                             .word right      ;(0-319 or 0-639)
```

Note[1]: standard window size: columns 64-255
rows 32-127

Note[1]: If the shadow pattern is zero, then no shadow is drawn.

Note[1]: Icon descriptors are stored in a table at $880C

Note[3]: Maximum # of Dialog Icons is 8. This can be worked around by drawing your own images and detecting mouse clicks over the images.

Note[1]: The following is a list of global variables stored by the window processor:

curPattern	**string**	**baselineOffset**	**curSetWidth**
curHeight	**curIndexTable**	**cardDataPntr**	**currentMode**
dispBufferOn	**mouseOn**	**msePicPtr**	**windowTop**
windowBottom	**leftMargin**	**rightMargin**	**appMain**
intTopVector	**ioBotVector**	**mouseVector**	**keyVector**
inputVector	**mouseFaultVec**	**otherPressVector**	**alarmTmtVector**
BRKVector	**RecoverVector**	**selectionFlash**	**alphaFlag**
iconSelFlag	**faultData**	**menuNumber**	**mouseTop**
mouseBottom	**mouseLeft**	**mouseRight**	**stringX,stringY**

I/O address's $D000-$D010 $D01B-$D01D $D025-$D026 $D015 $D028-$D02E

Position Commands:

After the position byte (or bytes) may appear a number of icon or command bytes. Most require position coordinates. The x and y positions are an offset from the upper left corner of the DB.

```
Icons x position uses bytes (cards)      0-40  x_boffset
Text  x position uses pixels             0-319 x_poffset
y position is always in pixels           0-199 y_offset
```

Note[3]: GEOS 128 always doubles the x positions in a dialog box when the system is in 80 column mode. Do not try to use **DOUBLE_W/DOUBLE_B** as this will be a VERY large x coordinates. **DBUSERICON** Structures DO need **DOUBLE_B** for width if the user icon is not a native 80 col icon.

Note[3]: Custom Dialog X-Cords DO require Doubling (or native 0-640 cords)

Dialog Box Icons

Icon	Value	Example	Description
OK	1	.byte OK .byte x_boffset .byte y_offset	Draw OK Icon
CANCEL	2		Draw CANCEL Icon
YES	3		etc...
NO	4		
OPEN	5		
DISK	6		
NOT-USED	7-10		Marked for future use. When is the future?

Dialog Commands

Command	Value	Example	Description
DBTXTSTR	11	.byte **DBTXTSTR** .byte x_poffset .byte y_offset .word ptrtTextStr	Put tTextStr
DBVARSTR	12	.byte DBVARSTR .byte x_poffset .byte y_offset .byte zPgPtr	Put text @@zPgPtr zPgPtr is an address of a zero page ptr to string
DBGETSTRING	13	.byte **DBGETSTRING** .byte x_poffset .byte y_offset .word ZPgPtr .byte BUFFERSIZE	Get typed user input. ZpgPtr points to address of a buffer to use for the input that is BUFFERSIZE bytes.
SBSYSOPV	14	.byte **SBSYSOPV**	Closes DB when the mouse is pressed anywhere other then over an Icon
DBGRPHSTR	15	.byte **DBGRPHSTR** .word gGraphicsString	Draws a **GraphicsString**
DBGetFileS	16	.byte **DBGetFileS** .byte x_boffset .byte y_offset	Presents a File Selection box for the user to pick from.
DBOPVEC	17	.byte **DBOPVEC** .word MsePressVector	Vector to call when mouse button is pressed.
DBUSERICON	18	.byte **DBUSERICON** .byte x_boffset .byte y_offset .word UserIcon	UserIcon Table .word ptrIconData .word NULL .byte width in bytes .byte Height in Pixels .word ptrIconAction Note: (width | **DOUBLE_B** for 128)
DB_USR_ROUT	19	.byte **DB_USR_ROUT** .WORD User_Vector	Call User_Vector after the DB has been drawn.
NULL	0	.byte NULL	Ends the Dialog Box Definition

Menu

Menu/Sub Menu Header

		Size	
+0	OFF_M_Y_TOP	byte	Top edge of menu rectangle (yl pixel position).
+1	OFF_M_Y_BOT	byte	Bottom edge of menu rectangle (y2 pixel position).
+2	OFF_M_X_LEFT	word	Left edge of menu rectangle (xl pixel position).
+4	OFF_M_X_RIGHT	word	Right edge of menu rectangle (x2 pixel position).
+6	OFF_NUM_M_ITEMS	byte	Menu type bitwise-or'ed with number of items in this menu/sub-menu.

Menu/Sub-menu Types (use in attribute byte):

HORIZONTAL	Arrange menu items in this menu/sub-menu horizontally.
VERTICAL	Arrange menu items in this menu/sub-menu vertically.
CONSTRAINED	Constrain the mouse to the menu/sub-menu. If the menu is a sub-menu, the mouse can still be moved off to the parent menu (off the top of a vertical sub menu or off the left of a horizontal menu).
UNCONSTRAINED	Do not constrain the mouse to the menu/sub-menu. If the user moves off of the menu, GEOS will retract it

Bitwise Breakdown of the Attribute Byte:

7	6	5 4 3 2 1 0
b7	b6	b5-b0

```
b7    orientation: 1 = vertical;    0 = horizontal,
b6    constrained: 1 = yes;         0 = no.
```
b5-b0 number of items in menu/sub-menu (up to **MAX_M_ITEMS**).

structures

disk

Directory Entry:

Offset	Constant	Size	Description
$00	**OFF_CFILE_TYPE**	1	DOS file type Bit 7 1=File Closed/Normal State Bit 6 Write Protect bit **ST_WR_PR** %01000000 Bit 0-2 Commodore file Type DEL = 0 ;deleted SEQ = 1 ;sequential PRG = 2 ;program USR = 3 ;user (GEOS) REL = 4 ;relative file. Invalid in GEOS CBM = 5 ;BAM Protection.
$01	**OFF_INDEX_PTR** **OFF_DE_TR_SC**	2	Index table pointer (VLIR file T/S) track/sector for file's 1st data block
$03	**OFF_FNAME**	16	File name padded with hard spaces $A0
$13	**OFF_GHDR_PTR**	2	track/sector of GEOS Header block
$15	**OFF_GSTRUC_TYPE**	1	GEOS file structure type **SEQUENTIAL**=0 **VLIR**=1
$16	**OFF_GFILE_TYPE**	1	GEOS file type **NOT_GEOS**=0 ;C-64 file No Header **BASIC**=1 ;C-64 Basic w/Header **ASSEMBLY**=2 ;C-64 Assembly w/Header **DATA**=3 ;C-64 DATA File w/Header **SYSTEM**=4 ;GEOS System File **DESK_ACC**=5 ;GEOS desk accessory **APPLICATION**=6 ;GEOS application **APPL_DATA**=7 ;GEOS data file **FONT**=8 ;GEOS font **PRINTER**=9 ;GEOS Print Driver **INPUT_DEVICE**=10 ;GEOS mouse etc. **DISK_DEVICE**=11 ;GEOS DISK driver **SYSTEM_BOOT**=12 ;GEOS boot file **TEMPORARY**=13 ;GEOS Swap File (The deskTop will automatically delete all temporary files when opening a disk.) **AUTO_EXEC**=14 ;Application to automatically be ran just after booting, but before deskTop runs. **INPUT_128**=15 ;128 Input driver
$17	**OFF_YEAR**	5	Y/M/D/H/M
$1C	**OFF_SIZE**	2	File Size in blocks

Header Block:

Offset	Constant	Size	Description
$00		2	$00,$FF When Creating a file with Save file, this location holds a word pointer to a buffer containing the filename
$02	O_GHIC_WIDTH	1	width in bytes of file icon
$03	O_GHIC_HEIGHT	1	height of file icon in pixels
$04	O_GHIC_PIC	64	Icon Data
$44	O_GHCMDR_TYPE	1	Commodore File Type DEL = 0 ;deleted SEQ = 1 ;sequential PRG = 2 ;program USR = 3 ;user (GEOS) REL = 4 ;relative file. Invalid in GEOS CBM = 5 ;BAM Protection.
$45	O_GHGEOS_TYPE	1	GEOS file type NOT_GEOS=0 ;C-64 file No Header BASIC=1 ;C-64 Basic w/Header ASSEMBLY=2 ;C-64 Assembly w/Header DATA=3 ;C-64 DATA File w/Header SYSTEM=4 ;GEOS System File DESK_ACC=5 ;GEOS desk accessory APPLICATION=6 ;GEOS application APPL_DATA=7 ;GEOS data file FONT=8 ;GEOS font PRINTER=9 ;GEOS Print Driver INPUT_DEVICE=10 ;GEOS mouse etc. DISK_DEVICE=11 ;GEOS DISK driver SYSTEM_BOOT=12 ;GEOS boot file TEMPORARY=13 ;GEOS Swap File (The deskTop will automatically delete all temporary files when opening a disk.) AUTO_EXEC=14 ;Application to automatically be ran just after booting, but before deskTop runs. INPUT_128=15 ;128 Input driver
$46	O_GHSTR_TYPE	1	GEOS file structure type
$47	O_GHST_ADDR	2	Start address of file
$49	O_GHEND_ADDR	2	end address of file (Only valid for Desk Accessories)
$4B	O_GHST_VEC	2	Application Initialization vector
$4D	O_GHFNAME	12 4 3	Permanent filename Version Example: V1.0 Always 3 Zeros.
$60	O_128_FLAGS	1	OS Combability Flag bit7 bit 6 0 0 $00 64/128 40-column mode only 0 1 $40 64/128 40 and 80-column modes 1 0 $80 64 Only. Does not run under GEOS 128 1 1 $C0 128 80-column mode only. Does not run under GEOS 64
$61	O_GH_AUTHOR	20	Application Author Name
$75	O_GHP_FNAME	20	permanent filename of parent application. Used by Data Files.

structures

$89	O_GHAPDAT	23	Data Area for Application Use
$A0	O_GHINFO_TXT	96	Null Terminated File Notes. Shows in Information box for file.

```
                  = 97        ;20 bytes: author's name (only if is applic.)
                  = $A0       ;offset to notes that are stored with the file
                              ;and edited in the deskTop "get info" box.

if file is an application data file:

O_GHP_DISK        = 97        ;20 bytes: disk name of parent application's
                              ;disk.

O_GHP_FNAME       = 117       ;20 bytes: permanent filename of parent
                              ;application.
```

Appendex

Contents

1. atoms
2. hardware
3. memory maps
4. macros

atoms

Categories

Identifier	Category
bit	bit operations
br	branching
cmp	Comparisons
flow	Alters flow of logic
math	Math
hw	Hardware
size	Code Base Reduction
text	Text Operations
util	Utility

Sources

Identifier	Source
gP1	geoProgrammer1.1
gD	geoDebugger
gW	geoWrite 128
gP'	geoProgrammer' 2.0
HHG	Created by PBM to perform actions for HHG2G Macros that were not defined in geoProgrammer1.1. Example: macro **bgt** is used in HHG2. But is not in geoProgrammer1.1. Macro logic was obvious so it was created here for use in the examples.
	Other sources will be added as used

name	Description		
DoDlg	Wrapper for DoDlgBox to reduce codebase size	gP	size
Lower	Convert character to Lowercase	gP'	text
Upper	Convert character to uppercase	gP'	text

atoms

blank

DoDlg: size

Name: **DoDlg**
Filename: _DoDlg
Category size
Source: geoProgrammer

Purpose: Reduce Foot Print of application that uses multiple dialog boxes.

Pass: a DBL - Low Part of dialog Box Address
x DBH - High byte of dialog Box Address

Returns: Same as **DoDlgBox**

Destroys: Same as **DoDlgBox**

Description:

Note: Normal way to call a dialog box is using **LoadW r0,**dbTable.
This takes 8 bytes to load **r0** before calling **DoDlgBox** and only 4 bytes to load a and x with the address bytes.

It takes 2 uses of DoDlg for it to cut a profit.
DoDlg size = 7 bytes.
Savings per use. 4 bytes.

Example:

```
        ...
        lda   #[dbTable
        ldx   #]dbTable
        jsr   DoDlgBox
        ...
```

Body:

```
DoDlg:
        sta   r0
        stx   r0H
        jmp   DoDlgBox
```

See also:

atoms

Lower: text

Name: **Lower**
Filename: _Lower
Category text
Source: geoProgrammer'

Purpose: Convert character to uppercase

Pass: accumulator CHAR - Character to process

Returns: If value in accumulator is a lowercase letter
returns uppercase of that letter.
otherwise returns accumulator unchanged

Destroys: Nothing

Description:

Note: †Carry is known to be clear at that point. no need to clc prior to the adc.

Example:

Body:

```
    Lower:
        cmp   #'A'              ; If character < 'A' then exit.
        bcc   14$
        cmp   #'Z'+1            ; If character < 'Z' then exit.
        bcs   10$
        adc   #('a'-'A')        ; Convert to Lower Case†
    10$
        rts
```

See also: Upper

Upper: text

Name: Upper
Filename: _Upper
Category text
Source: geoProgrammer'

Purpose: Convert character to uppercase

Returns: If value in accumulator is an uppercase letter
returns uppercase of that letter.
otherwise returns accumulator unchanged

Destroys: Nothing

Pass: accumulator CHAR - Character to process

Note: †Carry is known to be clear at that point. The -1 is to compensate for the additional +1 subtraction caused by the cleared carry. This uses assembler time to save runtime bytes (1) and cycles (2) by removing the need for the sec instruction.

Example: **KeyTrap**

Body:

```
    Upper:
          cmp   #'a'            ; If character < 'a' then exit.
          bcc   90$             ;
          cmp   #'z'+1          ; if character > 'z' then exit.
          bcs   90$             ;
          sbc   #('a'-'A') -1   ; Convert to Upper Case†
    90$
          rts
```

See also: Lower

hardware

----
dialog

hardware

6510 data register: (64,128) 01

CPU_DATA

```
;Machine Power on Default          KRNL_BAS_IO_IN
;GEOS Default                      RAM_64K
;GEOS During serial I/O            IO_IN

RAM_64K=$30            ;%11 0100      ;64K RAM
KRNL_BS=$33            ;%11 0011      ;Kernal + basic
IO_IN=$35              ;%11 0101      ;60K RAM, 4K I/O space in
KRNL_IO_IN=$36         ;%11 0110      ;Kernal + I/O
KRNL_BAS_IO_IN=$37     ;%11 0111      ;Kernal + basic + IO
```

	KRNL_BAS_IO_IN	RAM_64K	**IO_IN**
FFFF			
E000	8k KERNAL ROM	8K RAM	8K RAM
D000	I/O	4K RAM	I/O
C000	4K RAM	4K RAM	4K RAM
A000	8K BASIC	8K RAM	8K RAM
0100	24K RAM	24K RAM	24K RAM
0	Zero Page	Zero Page	Zero Page

See also:

17XX RAM Expansion:

```
EXP_BASE:
$DF00: STATUS REGISTER
        Bit 7:  INTERRUPT PENDING  (1 = interrupt waiting to be served)
                Not Used by GEOS
        Bit 6:  END OF BLOCK  (1 = transfer complete)
                unnecessary
        Bit 5:  FAULT  (1 = block verify error)
                Set if a difference between C64- and REU-memory areas was found
                during a compare-command.
        Bit 4:  SIZE  (1 = 256 KB) set on 1764 and 1750 and clear on 1700.
        Bits 3..0: VERSION     0

$DF01: COMMAND REGISTER. Write to this register to start operation
        Bit 7:  EXECUTE  (1 = transfer per current configuration)
                Set this bit to execute a command.
        Bit 6:  reserved  (normally 0)
        Bit 5:  LOAD  (1 = enable autoload option)
                With autoload enabled the address and length registers (see
                below) will be unchanged after a command execution.
                Otherwise the address registers will be counted up to the
                address off the last accessed byte of a DMA + 1,
                and the length register will be changed (normally to 1).
        Bit 4:   FF00
                If this bit is set command execution starts immediately
                after setting the command register.
                Otherwise command execution is delayed until write access to
                memory position $FF00
        Bits 3..2: reserved  (normally 0)
        Bits 1..0: TRANSFER TYPE
           00 = transfer C64 -> REU
           01 = transfer REU -> C64
           10 = swap C64 <-> REU
           11 = compare C64 - REU

$DF02:  .word   C64 BASE ADDRESS
$DF04:  .word   REU BASE ADDRESS
$DF05:  .byte   BANK
$DF07:  .word   Transfer Size
$DF09:  Interrupt Mask Register.       Not used by GEOS
$DF0A:  Address Control Register.
        Bit 7:     C64 ADDRESS CONTROL  (1 = fix C64 address)
        Bit 6:     REU ADDRESS CONTROL  (1 = fix REU address)
        Bits 5..0: unused
```

Note: By using a fixed address in the REU as a source you can very quickly initialize large blocks of ram

Full Reference
http://www.zimmers.net/anonftp/pub/cbm/documents/chipdata/programming.reu
Richard Hable

C128 MMU:

```
Configuration Register
MMUReg=$FF00       ;Mirror of D500. FF00 is Always Visible
```

```
;MMUReg Bits
;- Bit 0            ;Zone 5    $D000-DFFF
MIO    =%0          ;I/O
MCROM  =%1          ;Character ROM
;- Bit 1            ;Zone 2    ;$4000-7FFF
MBASIC =%00         ;Basic ROM
MEXTROM=%10         ;External Function ROM
;- Bits 2,3         ;Zone 3    $8000-BFFF
MUBASIC=%0000       ;Basic ROM
MUIROM =%0100       ;Internal Function ROM
MUEROM =%1000       ;External Function ROM
MURAM  =%1100       ;RAM
;- Bits 4,5         ;Zone 4    $C000-CFFF,
                               $E000-FEFF
MHKERNAL = %000000  ;KERNAL ROM
MHIROM=%010000      ;Internal Function ROM
MHEROM=%100000      ;External Function ROM
MHERAM=%110000      ;RAM
;- Bits 6,7         ;Bank Select
MBANK0=%00000000    ;Bank 0
MBANK1=%01000000    ;Bank 1
MBANK2=%10000000    ;Bank 2
MBANK3=%11000000    ;Bank 2
```

Configuration Register	
Bits	Description
7-6	Bank Select 00 Bank 0 01 Bank 1 10 Bank 2 11 Bank 3
5-4	C000-CFFF,E000-EFFF 00 Kernal ROM 01 Internal Function ROM 10 External Function ROM 11 RAM
3-2	8000-BFFF 00 Basic ROM 01 Internal Function ROM 10 External Function ROM 11 RAM
1	4000-7FFF 0 BASIC ROM low 1 RAM
0	D000-DFFF 0 I/O 1 1 RAM or Character ROM

```
BANK_0 = MBANK0| MHERAM | MURAM | MEXTROM | MCROM     ;No ROMs, RAM 0

BANK_0 =%00111111 ;No ROMs, RAM 0
BANK_1 =%01111111 ;No ROMs, RAM 1
BANK_2 =%10111111 ;No ROMs, RAM 2   ;Requires 512k expanded 128.
                                    ;Otherwise same as bank 0
BANK_3 =%11111111 ;No ROMs, RAM 3   ;Requires 512k expanded 128.
                                    ;Otherwise same as bank 0

BANK_4  =MBANK0|MHIROM|MUIROM|MEXTROM|MIO
BANK_5  =MBANK1|MHIROM|MUIROM|MEXTROM|MIO
BANK_6  =MBANK2|MHIROM|MUIROM|MEXTROM|MIO
BANK_7  =MBANK3|MHIROM|MUIROM|MEXTROM|MIO

BANK_8  =MBANK0|MHEROM|MUEROM|MEXTROM|MIO
BANK_9  =MBANK1|MHEROM|MUEROM|MEXTROM|MIO
BANK_10 =MBANK2|MHEROM|MUEROM|MEXTROM|MIO
BANK_11 =MBANK3|MHEROM|MUEROM|MEXTROM|MIO

BANK_12=%00000110 ;int function ROM, Kernal and IO, RAM 0
BANK_13=%00001010 ;
BANK_14=%00000001 ;all ROMs, char ROM ram 0
BANK_15=%00000000 ;all ROMs, RAM0 power on default
BANK_99=$00001110 ;IO, KERNAL, RAM 0 48K
```

Ram Configuration Register

```
MMURCR=$FF06      ;Mirror of D506. FF06 is Always Visible


;MMUReg Bits
;- Bit 0             ;Zone 5    $D000-DFFF
MIO    =%0           ;I/O
MCROM  =%1           ;Character ROM
;- Bit 1             ;Zone 2    ;$4000-7FFF
MBASIC =%00          ;Basic ROM
MEXTROM=%10          ;External Function ROM
;- Bits 2,3          ;Zone 3    $8000-BFFF
MUBASIC=%0000        ;Basic ROM
MUIROM =%0100        ;Internal Function ROM
MUEROM =%1000        ;External Function ROM
MURAM  =%1100        ;RAM
;- Bits 4,5          ;Zone 4    $C000-CFFF,
                                $E000-FEFF
MHKERNAL = %000000 ;KERNAL ROM
MHIROM=%010000       ;Internal Function ROM
MHEROM=%100000       ;External Function ROM
MHERAM=%110000       ;RAM
;- Bits 6,7          ;Bank Select
MBANK0=%00000000     ;Bank 0
MBANK1=%01000000     ;Bank 1
MBANK2=%10000000     ;Bank 2
MBANK3=%11000000     ;Bank 2
```

RAM Configuration Register	
Bits	Description
7-6	Bank Select for VIC 00 Bank 0 01 Bank 1 10 Bank 2 11 Bank 3
5-4	Not Used
3-2	Common Ram Location 00 Disabled 01 Bottom 10 Top 11 Both
0-1	Size of Common Ram 00 1k 01 4k 10 8k 11 16K

```
CMRAM_1K =%00
CMRAM_4K =%01
CMRAM_8K =%10
CMRAM_16K=%11

;-- Set Shared RAM size to 16K
lda   MMURCR
and   #%11111100
ora   CMRAM_16K
sta   MMURCR

.macro SetBankConfiguration(id) {
 .if(id==0) {
  lda #%00111111  // no ROMs, RAM0
 }
 .if(id==1) {
  lda #%01111111  // no ROMs, RAM1
 }
 .if(id==12) {
  lda #%00000110  // int.function ROM, Kernal and IO, RAM0
 }
 .if(id==14) {
  lda #%00000001  // all ROMs, char ROM, RAM0
 }
 .if(id==15) {
  lda #%00000000  // all ROMs, RAM0. default setting.
 }
 .if(id==99) {
  lda #%00001110  // IO, kernal, RAM0. 48K RAM.
 }
 sta MMUCR
}
.endm

.macro SetVICBank (bank) {
 lda $DD00
 and #%11111100
 ora #3 - bank
 sta $DD00
.endm
```

memory maps

GEOS Memory Map:

All address Values in Hex

C64 Memory Regions

00	100	200	300	400	6000	8000	8A00	8C00	9000	A000	BF40	D000	E000
Zero Page	Stack Page	deskTopVars	Vectors	AppRAM	Back Screen	Disk Buffers	Sprites	COLOR MATRIX	DISK BASE	Fore Screen	Kernal Low	I/O or Kernal	Kernal High

Zero Page

00	**CPU_DDR**	6510 data direction register
01	**CPU_DATA**	Built-in 6510 I/O port, bit oriented
02-21	**r0-r15**	GEOS Kernal zero Page pseudoregisters
22-23	**curPattern**	Pointer to fill pattern data
24-25	**string**	Pointer to input buffer
26	**fontTable**	;- Label for Start of Current Font Settings
26	**baselineOffset**	number of pixels from top of font to baseline.
27-28	**curSetWidth**	pixel width of font bitstream in bytes.
29	**curHeight**	card height in pixels (Point size[1]) of font
2A-2B	**curIndexTable**	pointer to font index table.
2C-2D	**cardDataPntr**	Pointer to font image data.
	;--- fontTable End	
2E	**currentMode**	Defines the current print style
2F	**dispBufferOn**	Controls the screen to draw too. Fore/Back or Both.
30	**mouseOn**	Mouse control flag.
31-32	**msePicPtr**	Pointer Mouse sprite, default= 84C1
	;--- Text Clipping	
33	**windowTop**	Top margin, usually 0 (Top of screen)
34	**windowBottom**	Bottom margin, usually 199
35-36	**leftMargin**	Left margin
37-38	**rightMargin**	Right margin
39	**pressFlag**	Input control flags
3A-3B	**mouseXPos**	Mouse's X position
3C	**mouseYPos**	Mouse's Y position
3D-3E	**returnAddress**	Address for inline return
3F	**graphMode**	40 / 80 column mode flag on 128
40-41	**returnAddress**	Pointer to click box data table
42-43	**callRouVector**	Jump vector used by **CallRoutine**
44-45	**dlgBoxVector**	**DoDlgBox** pointer to window descriptor block.
45-6F		Used by Kernal
70-7F	A2-A9	Generically Named. Application zPage area
80-FA	**zKerIO**	Kernal I/O*
BA	**curDevice**	current serial device number
FB-FE	A0-A1	Generically Named. Application zPage area
FF		Used by Kernal

*Note: 80-FA is only used by the kernal during IO. See **SwZp** for how to make safe use of this area in your applications.

Stack Page

```
==================================================================================
0100-01FF                 6510 Hardware Stack Area.
                          (GEODEBUGGER uses bottom of stack as a data area)

0200-02FF   deskTopVars   ;Application may freely use this block
0300-03FF
```

128 BackRAM:

GEOS Primary Bank is Bank 1.
BackRAM is bank 0. This allows common RAM to be turned on and have parts of bank 0 then Appear into the memory space of bank 1 as shared ram is always Bank 0 ram and is always visible to the CPU when active.

Bank 0:
0000-03FF: ?

0400-1FFF: Soft Sprites
2000-9FFF: Swap area for Desk Accessories
If your application does not use Desk Accessories this may be used as Application data area.
A000-FFFF: ??

Bank 0 Back Ram

$0000	$400	$FF00	$FF05
	BANK 0	MMU	ROM

Bank 1 GEOS Address Space

$0000	$400	$FF00	$FF05
	BANK 1 GEOS APPLICATION SPACE	MMU	ROM

Bank 2

$0000	$400	$FF00	$FF05
	BANK 2 (bank 0 if 128 is not expanded)	MMU	ROM

Bank 3

$0000	$400	$FF00	$FF05
	BANK 3 (bank 1 if 128 is not expanded)	MMU	ROM

Bank 14

$0000	$400	$4000	$D000	$E000	$FF00	$FF05
	RAM 0	Basic ROM	Char Rom	Kernal ROM	MMU	ROM

Bank 15

$0000	$400	$4000	$D000	$E000	$FF00	$FF05
Common RAM	RAM 0	Basic ROM	I/O	Kernal ROM	MMU	ROM

REU-BANK0

Macros

Terms

addend	a number which is added to another.
addr	Target for a relative branch.
augend	the number to which an addend is added.
bitNumber	Index for bit position. example %10000000 / bitNumber 7 is set
difference	Result of subtraction
dest	An address to store a macro result.
immed	A Constant number
minuend	A number from which another is to be subtracted
result	The Sum of addition New value after BIT operation
source	An address to load from. Address or Immediate value in Byte macros.
subtrahend	A number to be subtracted from another.
value	a Constant number.

Categories

Identifier	Category
bit	bit operations
br	branching
cmp	Comparisons
flow	Alters flow of logic
math	Math
hw	Hardware
util	Utility

Sources

Identifier	Source
gP1	geoProgrammer1.1
gP'2	geoProgrammer' 2.0
HHG	Created by PBM to perform actions for HHG2G Macros that were not defined in geoProgrammer1.1. Example: macro **bgt** is used in HHG2. But is not in geoProgrammer1.1. Macro logic was obvious so it was created here for use in the examples.
	Other sources will be added as used

By Category

bit operations

Macro	Parameters	Description	Source
rmb	 bitNumber dest	resets bit in destination byte bit number in byte to reset address of byte which contains bit to reset	gP1
rmbf	 bitNumber dest	resets bit in destination byte bit number in byte to reset address of byte which contains bit to reset Destroys: accumulator	gP1
smb	 bitNumber result	Set bit in result byte bit number in byte to set address of byte which contains bit to set	gP1
smbf	 bitNumber result	Set bit in result byte Destroys: the accumulator bit number in byte to set address of byte which contains bit to set	gP1

branching

Macro	Parameters	Description	Source
bbr	 bitNumber source addr	Tests bit in source byte, branches if reset bit number in byte to test (7 for MSD) address of byte which contains bit to test where to branch to if bit is reset	gP1
bbrf	 bitNumber source addr	Test bit in source byte, branches if reset bit number in byte to test (7 for MSD) address of byte which contains bit to test where to branch to if bit is reset Destroyed: accumulator if bitNumber is < 6	gP1
bbs	 bitNumber source addr	Tests bit in source byte, branches if is set bit number in byte to test (7 for MSD) address of byte which contains bit to test where to branch to if bit is set	gp1
bbsf	 bitNumber source addr	Test bit in source byte, branches if is set bit number in byte to test (7 for MSD) address of byte which contains bit to test where to branch to if bit is set Destroyed: accumulator if bitNumber is < 6	
bge	addr	relative branch if A>=B Carry is set	gP1
bgt	addr	relative branch if A>B Carry Flag is set and Z Flag is not set.	HHG
blt	addr	relative branch if A<B Carry is Clear	HHG
bra	addr	Unconditional branch to relative addr.	gP1

comparisons

Macro	Parameters	Description	Source
CmpB	 source dest	Compare contents of source and dest bytes address of first byte or #Immidiate value address of second byte	gP1
CmpBI	 source	Compare contents of source to a constant address of first byte	gP1

	immed	value to compare to	
CmpW	 source dest	Compare contents of source and dest bytes address of first byte address of second byte	gP1
CmpWI	 source immed	Compare contents of source to a constant address of first word Appendix constant value to compare to	gP1

Math

Macro	Parameters	Description	Source
add	addend	a = a + addend	gP1
AddB	 addend augend	augend = augend + addend address of byte to add or #constant value address of byte to add to	gP1
AddBW	 addend augend	augend = augend + addend address of byte to add address of word to add to	gP'
AddVB	 value augend	augend = augend + value constant to add to augend address of byte to add to	gP1
AddVW	 value augend	augend = augend + value constant to add to augend address of word to add to	gP1
AddW	 addend augend	augend = augend + addend address of word to add address of word to add to	gP1
sub	 subtrahend	a = a – subtrahend address of byte to subtract or #Constant value	gP1
SubB	 subtrahend minuend	minuend = minuend – subtrahend (m=m-s) address of byte to subtract or #Constant Value address of byte to subtract from	gP1
SubBW	 subtrahend difference	Minuend = Minuend – subtrahend (M=M-s) address of byte to subtract address of word to subtract from	gP'
SubW	 subtrahend difference	Minuend = Minuend – subtrahend (M=M-S) address of word to subtract address of word to subtract from	gP1

Hardware

Macro	Parameters	Description	Source

Utility

Macro	Parameters	Description	Source
IncW	 dest	increment word by 1 address of word to increment	gp2
LdW	 dest value	Load a word with a compacted value address of byte to load with value. constant to load. *(If High and Low parts of constant are the same. accumulator only loaded once.)*	gP2
LoadW	 dest value	Load a word with a value address of word to load with value constant to Load	gP1
MoveB		Move byte contents of source to destination.	gP1

	source dest	source address destination address	
MoveW	 source dest	Move word contents of source to destination. source address destination address	gP1
PopB	 dest	Pop a byte from the stack where to store byte value	gP1
PopW	 dest	Pop a byte from the stack where to store byte value	gP1
PushB	 source	Push the byte at source onto the stack address of the byte to push or #Constant value	gP1
PushW	 source	Push the word at source onto the stack address of the byte to push	gP1

By Name

Macro	Parameters	Description	Source	Category
add	addend	a = a + addend	gP1	math
AddB	 addend augend	augend = augend + addend address of byte to add or #constant value address of byte to add to	gP1	math
AddBW	 addend augend	augend = augend + addend address of byte to add address of word to add to	gP'	math
AddVB	 value augend	augend = augend + value constant to add to augend address of byte to add to	gP1	math
AddVW	 value augend	augend = augend + value constant to add to augend address of word to add to	gP1	math
AddW	 addend augend	augend = augend + addend address of byte to add address of byte to add to	gP1	math
bbr	 bitNumber source addr	tests bit in source byte, branches if reset bit number in byte to test (7 for MSD) address of byte which contains bit to test where to branch to if bit is reset	gP1	br
bbrf	 bitNumber source addr	tests bit in source byte, branches if reset bit number in byte to test (7 for MSD) address of byte which contains bit to test where to branch to if bit is reset Destroyed: accumulator if bitNumber is < 6	gP1	br
bbs	 bitNumber source addr	tests bit in source byte, branches if is set bit number in byte to test (7 for MSD) address of byte which contains bit to test where to branch to if bit is set	gP1	br
bbsf	 bitNumber source addr	tests bit in source byte, branches if is set bit number in byte to test (7 for MSD) address of byte which contains bit to test where to branch to if bit is set Destroyed: accumulator if bitNumber is < 6	gP1	br
bgt	addr	relative branch if A>B Carry Flag is set and Z Flag is not set.	HHG	br
blt	addr	relative branch if A<B Carry is Clear	HHG	br
bra	addr	Unconditional branch to relative addr.	gP1	br
cldaI	 label value	load accumulator on branch to label Label for branch targeting constant to load into accumulator on branch	gP'2	flow
CmpB	 source dest	compare contents of source and dest bytes address of first byte or #Constant value address of second byte or #Constant value	gP1	cmp
CmpBI	 source immed	compares contents of source to value address of first byte value to compare to	gP1	cmp
CmpW	 source	compare contents of source and dest bytes address of first byte	gP1	cmp

	dest	address of second byte		
CmpWI	 source immed	compares contents of source to a constant address of first word constant value to compare to	gP1	cmp
IncW	 dest	increment word by 1 address of word to increment	gp2	util
LdW	 dest value	Load a word with a compacted value address of byte to load with value. constant to load. *(If High and Low parts of constant are the same. accumulator only loaded once.)*	gP2	util
LoadB	 dest value	Load a byte with a value. address of byte to load with value. constant to load.	gP1	util
LoadW	 dest value	Load a word with a compacted value address of byte to load with value. constant to load.	gP1	util
MoveB	 source dest	Move byte contents of source to destination. source address destination address	gP1	util
MoveW	 source dest	Move word contents of source to destination. source address destination address	gP1	util
PopB	 dest	Pop a byte from the stack where to store byte value	gP1	util
PopW	 dest	Pop a byte from the stack where to store byte value	gP1	util
PushB	 source	Push the byte at source onto the stack address of the byte to push or #Constant value	gP1	util
PushW	 source	Push the word at source onto the stack address of the byte to push	gP1	util
rmb	 bitNumber dest	resets bit in destination byte bit number in byte to reset address of byte which contains bit to reset	gP1	bit
rmbf	 bitNumber dest	resets bit in destination byte bit number in byte to reset address of byte which contains bit to reset Destroys: accumulator	gP1	bit
smb	 bitNumber dest	set bit in destination byte bit number in byte to set (7 for MSD) address of byte which contains bit to set	gP1	bit
smbf	 bitNumber result	Set bit in result byte Destroys: accumulator bit number in byte to set address of byte which contains bit to set	gP1	bit
sub	subtrahend	a = a – subtrahend address of byte to subtract or #Constant value	gP1	math
SubB	 subtrahend minuend	minuend = minuend – subtrahend address of byte to subtract or #Constant Value address of byte to subtract from	gP1	math
SubBW	 subtrahend difference	minuend = minuend – subtrahend address of byte to subtract address of word to subtract from	gP'	math

SubW	 subtrahend minuend	minuend = minuend – subtrahend address of word to subtract address of word to subtract from	gP1	math

add: math

```
;*********************************************************
;
;     Add Byte:    add addend
;
;     Args: addend -    address of byte to add
;                        or constant value
;     Action:           a = a + addend
;
;*********************************************************
.macro      add         addend
      clc
      adc   addend
.endm
```

Note:

Example:

See also: AddB, AddW

AddB: math

```
;*********************************************************
;
;     Add Bytes:    AddB addend, augend
;
;     Args: addend - address of byte to add
;                         or constant value
;           augend - address of byte to add to
;
;     Action:     augend = augend + addend
;
;*********************************************************
.macro      AddB  addend,augend
      clc               ;must add with carry
      lda   addend      ;get source byte
      adc   augend      ;add to destination byte
      sta   augend      ;store result
.endm
```

Note:

Example:

```
.ramsect
      myVar .block 1
      count .block 1
.psect

      AddB  #20,myVar         ;Add Constant to myVar
                              ;(# required for constant)
      AddB  myVar, count      ;Add myVar to count
```

See also: AddB, AddW

AddBW: math

```
;*************************************************************
; geoProgrammer'
;     Add Byte To Word:    AddBW addend, augend
;
;     Args: addend - address of byte to add
;           augend - address of word to add to
;
;     Action:     augend = augend + addend
;*************************************************************
.macro      AddBW addend,augend
      lda   addend
      clc
      adc   augend
      sta   augend
      bcc   NoInc
      inc   augend+1
NoInc:
.endm
```

Note:

Example:

See also: AddB,AddW

AddVB: math

```
;*********************************************************
;
;     Add Value To Byte:    AddVB value, augend
;
;     Args: value - constant to add to augend
;           augend - address of byte to add to
;
;     Action:     augend = augend + value
;
;*********************************************************
.macro      AddVB value, augend
      lda   augend
      clc
      adc   #value
      sta   augend
.endm
```

Note: This macro is redundant with AddB since AddB can do immediate values too. Left in geoProgrammer' 2.0 for backwards compatibility with existing source.
Example:

See also: AddW

AddVW: math

```
;***********************************************************
;
;     Add Value to Word:    AddVW value,augend
;
;     Args: value - constant to add to augend
;           augend - address of word to add to
;
;     Action:     augend = augend + value
;
;***********************************************************
.macro      AddVW value,augend
      clc                ;must add with carry
      lda   #[(value)    ;get low byte of value
      adc   augend       ;add to low byte of word
      sta   augend       ;store updated value

.if   (value >= 0) && (value <= 255)
      bcc   noInc        ;carry was set if adc above overflowed.
      inc   augend+1     ;increment high byte of word
noInc:
.else
      lda   #](value)    ;carry was set if adc above overflowed.
      adc   augend+1     ;add carry + 0 to high byte of address
      sta   augend+1     ;store result
.endif

.endm
```

Note:

Example:

See also: AddB

AddW: math

```
;*********************************************************
;
;     Add Words:    AddW addend,augend
;
;     Args: addend - address of word to add
;           augend - address of word to add to
;
;     Action:     augend = augend + addend
;
;*********************************************************
.macro      AddW  addend,augend
      lda   addend      ;get addend low byte
      clc
      adc   augend      ;add to destination low byte
      sta   augend      ;store result, sec carry with overflow
      lda   addend+1    ;get source high byte
      adc   augend+1    ;add with carry to high byte dest
      sta   augend+1    ;store result
.endm
```

Note:

Example:

See also: AddB

bge: branch

```
;************************************************************
;
;     Branch On Result of compare:  bge addr
;                                   if a >= b:
;     Args: addr - where to branch to
;
;     Action:     relative branch if carry is set
;
;************************************************************
.macro      bge addr
      bcs   addr
.endm
```

Note:

Example: **RoadTrip**

See also: bge, bgt, blt, ble

bgt: **branch**

```
;***********************************************************
;
;     Branch On Result of compare:  bgt addr
;                                   if a > b:
;     Args: addr - where to branch to ;
;     Action:     relative branch if carry flag is set and
;                                        Z flag is not set
;
;***********************************************************
.macro      bgt addr
      beq   label
      bcs   addr
label:
.endm
```

Note:

Example: **NewIsMseInRegion**

See also: bge, bgt, blt, ble

ble: **branch**

```
;*************************************************************
;
;     Branch On Result of compare:  ble addr
;                                   if a <= b
;     Args: addr - where to branch to
;
;     Action:     relative branch if carry flag is not set or
;                                        Z flag is set
;
;*************************************************************
.macro      ble   addr
      bcc   addr
.endm
```

Note:

Example:

See also: bge, bgt, blt, ble

blt: **branch**

```
;*********************************************************
;
;     Branch On Result of compare:  blt addr
;                                   if a < b:
;     Args: addr - where to branch to
;
;     Action:     relative branch if carry flag is clear
;
;*********************************************************
.macro      blt   addr
      bcc   addr
.endm
```

Note:

Example: **NewIsMseInRegion**

See also: bge, bgt, ble, bra

bra:	**branch**

```
;*************************************************************
;
;     Branch Relative Always:    bra addr
;
;     Args: addr - where to branch to
;
;     Action:     unconditional relative branch
;
;*************************************************************
.macro      bra   addr
      clv
      bvc   addr
.endm
```

Note:

Example: **RoadTrip**

See also: bge, bgt, blt, ble

bbr: **branch**

```
;*********************************************************
;
; Branch on Bit Reset:    bbr bitNumber,source,addr
;
; Args:     bitNumber - bit number in byte to test (7 for MSD, 0 for LSD)
;           source - address of byte which contains bit to test
;           addr - where to branch to if bit is reset
;
; Action:   tests bit in source byte, branches if is reset
;
;*********************************************************
.macro      bbr   bitNumber,source,addr
      php
      pha
      lda   source
      and   #(1 << bitNumber)
      bne   nobranch
      pla
      plp
      bra   addr
 nobranch:
      pla
      plp
 .endm
```

Note:

Example:

See also: bbrf

bbrf: **branch**

```
;*******************************************************************
;
; Branch on Bit Reset:    bbrf bitNumber,source,addr
;
; Args:     bitNumber - bit number in byte to test (7 for MSD, 0 for LSD)
;           source - address of byte which contains bit to test
;           addr - where to branch to if bit is reset
;
; Action:   tests bit in source byte, branches if is reset
; Destroys: accumulator if bitNumber is < 6
;
;*******************************************************************

.macro      bbrf  bitNumber,source,addr
.if   (bitNumber = 7)
      bit   source
      bpl   addr
.elif (bitNumber = 6)
      bit   source
      bvc   addr
.else       lda   source
      and   #(1 << bitNumber)
      beq   addr
.endif
.endm
```

Note:

Example:

See also: bbr

bbs: **branch**

```
;*********************************************************
;
;     Branch on Bit Set:    bbs bitNumber,source,addr
;
;     Args: bitNumber - bit number in byte to test (7 for MSD, 0 for LSD)
;           source - address of byte which contains bit to test
;           addr - where to branch to if bit is set
;
;     Action:     tests bit in source byte, branches if is set
;
;*********************************************************
.macro      bbs   bitNumber,source,addr
      php
      pha
      lda   source
      and   #(1 << bitNumber)
      beq   nobranch
      pla
      plp
      bra   addr
nobranch:
      pla
      plp
.endm
```

Note:

Example:

See also: bbr

bbsf: **branch**

```
;***********************************************************
;     Branch on Bit Set fast:    bbsf bitNumber,source,addr
;
;     Args: bitNumber - bit number in byte to test (7 for MSD, 0 for LSD)
;           source - address of byte which contains bit to test
;           addr - where to branch to if bit is set
;
;     Action:     tests bit in source byte, branches if is set
;     Destroys:   if bitNumber is < 6 the accumulator
;***********************************************************
.macro      bbsf  bitNumber,source,addr
.if   (bitNumber = 7)
      bit   source
      bmi   addr
.elif (bitNumber = 6)
      bit   source
      bvs   addr
.else
      lda   source
      and   #(1 << bitNumber)
      bne   addr
.endif
.endm
```

Note:

Example:

See also: bbr

macros by name

cldaI: **flow**

.if **COMMENT**

Source: geoProgrammer' by PBM

Purpose: Conditional Load Immediate: **cldaI** label,value

Pass: label - Label for branch targeting

value - constant to load into accumulator if branch target is used.

Action: load accumulator on branch to label.

.endif

```
.macro cldaI label,value
      .byte $2c
label:
      lda #[(value)
.endm
```

Note:

Example:

```
            lda   #4          ; If flow gets here a=4 when PointRecord called.
cldaI 40$,        #3          ; Local labels are ok. a=3 if branch to 40$
cldaI Rec2,       #2          ; if branch or jmp/jsr to Rec2 a = 2
cldaI Rec1,       #1          ; if jmp/jsr to Rec1 a = 1
      jsr   PointRecord
      ...
```

See also:

CmpB: cmp

```
;*********************************************************
;
; Compare Bytes:    CmpB  source,dest
;
; Args:     source - address of first byte
;                  - or #Immidiate value
;           dest - address of second byte
;                  - or #Immidiate value
; Action:   compare contents of source byte to contents of dest. byte
;
;*********************************************************
.macro      CmpB  source,dest
      lda   source      ;get source byte
      cmp   dest        ;compare source to dest
.endm
```

Note:

Example:

```
.ramsect
      myVar .block 1
      count .block 1
.psect

      CmpB  #20,myVar         ;Compare Constant with variable
      CmpB  myVar, count      ;Compare two variables
      CmpB  count,#40         ;variables with Constant
```

See also:

CmpBI: cmp

```
;*******************************************************
;
;     Compare Byte To Value:    CmpBI source,immed
;
;     Args: source - address of first byte
;           immed - value to compare to
;
;     Action:     compares contents of source to value
;
;*******************************************************
.macro      CmpBI source,immed
      lda   source      ;get source byte
      cmp   #immed            ;compare source to immediate value
.endm
```

Note: This macro is redundant with CmpB since AddB can do immediate values too. Left in geoProgrammer' 2.0 for backwards compatibility with existing source.

Example:

See also:

CmpW: cmp

```
;*********************************************************;
;     Compare Words:    CmpW source,dest
;
;     Args: source - address of first word
;           dest - address of second word
;
;     Action:     compare contents of source word to contents of dest. word
;
;*********************************************************
.macro      CmpW  source,dest
      lda   source+1    ;get high source byte
      cmp   dest+1      ;compare source to dest
      bne   done        ;need to do low byte?
      lda   source      ;do low byte
      cmp   dest        ;compare low byte
done:
.endm
```

Note:

Example:

See also:

CmpWI: branch

```
;*******************************************************;
;     Compare Word To a Constant:    CmpWI source,immed
;
;     Args: source - address of first word
;           immed - value to compare to
;
;     Action:     compares contents of source to a Constant
;
;*******************************************************
.macro      CmpWI source,immed
      lda   source+1    ;get high byte
      cmp   #](immed)   ;test high byte of immediate Constant
      bne   done        ;don't need to do low byte
      lda   source      ;get low byte
      cmp   #[(immed)   ;test low byte
done:
.endm
```

Note:

Example:

See also:

IncW: utility

```
;***********************************************************
; geoProgrammer'
; Conditional Load:    IncW   dest
;
; Args:     dest - address of word to increment
;
; Action:   increment word by 1.
;
;***********************************************************
.macro      IncW dest
      inc   dest
      bne   done
      inc   dest+1
      done:
.endm
```

Note:

Example:

See also:

LdW:	**utility**

```
;***********************************************************
; geoProgrammer'
; Load Compacted Word:    LdW dest,value
;
;     Args: dest - address of word to load with value
;           value - word to load
;
;     Action:     Load a word with a compacted value
;     (If High and Low parts of constant are the same.
;           accumulator only loaded once.)
;***********************************************************
.macro      LdW   dest,value
      lda   #](value)   ;get higher byte of value to load
      sta   dest+1      ;store it
.if   #](value) <> #[(value)
      lda   #[(value)   ;get lower byte of value to load
.endif
      sta   dest        ;store it
.endm
```

Note:

Example: **ShowBitmap**

See also: LoadB

LoadB: utility

```
;*************************************************************
;     Load Byte:    LoadB  dest,value
;
;     Args: dest - address of byte to load with value
;           value - byte to load
;
;     Action:     Load a byte with a value
;*************************************************************
.macro      LoadB dest,value
      lda   #value      ;load value
      sta   dest        ;store it
.endm
```

Note:

Example: **ShowBitmap**

See also: LoadW

LoadW: **utility**

```
;*****************************************************************
; Load Word:    LoadW  dest,value
;
; Args:     dest - address of word to load with value
;           value - word to load
;
; Action:   Load a word with a value
;*****************************************************************
.macro      LoadW dest,value
      lda   #](value)   ;get higher byte of value to load
      sta   dest+1      ;store it
      lda   #[(value)   ;get lower byte of value to load
      sta   dest+0      ;store it
.endm
```

Note:

Example: **ShowBitmap**

See also: LoadB

MoveB: utility

```
;*********************************************************
; Move Byte:    MoveB source,dest
;
; Args:     source - source address
;           dest - destination address
;
; Action:   Moves byte contents of source to destination.
;*********************************************************
.macro      MoveB source,dest
      lda   source      ;load data from source
      sta   dest        ;store it in destination
.endm
```

Note:

Example:

See also: MoveW

MoveW: utility

```
;*************************************************************
;     Move Word:    MoveW  source,dest
;
;     Args: source - source address
;           dest - destination address
;
;     Action:     Moves a word from source address to dest address.
;*************************************************************
.macro      MoveW source,dest
      lda   source+1    ;get high byte
      sta   dest+1      ;store it
      lda   source+0    ;get low byte
      sta   dest+0      ;store it
.endm
```

Note:

Example:

See also: MoveB

PopB: utility

```
;*************************************************************
;
;     Pop Byte:     PopB dest
;
;     Args: dest - where to store byte value
;
;     Action:     Pops a byte from the stack
;
;************************************************************
.macro      PopB  dest
      pla         ;get byte
      sta   dest  ;and save it
.endm
```

Note:

Example:

See also:

PopW: **utility**

```
;*************************************************************
;
;     Pop Word:    PopW dest
;
;     Args: dest - where to store word value
;
;     Action:     Pop a word from the stack
;
;*************************************************************
.macro      PopW  dest
      pla         ;get low byte of word
      sta   dest+0      ;and save it
      pla               ;get high byte of word
      sta   dest+1      ;and save it
.endm
```

Note:

Example:

See also:

PushB: utility

```
;*****************************************************************
;     Push Byte:    PushB source
;
;     Args: source - address of the byte to push
;                     or a Constant value
;
;     Action:     Pushes the byte at source onto the stack
;                     or a Constant value
;*****************************************************************
.macro      PushB source
      lda   source      ;get byte
      pha               ;and push it
.endm
```

Note:

Example:

See also: MoveB

PushW: **utility**

```
;*************************************************************
;
;     Push Word:    PushW source
;
;     Args: source - address of the word to push
;
;     Action:     Pushes the word at source onto the stack
;
;************************************************************
.macro      PushW source
      lda   source+1    ;get high byte of word
      pha               ;and push it
      lda   source      ;get low byte of word
      pha               ;and push it
.endm
```

Note:

Example:

See also:

rmb: **bit**

```
;**********************************************************
;
;     Reset Bit:    rmb bitNumber,dest
;
;     Args: bitNumber - bit number in byte to reset
;            (7 for MSD, 0 for LSD)
;           dest - address of byte which contains bit to reset
;
;     Action:     resets bit in destination byte
;     Destroys:   the accumulator
;
;*********************************************************
.macro      rmb   bitNumber,dest
      pha
      lda   dest
      and   #[~(1 << bitNumber)
      sta   dest
      pla
.endm
```

Note:

Example:

rmbf: **bit**

```
;************************************************************
;
;     Reset Bit:    rmbf bitNumber,dest
;
;     Args: bitNumber - bit number in byte to reset
;             (7 for MSD, 0 for LSD)
;           dest - address of byte which contains bit to reset
;
;     Action:     resets bit in destination byte
;     Destroys:   the accumulator
;
;***********************************************************
.macro      rmbf  bitNumber,dest
      lda   dest
      and   #[~(1 << bitNumber)
      sta   dest
.endm
```

Note:

Example:

See also:

smb:	**branch**

```
;**********************************************************
;
; Set Bit:    smb bitNumber,result
;
; Args:     bitNumber - bit number in byte to set (7 for MSD, 0 for LSD)
;           result    - address of byte which contains bit to set
;
; Action:   sets bit in result byte
;
;**********************************************************
.macro      smb   bitNumber,result
      pha
      lda   result
      ora   #(1 << bitNumber)
      sta   result
      pla
.endm
```

Note:

Example:

See also: smbf

smbf: **branch**

```
;*******************************************************************
;
; Set Bit:    smbf bitNumber,result
;
; Args:     bitNumber - bit number in byte to set (7 for MSD, 0 for LSD)
;           result    - address of byte which contains bit to set
;
; Action:   sets bit in result byte
; Destroys:   the accumulator
;
;*******************************************************************
.macro      smbf  bitNumber,result
      lda   result
      ora   #(1 << bitNumber)
      sta   result
.endm
```

Note:

Example:

See also: smb

sub: **utility**

```
;*********************************************************
;
; Subtract Byte:    sub subtrahend
;
; Args:     subtrahend - address of byte to subtract, or immediate value
;
; Action:   a = a - subtrahend
;
;**********************************************************
.macro      sub   subtrahend
      sec
      sbc   subtrahend
.endm
```

Note:

Example:

See also:

SubB: **math**

```
;***********************************************************
;
; Sub Bytes:    SubB subtrahend, minuend
;
; Args:       subtrahend - address of byte to subtract
;                   or #Constant Value
;             minuend - address of byte to subtract from
;
; Action:     minuend = minuend - subtrahend
;
;***********************************************************
.macro      SubB  subtrahend, minuend
      sec               ;must add with carry
      lda   minuend     ;get minuend byte
      sbc   subtrahend  ;subtract subtrahend byte
      sta   minuend     ;store result in minuend
.endm
```

Note:

Example:

See also:

SubBW: math

```
;*******************************************************
; geoProgrammer'
; Sub Bytes:    SubBW subtrahend, minuend
;
; Args:      subtrahend - address of byte to subtract
;                        or #Constant Value
;            minuend - address of byte to subtract from
;
; Action:    minuend = minuend - subtrahend
;
;*******************************************************
.macro SubBW subtrahend, minuend
      sec
      lda minuend
      sbc subtrahend
      sta minuend
      lda minuend +1
      sbc #0
      sta minuend +1
.endm
```

Note:

Example:

See also:

SubW: **math**

```
;*************************************************************
;
; Sub Words:    SubW  subtrahend, minuend
;
; Args:         subtrahend - address of word to subtract
;               minuend - address of word to subtract from
;
; Action:       minuend = minuend - subtrahend
;
;*************************************************************
.macro        SubW  source, minuend
        lda   minuend            ;get source low byte
        sec
        sbc   subtrahend         ;subtract from minuend low byte
        sta   minuend            ;store result, clc carry with overflow
        lda   minuend +1         ;get subtrahend high byte
        sbc   subtrahend+1       ;sub with carry from minuend high byte
        sta   minuend +1         ;store result in minuend
.endm
```

Note:

Example:

See also:

disk

1st:

If you use TRUE/FALSE and Block Comments then these constants should always appear first in your constants files.

```
TRUE=-1
FALSE=0
COMMENT=FALSE
```

disk

C128:

```
ADD1_W=$2000
DOUBLE_W=$8000
DOUBLE_B=$80
DBL_W_P1=DOUBLE_W | ADD1_W

GR_40=0           ;graphMode
GR_80=%10000000
```

Dialog Box:

```
DEF_DB_POS        = $80      ;command for default dialogue box position
SET_DB_POS        = 0        ;command for user-set DB position
```

Descriptor table commands

```
OK                = 1        ;Put up system icon for "OK", command is
                             ;followed by 2 byte position indicator, x pos.
                             ;in bytes, y pos. in pixels. NOTE: positions
                             ;are offsets from the top left corner of the
                             ;dialogue box.
CANCEL            = 2        ;Like OK, system DB icon, position follows
YES               = 3        ;Like OK, system DB icon, position follows
NO                = 4        ;Like OK, system DB icon, position follows
OPEN              = 5        ;Like OK, system DB icon, position follows
DISK              = 6        ;Like OK, system DB icon, position follows
FUTURE1           = 7        ;reserved for future system icons
FUTURE2           = 8        ;reserved for future system icons
FUTURE3           = 9        ;reserved for future system icons
FUTURE4           = 10       ;reserved for future system icons
DBTXTSTR          = 11       ;Command to display a text string.
DBVARSTR          = 12       ;Used to put out variant strings.
DBGETSTRING       = 13       ;Get an ASCII string from the user.
DBSYSOPV          = 14       ;Any press not over an icon return to applic.
DBGRPHSTR         = 15       ;Execute graphics string.
DBGETFILES        = 16       ;Get filename from user.
DBOPVEC           = 17       ;User defined other press vector.
DBUSRICON         = 18       ;User defined icon.
DB_USR_ROUT       = 19       ;User defined routine.
```

Offsets into descriptor table.

```
OFF_DB_FORM       = 0        ;box form description, i.e. shadow or not
OFF_DB_TOP        = 1        ;position for top of dialogue box
OFF_DB_BOT        = 2        ;position for bottom of dialogue box
OFF_DB_LEFT       = 3        ;position for left of dialogue box
OFF_DB_RIGHT      = 5        ;position for right of dialogue box
OFF_DB_1STCMD     = 7        ;1st command in dialogue box  *
                             ;descriptor table
```

Icon dimensions.

```
SYSDBI_WIDTH      = 6        ;width in bytes   *
SYSDBI_HEIGHT     = 16       ;height in pixels *
```

```
;These equates define a standard, default, dialogue box position and
;size as well as some standard positions within the box for outputting
;text and icons.
```

Default Coordinates

```
DEF_DB_TOP          = 32        ;top y coordinate of default box
DEF_DB_BOT          = 127       ;bottom y coordinate of default box
DEF_DB_LEFT         = 64        ;left edge of default box
DEF_DB_RIGHT        = 255       ;right edge of default box
```

Standard Text Locations

```
TXT_LN_X            = 16        ;standard text x start
TXT_LN_1_Y          = 16        ;standard text line y offsets
TXT_LN_2_Y          = 32
TXT_LN_3_Y          = 48
TXT_LN_4_Y          = 64
TXT_LN_5_Y          = 80
```

Standard Icon Locations

```
DBI_X_0             = 1         ;left side standard icon x position *
DBI_X_1             = 9         ;center standard icon x position    *
DBI_X_2             = 17        ;right side standard icon x position      *
DBI_Y_0             = 8         ;left side standard icon y position *
DBI_Y_1             = 40        ;center standard icon y position    *
DBI_Y_2             = 72        ;right side standard icon y position      *
```

Disk:

```
DK_NM_ID_LEN        = 18        ; # of characters in disk name

;Equates for variable "driveType". High two bits of driveType have special
;meaning (only 1 may be set):
;        Bit 7: if 1, then RAM DISK
;        Bit 6: if 1, then Shadowed disk
DRV_NULL            = 0         ;No drive present at this device address
DRV_1541            = 1         ;Drive type Commodore 1541
DRV_1571            = 2         ;Drive type Commodore 1571
DRV_1581            = 3         ;Drive type Commodore 1581
DRV_NETWORK         = 15        ;Drive type for GEOS geoNet "drive"
```

Directory:

DirHeader:

```
;Offsets into a directory header structure

OFF_TO_BAM          = 4         ;first BAM entry
OFF_DISK_NAME       = 144       ;disk name string
OFF_OP_TR_SC        = 171       ;track and sector for off page directory
                                ;entries. 8 files may be moved off page.
OFF_GS_ID           = 173       ;where GEOS ID string is located    *
OFF_GS_DTYPE        = 189       ;GEOS disk type. Currently, is 0 for      *
                                ;normal disk, 'B' for BOOT disk. Zeroed
                                ;on destination disk during disk copy.
```

DirBlock:

```
FRST_FILE_ENTRY     = 2         ;first dir entry is at byte #2
```

DirEntry:

```
ST_WR_PR          = $40       ;write protect bit: bit 6 of byte 0 in the
                              ;directory entry
```

DirEntryOffsets:

```
OFF_CFILE_TYPE    = 0         ;standard commodore file type indicator
OFF_INDEX_PTR     = 1         ;Index table pointer (VLIR file)
OFF_DE_TR_SC      = 1         ;track for file's 1st data block
OFF_FNAME         = 3         ;file name  *
OFF_GHDR_PTR      = 19        ;track/sector info on where header block is
OFF_GSTRUC_TYPE   = 21        ;GEOS file structure type     *
OFF_GFILE_TYPE    = 22        ;geos file type indicator
OFF_YEAR          = 23        ;year (1st byte of date stamp)      *
OFF_SIZE          = 28        ;size of the file in blocks   *
OFF_NXT_FILE      = 32        ;next file entry in directory structure
```

low-level GEOS disk handling routines

```
N_TRACKS          = 35        ;# of tracks available on the 1541 disk
DIR_TRACK         = 18        ;track # reserved on disk for directory
DIR_1581_TRACK    = 40        ;1581 track # reserved on disk for directory
```

Disk access commands

```
MAX_CMND_STR      = 32        ;maximum length a command string would have
DIR_ACC_CHAN      = 13        ;default direct access channel
REL_FILE_NUM      = 9         ;logical file number & channel used for
                              ;relative files.
CMND_FILE_NUM     = 15        ;logical file number & channel used for
                              ;command files
 ;- Indexes to a command buffer. for setting the track and sector number for a
 ;- direct access command.
TRACK             = 9         ;offset to low byte decimal ASCII track number
SECTOR            = 12        ;offset to low byte decimal ASCII sector number
```

DiskError:

```
NO_ERROR            = 0         ;"No Error"
NO_BLOCKS           = 1         ;"not enough blocks"    *
INV_TRACK           = 2         ;"invalid track"  *
INSUFF_SPACE        = 3         ;"not enough blocks on disk"  *
FULL_DIRECTORY      = 4         ;"directory full"
FILE_NOT_FOUND      = 5         ;"file not found"
BAD_BAM             = 6         ;"bad Block Availability Map"
UNOPENED_VLIR       = 7         ;"unopened VLIR file"   *
INV_RECORD          = 8         ;"invalid record" *
OUT_OF_RECORDS      = 9         ;"cannot insert/append more records"
STRUCT_MISMAT       = 10        ;"file structure mismatch"    *
BFR_OVERFLOW        = 11        ;"buffer overflow during load"      *
CANCEL_ERR          = 12        ;"deliberate cancel error"
DEV_NOT_FOUND       = 13        ;"device not found"     *
INCOMPATIBLE        = 14        ;This error is returned when an attempt is made
                                ;to load a program that can't be run on the
                                ;current graphics modes under the C-128 GEOS.
HDR_NOT_THERE       = $20       ;"cannot find file header block"    *
NO_SYNC             = $21       ;"can't find sync mark on disk"
DBLK_NOT_THERE      = $22       ;"data block not present"     *
DAT_CHKSUM_ERR      = $23       ;"data block checksum error"  *
WR_VER_ERR          = $25       ;"write verify error"
WR_PR_ON            = $26       ;"disk is write protected"
HDR_CHKSUM_ERR      = $27       ;"checksum error in header block"
DSK_ID_MISMAT       = $29       ;"disk ID mismatch"     *
BYTE_DEC_ERR        = $2E       ;"can't decode flux transitions off of disk"
DOS_MISMATCH        = $73       ;"wrong DOS indicator on the disk"
```

FileType:

```
;This is the value in the "GEOS file type" byte of a directory
;entry that is pre-GEOS:

NOT_GEOS          = 0         ;Old C-64 file, without GEOS header
                              ;        (PRG, SEQ, USR, REL)
;The following are GEOS file types reserved for compatibility
;with old C64 files, that have simply had a GEOS header placed
;on them.  Users should be able to double click on files of
;type BASIC and ASSEMBLY, whereupon they will be fast-loaded
;and executed from under BASIC.
BASIC             = 1         ;C-64 BASIC program, with a GEOS header
                              ;attached. (Commodore file type PRG.)
                              ;To be used on programs that
                              ;were executed before GEOS with:
                              ;        LOAD "FILE",8
                              ;        RUN
ASSEMBLY          = 2         ;C-64 ASSEMBLY program, with a GEOS header
                              ;attached. (Commodore file type PRG.)
                              ;To be used on programs that were executed
                              ;before GEOS with:
                              ;        LOAD "FILE",8,1
                              ;        SYS(Start Address)
DATA              = 3         ;Non-executable DATA file (PRG, SEQ, or USR)
                              ;with a GEOS header attached for icon & notes
                              ;ability.
;The following are file types for GEOS applications & system use:
;ALL files having one of these GEOS file types should be of
;Commodore file type USR.

SYSTEM            = 4         ;GEOS system file
DESK_ACC          = 5         ;GEOS desk accessory file
APPLICATION       = 6         ;GEOS application file
APPL_DATA         = 7         ;data file for a GEOS application
FONT              = 8         ;GEOS font file
PRINTER           = 9         ;GEOS printer driver
INPUT_DEVICE      = 10        ;INPUT device (mouse, etc.)
DISK_DEVICE       = 11        ;DISK device driver
SYSTEM_BOOT       = 12        ;GEOS system boot file (for GEOS, GEOS BOOT,
                              ;        GEOS KERNAL)
TEMPORARY         = 13        ;Temporary file type, for swap files.
                              ;The deskTop will automatically delete all
                              ;files of this type upon opening a disk.
AUTO_EXEC         = 14        ;Application to automatically be loaded & run
                              ;just after booting, but before deskTop runs.
INPUT_128         = 15        ;128 Input driver
NUM_FILE_TYPES    = 15        ;# of file types, including NON_GEOS (=0)
;GEOS file structure types. Each "structure type" specifies the organization
;of data blocks on the disk, and has nothing to do with the data in the blocks.
SEQUENTIAL        = 0         ;standard T,S structure (like commodore SEQ
                              ;        and PRG files)
VLIR              - 1         ;Variable-length-indexed-record file (used for
                              ;Fonts, Documents & some programs)
                              ;This is a GEOS only format.
```

Standard Commodore file types (supported by the old 1541 DOS)

```
DEL              = 0       ;deleted file
SEQ              = 1       ;sequential file
PRG              = 2       ;program file
USR              = 3       ;user file
REL              = 4       ;relative file
CBM              = 5       ;CBM BAM protection file, currently only on
                           ;1581 disk drivers.  Used to protect specific
                           ;blocks/tracks from collection at validation
                           ;time.

TOTAL_BLOCKS     = 664     ;number of blocks on 1541 disk, not including
                           ;        directory track.
```

disk

Flag Equates

Values for pressFlag variable

```
KEYPRESS_BIT      = 7         ;other keypress
INPUT_BIT         = 6         ;input device change
MOUSE_BIT         = 5         ;mouse press

SET_KEYPRESS      = %10000000 ;other keypress
SET_INPUTCHG      = %01000000 ;input device change    *
SET_MOUSE         = %00100000 ;mouse press
```

Values for faultFlag variable

```
OFFTOP_BIT        = 7         ;mouse fault up
OFFBOTTOM_BIT     = 6         ;mouse fault down
OFFLEFT_BIT       = 5         ;mouse fault left
OFFRIGHT_BIT      = 4         ;mouse fault right
OFFMENU_BIT       = 3         ;menu fault

SET_OFFTOP        = %10000000 ;mouse fault up
SET_OFFBOTTOM     = %01000000 ;mouse fault down
SET_OFFLEFT       = %00100000 ;mouse fault left
SET_OFFRIGHT      = %00010000 ;mouse fault right
SET_OFFMENU       = %00001000 ;menu fault

ANY_FAULT         = %11111000
```

disk

Get File:

```
;The following equates define file loading options for several of the
;GEOS file handling routines like GetFile. These bit definitions are used to
;set the RAM variable loadOpt.

ST_LD_AT_ADDR     = $01       ;"Load At Address": Load file at caller
                              ;specified address instead of address file was
                              ;saved from.

ST_LD_DATA        = $80       ;"Load Datafile": Used when application
                              ;datafile is opened from deskTop. Used to
                              ;indicate to application that r2 and r3
                              ;contain information about where to
                              ;find the selected datafile.

ST_PR_DATA        = $40       ;"Print Datafile": Used when application
                              ;datafile is selected for printing from deskTop.
                              ;Used to indicate to application that r2 and r3
                              ;contain information about where to find the
                              ;selected datafile.
```

disk

Graphics/Screen Equates

Constants for screen size

```
SC_BYTE_WIDTH     = 40        ;width of screen in bytes     *
SC_PIX_WIDTH      = 320       ;width of screen in pixels    *
SC_PIX_HEIGHT     = 200       ;height of screen in scanlines      *
SC_SIZE           = 8000      ;size of screen memory in bytes     *
```

Bits used to set dispBufferOn flag

(controls which screens get written to)

```
ST_WR_FORE        = $80       ;write to foreground
ST_WR_BACK        = $40       ;write to background
ST_WRGS_FORE      = $20       ;graphics strings only write to foreground.
```

Values for graphics strings

```
MOVEPENTO         = 1         ;move pen to x,y
LINETO            = 2         ;draw line to x,y
RECTANGLETO       = 3         ;draw a rectangle to x,y
NEWPATTERN        = 5         ;set a new pattern
ESC_PutString     = 6         ;start PutString interpretation
FRAME_RECTO       = 7         ;draw frame of rectangle
PEN_X_DELTA       = 8         ;move pen by signed word delta in x
PEN_Y_DELTA       = 9         ;move pen by signed word delta in y
PEN_XY_DELTA      = 10        ;move pen signed word delta in x & y
```

Screen colors

```
BLACK             = 0
WHITE             = 1
RED               = 2
CYAN              = 3
PURPLE            = 4
GREEN             = 5
BLUE              = 6
YELLOW            = 7
ORANGE            = 8
BROWN             = 9
LTRED             = 10
DKGREY            = 11
GREY              = 12
MEDGREY           = 12
LTGREEN           = 13
LTBLUE            = 14
LTGREY            = 15
```

Values for PutDecimal calls

```
SET_LEFTJUST      = %10000000 ;left justified   *
SET_RIGHTJUST     = %00000000 ;left justified   *
SET_SUPRESS       = %01000000 ;no leading 0's
SET_NOSUPRESS     = %00000000 ;leading 0's
```

Header Block:

Offsets into a GEOS file header block

```
O_GHIC_WIDTH      = 2        ;byte: width in bytes of file icon
O_GHIC_HEIGHT     = 3        ;byte: indicates height of file icon
O_GHIC_PIC        = 4        ;64 bytes: picture data for file icon
O_GHCMDR_TYPE     = 68       ;byte: Comm. file type
O_GHGEOS_TYPE     = 69       ;byte: GEOS file type
O_GHSTR_TYPE      = 70       ;byte: GEOS file structure type
O_GHST_ADDR       = 71       ;2 bytes: start address of file in mem
O_GHEND_ADDR      = 73       ;2 bytes: end address of file in memory
O_GHST_VEC        = 75       ;2 bytes: init vector if file is appl.
O_GHFNAME         = 77       ;20 bytes, permanent filename.

O_128_FLAGS       = 96       ;1 byte, flags to indicate if this program
                             ;will run under the C128 OS in 40 column and
                             ;in 80 column. These flags are valid for
                             ;applications, desk accessories, and auto-exec
                             ;files.
                             ;Bit 7: zero if runs in 40 column.
                             ;Bit 6: one if runs in 80 column.
bit 7 bit 6
  0    0          $00        64/128 40-column mode only
  0    1          $40        64/128 40/80 and 80-column modes
  1    0          $80        64 Only. Does not run under GEOS 128
  1    1          $C0        128 80-column mode only.

O_GH_AUTHOR       = 97       ;20 bytes: author's name (only if is applic.)
O_GHAPDAT         = $89      ;Application Data.
O_GHINFO_TXT      = $A0      ;offset to notes that are stored with the file
                             ;and edited in the deskTop "get info" box.

if file is an application data file:

O_GHP_DISK        = 97       ;20 bytes: disk name of parent application's
                             ;disk.

O_GHP_FNAME       = 117      ;20 bytes: permanent filename of parent
                             ;application.
```

Hardware:

```
;The following equates define the numbers written to the CPU_DATA register
;(location $0001 in C-64). These numbers control the hardware memory map
;of the C-64.

IO_IN=$35                       ;60K RAM, 4K I/O space in
RAM_64K=$30                     ;64K RAM
KRNL_BAS_IO_IN=$37              ;both Kernal and basic ROM's mapped into memory
KRNL_IO_IN=$36                  ;Kernal ROM and I/O space mapped in

;graphics control register #l grcntrl1
;Location D011
;ie msb raster /ECM /BMM /DEN /RSEL /y scroll bits.
ST_ECM = $40
ST_BCM = $20
ST_DEN = $10
ST_25ROW = $08

;**************************************************************************
;     Principal Memory Map Equates
;**************************************************************************

APP_RAM           =  $0400    ;start of application space   *
BACK_SCR_BASE     =  $6000    ;base of background screen    *
PRINTBASE         =  $7900    ;load address for print drivers     *
APP_VAR           =  $7F40    ;application variable space   *
OS_VARS           =  $8000    ;OS variable base
SPRITE_PICS       =  $8A00    ;base of sprite pictures
COLOR_MATRIX      =  $8C00    ;video color matrix
DISK_BASE         =  $9000    ;disk driver base address
SCREEN_BASE       =  $A000    ;base of foreground screen
OS_ROM            =  $C000    ;start of OS code space
OS_JUMPTAB        =  $C100    ;start of GEOS jump table
vicbase           =  $D000    ;video interface chip base address.
sidbase           =  $D400    ;sound interface device base address.
ctab              =  $D800
cia1base          =  $DC00    ;1st communications interface adaptor (CIA).
cia2base          =  $DD00    ;second CIA chip
EXP_BASE          =  $DF00    ;Base address of RAM expansion unit
MOUSE_JMP         =  $FE80    ;start of mouse jump table
MOUSE_BASE        =  $FE80    ;start of input driver
END_MOUSE         =  $FFFA    ;end of input driver
```

Icon:

```
;These equates are bit values for iconSelFlag that determine how an icon
;selection is indicated to the user. If ST_FLASH is set, ST_INVERT is
;ineffective.

ST_FLASH          = $80       ;bit to indicate icon should flash
ST_INVERT         = $40       ;bit to indicate icon should be inverted
```

offsets into the icon structure

```
OFF_NM_ICNS       = 0         ;number of icons in structure *
OFF_IC_XMOUSE     = 1         ;mouse x start position *
OFF_IC_YMOUSE     = 3         ;mouse y start position *
```

Offsets into an icon record in icon structure.

```
Constant Declarations from HHG2G.
Adopted for Official Constants in geoProgrammer 2.x+
OFF_I_PIC         = 0         ;picture pointer for icon
OFF_I_X           = 2         ;x position of icon
OFF_I_Y           = 3         ;y position of icon.
OFF_I_WIDTH       = 4         ;width of icon.
OFF_I_HEIGHT      = 5         ;height of icon.
OFF_I_EVENT       = 6         ;pointer to service routine for icon is selected.
IRECSIZE          = 8         ;Size of Icon Record

Constant Declarations from geoProgrammer 1.x
Included for backwards combability
OFF_PIC_ICON      = 0         ;picture pointer for icon
OFF_X_ICON_POS    = 2         ;x position of icon
OFF_Y_ICON_POS    = 3         ;y position of icon
OFF_WDTH_ICON     = 4         ;width of icon
OFF_HEIGHT_ICON   = 5         ;height of icon
OFF_SRV_RT_ICON   = 6         ;pointer to service routine for icon
OFF_NX_ICON       = 8         ;next icon in icon structure  *
```

Keyboard:

```
;Values for keys

KEY_INVALID       = 31
KEY_F1            = 1
KEY_F2            = 2
KEY_F3            = 3
KEY_F4            = 4
KEY_F5            = 5
KEY_F6            = 6
KEY_F7            = 14
KEY_F8            = 15
KEY_UP            = 16
KEY_DOWN          = 17
KEY_HOME          = 18
KEY_CLEAR         = 19
KEY_LARROW        = 20
KEY_UPARROW       = 21
KEY_STOP          = 22
```

```
KEY_RUN          = 23
KEY_BPS          = 24
KEY_LEFT         = BACKSPACE
KEY_RIGHT        = 30
KEY_DELETE       = 29
KEY_INSERT       = 28
```

disk

Menu:

Types

```
HORIZONTAL        = %00000000
VERTICAL          = %10000000
CONSTRAINED       = %01000000
UN_CONSTRAINED    = %00000000
```

Offsets

```
OFF_MY_TOP        = 0         ;offset to y pos of top of menu
OFF_MY_BOT        = 1         ;offset to y pos of bottom of menu
OFF_MX_LEFT       = 2         ;offset to x pos of left side of menu
OFF_MX_RIGHT      = 4         ;offset to x pos of right side of menu
OFF_NUM_M_ITEMS   = 6         ;offset to Alignment |Movement |Number of items
OFF_1ST_M_ITEM    = 7         ;offset to record for 1st menu item in structure
```

Actions

```
SUB_MENU          = $80       ;for setting byte in menu table that indicates
DYN_SUB_MENU      = $40       ;whether the menu item causes action      *
MENU_ACTION       = $00       ;or sub menu
```

Mouse Equates

Bit flags for mouseOn variable

```
SET_MSE_ON        = %10000000 ;      *
SET_MENUON        = %01000000
SET_ICONSON       = %00100000

MOUSEON_BIT       = 7
MENUON_BIT        = 6
ICONSON_BIT       = 5
```

Process:

Possible values for processFlags

```
SET_RUNABLE=%10000000          ;runnable flag
SET_BLOCKED=%01000000          ;process blocked flag
SET_FROZEN=%00100000           ;process frozen flag
SET_NOTIMER=%00010000          ;not a timed process flag

RUNABLE_BIT=7                  ;runable flag
BLOCKED_BIT=6                  ;process blocked flag
FROZEN_BIT=5                   ;process frozen flag
NOTIMER_BIT=4                  ;not a timed process flag
```

disk

Text:

Bit flags in mode

```
SET_UNDERLINE     = %10000000
SET_BOLD          = %01000000
SET_REVERSE       = %00100000
SET_ITALIC        = %00010000
SET_OUTLINE       = %00001000
SET_SUPERSCRIPT   = %00000100
SET_SUBSCRIPT     = %00000010
SET_PLAINTEXT     = 0

UNDERLINE_BIT     = 7
BOLD_BIT          = 6
REVERSE_BIT       = 5
ITALIC_BIT        = 4
OUTLINE_BIT       = 3
SUPERSCRIPT_BIT   = 2
SUBSCRIPT_BIT     = 1
```

PutChar constants

```
EOF               = 0         ;end of text object
NULL              = 0         ;end of string
BACKSPACE         = 8         ;move left a card
TAB               = 9
FORWARDSPACE      = 9         ;move right one card
LF                = 10        ;move down a card row
HOME              = 11        ;move to left top corner of screen
UPLINE            = 12        ;move up a card line
PAGE_BREAK        = 12        ;page break
CR                = 13        ;move to beginning of next card row
ULINEON           = 14        ;turn on underlining    *
ULINEOFF          = 15        ;turn off underlining   *
ESC_GRAPHICS      = 16        ;escape code for graphics string
ESC_RULER         = 17        ;ruler escape
REV_ON            = 18        ;turn on reverse video
REV_OFF           = 19        ;turn off reverse video
GOTOX             = 20        ;use next byte as 1+x cursor
GOTOY             = 21        ;use next byte as 1+y cursor
GOTOXY            = 22        ;use next bytes as 1+x and 1+y cursor
NEWCARDSET        = 23        ;use next two bytes as new font id
BOLDON            = 24        ;turn on BOLD characters
ITALICON          = 25        ;turn on ITALIC characters
OUTLINEON         = 26        ;turn on OUTLINE characters
PLAINTEXT         = 27        ;plain text mode
USELAST           = 127       ;erase character
SHORTCUT          = 128       ;shortcut character
```

disk

VIC Chip

```
GRBANK0         = %11       ;bits indicate VIC ram is $0000 - $3fff, 1st 16K
GRBANK1         = %10       ;bits indicate VIC ram is $4000 - $7fff, 2nd 16K
GRBANK2         = %01       ;bits indicate VIC ram is $8000 - $bfff, 3rd 16K
GRBANK3         = %00       ;bits indicate VIC ram is $c000 - $ffff, 4th 16K

MOUSE_SPRNUM    = 0         ;sprite number used for mouse *
                            ;(used to set VIC)

VIC_YPOS_OFF    = 50        ;Position offset from 0 to position a     *
                            ;hardware sprite at the top of the screen.
                            ;Used to map from GEOS coordinates to hardware
                            ;position coordinates.

VIC_XPOS_OFF    = 24        ;As above, offset from hardware 0   *
                            ;position to left of screen, used to map GEOS
                            ;coordinates to VIC.

ALARMMASK       = %00000100 ;mask for the alarm bit in the cia chip
                            ;interrupt control register.
```

VDC:

```
;VDC_cr=$D600
;VDC_dr=$D601

VDC_HT=$00     ; Horizontal Total
VDC_HD=$01     ; Horizontal Displayed
VDC_HP=$02     ; Horizontal Sync
VDC_VHW=$03    ; Vertical Sync Width | Horizontal Sync Width
VDC_VT=$04     ; Vertical Total
VDC_VA=$05     ; Vertical Total Adjust
VDC_VD=$06     ; Vertical Total Adjust
VDC_VP=$07     ; Vertical Displayed
VDC_IM=$08     ; Interlaced Mode
VDC_CTV=$09    ; Rasterlines Per character row
VDC_CMS=$0A    ; Cursor Mode / Cursor Start
VDC_CE=$0B     ; Cursor end
VDC_DSH=$0C    ; Start of Display Memory in VDC RAM
VDC_DSL=$0D
VDC_CPH=$0E    ; Text Mode Cursor Address
VDC_CPL=$0F
VDC_LPV=$10    ; Light Pen V/H position
VDC_LPH=$11
VDC_UAH=$12    ; VDC pointer
VDC_UAL=$13
VDC_AAH=$14    ; Start of Attribute Memory in VDC RAM
VDC_AAL=$15
VDC_CGW=$16    ; Character Width
VDC_CDV=$17    ; Character Height
VDC_VSS=$18    ; Block Fill/Copy
               ; Reverse/Blink control
               ; Vertical smooth scroll
VDC_HSS=$19    ; Bitmap/Attributes/Gap fill/Pixel Clock
               ; Horizontal Smooth scroll
VDC_FBG=$1A    ; Foreground Color / Background Color
VDC_AI=$1B     ; Address increment
VDC_CB=$1C     ; Character base address / RAM-Type
VDC_UL=$1D     ; Underscan scan line
VDC_WC=$1E     ; Block copy/fill word count
VDC_DA=$1F     ; Data Register: Data byte pointed to by current VDC pointer
VDC_BAH=$20    ; Block Copy Source Address.
VDC_BAL=$21
VDC_DEB=$22    ; Display enable begin
VDC_DEE=$23    ; Display Enable end
VDC_DRR=$24    ; DRAM refresh rate
VDC_HVS=$25    ; hsync/vsync
```

Obsolete

Desk Accessory save foreground bit.

```
FG_SAVE            = %10000000 ;save and restore foreground graphics data.
CLR_SAVE           = %01000000 ;save and restore color information.
```

disk

pseudoregisters:

Pseudoregisters are used when calling into the GEOS kernal. Each call will have a list of registers to setup. Registers have common uses across the GEOS API but none are exclusively for only one thing. r12-r15 are very rarely used and make for very safe temporary zpage use. Never use other data areas for temporary storage unless you have already used all of the available options in **r0-r15** that do not conflict with your current kernal interaction.

```
.zsect   $02
    r0     .block   2     ; Pointer
    r1     .block   2     ; Used in RAM operations
    r2     .block   2     ; Ptr to diskname , Buffer Size during Disk I/O
    r3     .block   2     ; Left Margin,  Ptr dataFileName
    r4     .block   2     ; Ptr to Disk Buffers, margins on boxes
    r5     .block   2     ; Ptr to DirEntry
    r6     .block   2     ; Ptr to T/S List for block allocates
    r7     .block   2     ; Start address of Read/Write buffer
    r8     .block   2     ; Internal Kernal use during some kernal calls
    r9     .block   2     ; Pointer to disk structures. DirEntrys/ Info Sector etc.
    r10    .block   2     ; Class Pointer.
    r11    .block   2     ; x Position for PutChar
    r12    .block   2     ; Not Used by Kernel as a parameter
    r13    .block   2     ; Not Used by Kernel as a parameter
    r14    .block   2     ; Not Used by Kernel as a parameter
    r15    .block   2     ; Not Used by Kernel. Commonly used in GEOS Application
;Equates for access to Low and High parts of pseudoregisters.
    r0L    =        $02   ; holds result after DoDlgBox
    r0H    ==       $03
    r1L    =        $04   ; Track Number in Disk I/O
    r1H    ==       $05   ; Sector Number in Disk I/O, Y Position for PutChar
    r2L    =        $06   ; Top Margin,    Pixel Width, Str Length
    r2H    ==       $07   ; Bottom Margin, Pixel Height
    r3L    =        $08   ; Top Margin    Track for Allocate Block.
    r3H    ==       $09   ; Bottom Margin Sector for Allocate Block
    r3L    =        $0A   ; Sprite Number
    r3H    ==       $0B   ; Dest Bank on Move operations.
    r5L    =        $0C   ;
    r5H    ==       $0D   ;
    r6L    =        $0E
    r6H    ==       $0F
    r7L    =        $10   ; FileType to find with FindFTypes
    r7H    ==       $11   ; Number of files to get from FindFTypes
    r8L    =        $12
    r8H    ==       $13
    r9L    =        $14
    r9H    ==       $15
    r10L   =        $16   ; Desk Top Page number
    r10H   ==       $17
    r11L   =        $18   ; row Number in DrawPoint
    r11H   ==       $19
    r12L   =        $1A
    r12H   ==       $1B
    r13L   =        $1C
    r13H   ==       $1D
    r14L   =        $1E
    r14H   ==       $1F
    r15L   =        $20   ; This is the first Goto for temp zpage use.
    r15H   ==       $03
```

Disk

Disk Errors:

GEOS I/O Routines return errors in the X register

Standard Constant	Dec	Hex	Description
NO_ERROR	0	$00	No Error Occurred
NO_BLOCKS	1	$01	Not Enough Blocks On Disk
INV_TRACKS	2	$02	Invalid Track or Sector
INSUFF_SPACE	3	$03	Disk Full, Insufficient Space
FULL_DIRECTORY	4	$03	Directory is Full
FILE_NOT_FOUND	5	$05	File Not Found
BAD_BAM	6	$06	Bad Bam: Attempt to deallocate an unallocated block. (Or the reverse)
UNOPENED_VLIR	7	$07	VLIR file not open Illegal VLIR chain number.
INV_RECORD	8	$08	Invalid VLIR Record. Bad Track/Sector
OUT_OF_RECORDS	9	$09	Out of Records: Too many VLIR chains
STRUCT_MISMATCH	10	$0A	Geos Structure Mismatch File is not a VLIR file.
BFR_OVERFLOW	11	$0B	Buffer Overflow: **ReadRecord** max read size exceeded.
CANCEL_ERR	12	$0C	Deliberate Cancel Error
DEV_NOT_FOUND	13	$0D	Device Not Found
INCOMPATIBLE	14	$0E	Incompatible 40/80
HDR_NOT_THERE	32	$20	Disk Block Read error: No Header Block sync character.
NO_SYNC	33	$21	Unformatted or Missing Disk
DBLK_NOT_THERE	34	$22	No Data Block Found
DAT_CHKSUM_ERR	35	$23	Data Block Checksum Error
WR_VER_ERR	37	$25	Write Verify Error
WR_PR_ON	38	$26	Write Protect On
HDR_CHK_SUM_ERR	39	$27	Disk Block Write: Header Checksum Error
DSK_ID_MISMAT	41	$29	Disk ID Mismatch
BYTE_DEC_ERR	46	$2E	Drive Speed Read error
DOS_MISMATCH	115	$73	Wrong DOS Indicator

Address (hex)

Name 64 128 Size Default Saved Description †128 BackRAM

Kernal Variables

Name	64	128	Size	Default	Saved	Description
alarmSetFlag:	851C	851C	1	FALSE	No	TRUE if the alarm is set for geos to monitor, else FALSE
alarmTmtVector:	84AD	84AD	2	0	Yes	address of a service routine for the alarm clock time-out (ringing, graphic etc.) that the application can use if necessary.
alphaFlag	84B4	84B4	1	0	Yes	Flag for alphanumeric string input 0 if not getting text input 11xx xxxx if getting text input. bit Description --- -- b7: Flag indicating alphanumeric input is on b6: Flag indicating prompt is visible b5-0: Counter before prompt flashes
appMain:	849B	849B	2	0	No	Vector that allows applications to include their own main loop code. The code pointed to by **appMain** will run at the end of every GEOS **MainLoop**.
backBufPtr:	-	131B †	16	None	No	Screen pointer where the back buffer came from. Resides in back ram of C128.
bakclr0: [0-3]	D021 : D024	D021 : D024	1	?	No	Background colors 0-3. 1 Byte each, 4 Total Bytes. Hardware Registers
backXBufNum:	-	132B †	8	None	No	For each sprite, there is one byte here for how many bytes wide the corresponding sprite is. Used by C128 soft sprite routines and resides in back ram.
backYBufNum:	-	1333 †	8	None	No	For each sprite, there is one byte here for how many scanlines high the corresponding sprite. Used by soft sprite routines and resides in back ram.
bootName:	C006	C006	9	GEOS BOOT	No	This is the start of the "GEOS BOOT" string.
BRKVector:	84AF	84AF	2	CF85	Yes	Vector to the routine that is called when a BRK instruction is encountered. The default is to the operating system System Error dialog box routine.
bkvec:	0316	0316	2	?	No	BRK instruction vector when ROMs are switched in.
baselineOffset:	26	26	1	$06	Yes	Offset from top line to baseline in character set. i.e. it changes as fonts change. Default $06 - for BSW 9 Font
callRouVector	42	42	2	None	No	
CPU_DATA:	01	01	1	**RAM_64K**	No	Address of **6510 data register** that controls the hardware memory map of the C64.

variables

Address (hex)

Name	64	128	Size	Default	Saved	Description †128 BackRAM
CPU_DDR:	00	00	1	%101111	No	address of 6510 data direction register Note: Writing $00 to this address will disable output to **CPU_DATA** register. This may cause unexpected results.
cardDataPntr:	2C	2C	2	D2DC (BSW 9)	Yes	This is a pointer to the actual card graphic data for the current font in use.
curDirHead:	8200	8200	256	$00	No	buffer containing header information for the disk in currently selected drive.
curDevice:	BA	BA	1	$08	No	current serial device number. See **curDrive** for more information
curDrive:	8489	8489	1	$08	No	device number of the currently active disk drive. For Commodore, allowed values are 8 – 11.
curEnable:	-	1300 †	1	None	No	This is an image of the C64 mobenble register.
curHeight:	29	29	1	$09	Yes	card height in pixels of the current font in use.
curIndexTable:	2A	2A	2	D218	Yes	pointer to the table of sizes, in bytes, of each card in of the current font.
curmobx2:	-	1302 †	1	None	No	Image of the C64 mobx2 register. Used for C128 soft sprites. Resides in back ram
curmoby2:	-	1301 †	1	None	No	Image of C64 moby2 register. Used for C128 soft sprites. Resides in back ram.
curPattern:	22	22	2	D010	Yes	Pointer to the first byte of the graphics data for the current pattern in use. Note: Each pattern is 1 byte wide and 8 bytes high, to give an 8 by 8 bit pattern.
curRecord:	8496	8496	1	0	No	Current record number for an open VLIR file. Note: When a VLIR file is opened, using **OpenRecordFile**. **curRecord** is set to 0 if there is at least 1 record in the file, or -1 if their are no records.
currentMode:	2E	2E	1	$00	Yes	Current text drawing mode. Each bit is a flag for a drawing style. If set, that style is active, if clear it is inactive. The bit usage and constants for manipulating these bits are as follows. Bit Style Constant --- ----- -------- b7: Underline **SET_UNDERLINE:** = %10000000 b6: Bold SET_BOLD = %01000000 b5: Reverse SET_REVERSE = %00100000

Address (hex)

Name	64	128	Size	Default	Saved	Description †128 BackRAM
						b4: Italics SET_ITALIC = %00010000 b3: Outline SET_OUTLINE = %00001000 b2: Superscript SET_SUPERSCRIPT = %00000100 bl: Subscript SET_SUBSCRIPT = %00000010 b0: Unused To Clear all flags (plain text) SET_PLAINTEXT = %00000000 Any combination of flags can be set or clear. If current mode is plaintext, all flags are clear. Constants that can be used within text strings themselves that affect currentMode are: UNDERLINEON, UNDERLINEOFF, REVERSEON, REVERSEOFF, BOLDON, ITALICON, OUTLINEON, PLAINTEXT
curSetWidth:	3c	3c	2	$00	Yes	Card width in pixels for the current font
curType:	88C6	88C6	1	Drive 8 Type	Np	Holds the current disk type. This value is copied from **driveType** for quicker access to the current drive b7: Set if the disk is a RAM disk b6: Set if using disk shadowing Only one of bit 6 or 7 may be set. Other constants used with **curType** are DRV_NULL = 0 No drive present at this device address DRV_1541 = 1 Drive type Commodore 1541 DRV_1571 = 2 Drive type Commodore 1571 **DRV_1581** = 3 Drive type Commodore 1581
curXpos0:	-	1303 †	16	None	No	The current X positions of the C128 soft sprites. BackRAM
curYposO:	-	1313 †	8	None	No	The current Y positions of the C128 soft sprites. BackRAM
dataFileName:	8442	8442	17	None	No	Name of a data file to open. The name is passed to the parent application so the file can be opened.
dataDiskName:	8453	8453	18	None	No	Holds the disk name that an application's data file is on.
dateCopy:	C018	C018	3	YMD	No	Copy of system Variables year, Month, and day.
day:	8518	8518	1	20	No	Holds the value for current **day**.
dblClickCount:	8515	8515	1	$00	No	Used to determine when an icon is double clicked on. When an icon is selected, **dblClickCount** is loaded with a value of CLICK_COUNT (30). **dblClickCount** is then decremented each interrupt. If the value is non-zero when the icon is again selected, then the double click flag (**r0H**) is passed to the service routine with a value of **TRUE**. If the

Address (hex)

Name	64	128	Size	Default	Saved	Description †128 BackRAM
						dblClickCount variable is zero when the icon is clicked on, then the flag is passed with a value of **FALSE**.
diskBlkBuf:	8000	8000	256	$00	No	General disk block buffer. Initialized to all zeros
dispBufferOn:	2F	2F	1	$C0	Yes	Routes graphic and text operations to either the foreground screen, background buffer, or both simultaneously. b7: 1 = draw to foreground screen buffer b6: 1 = draw to background buffer b5: 1 = Limit GetString text entry to foreground screen. 0 = GetString text entry will use b7,b6 b4-b0: reserved for future use? should always be 0 **ST_WR_FORE** = %10000000 ;$80 **ST_WR_BACK** = %01000000 ;%40 Default is **ST_WR_FORE** \| **ST_WR_BACK** ;$C0 Use ST_WR_FORE (write to foreground) and ST_WR_BACK (write to background) to access these bits. %00xxxxxxx is an undefined state and will result in sending most graphic operations to the center of the display area.
dlgBoxRamBuf:	851F	851F	417	None	Yes	This is the buffer for variables that are saved when desk accessories or dialog boxes are run.
doRestFlag:	-	1B54 †	1	$00	No	Flag needed because of overlapping soft sprite problems on C128. Set to TRUE if we see a sprite that needs to be redrawn and therefore all higher numbered sprites need to be redrawn as well. Resides in BackRAM.
driveType:	848E	848E	4	Drive 8 Type	No	There are 4 bytes at location **driveType**, one for each of four possible drives. Each byte has the following format: b7: Set if drive is RAM DISK b6: Set if Shadowed disk (Only 1 of bit 7 or bit 6 may be set) Constants and values used for drive types are Constant Value Description -------- ----- ----------- DRV_NULL = 0 ; No drive present at this device address

Address (hex)

Name	64	128	Size	Default	Saved	Description †128 BackRAM
						DRV_1541 = 1 ; Drive type Commodore 1541 DRV_1571 = 2 ; Drive type Commodore 1571 **DRV_1581** = 3 ; Drive type Commodore 1581
dir2Head:	8900	8900	256	None	No	1571,1581 Second BAM block
dir3Head:	9C80	9C80	256	None	No	1581 Third BAM block
diskOpenFlg:	848A	848A	1	$00	No	This flag byte is not used by the Kernal. It is initialized to $00 when the entire block is cleared at startup. It is never touched again by the Kernal. It is used by the DeskTop. The flag follows the status of the currently selected drive. If the disk is open this byte is set to **TRUE**. If you close the disk using DeskTop it changes this byte to **False**. This byte could be freely used by applications to perform the same function as the DeskTop (or for any other purpose as well). But it would be up to the Application to set and maintain the value of the byte.
dlgBoxVector:	44	44	2	None	No	
DrACurDkNm:	841E	841E	16	None	No	Disk name of the current disk in drive A, padded with $A0
DrBCurDkNm:	8430	8430	16	None	No	Disk name of the current disk in drive B, padded with $A0
DrCCurDkNm:	88DC	88DC	16	None	No	Disk name of the current disk in drive C, padded with $A0
DrDCurDkNm:	88EE	88EE	16	None	No	Disk name of the current disk in drive D, padded with $A0
driveData:	88BF	88BF	4	None	No	One byte is reserved for each disk drive, to be used by the with disk driver. Each driver may use it differently.
iconSelFlag:	84B5	84B5	1	$00	Yes	Flag bits in b7 and b6 specify how the system should indicate icon selection to the user. If no bits are set, then the system does nothing to indicate icon selection, and the service routine is simply called. The possible flags are: **ST_FLASH** = $80 ; flash the icon **ST_INVERT** = $40 ; invert the selected icon If **ST_FLASH** is set, the **ST_INVERT** flag is ignored and the icon flashes but is not inverted when the programmer's routine is called. If ST_INVERT is set, and ST_FLASH is CLEAR, then the icon will be inverted when the programmer's routine is called.
dirEntryBuf:	8400	8400	30	$00	No	Buffer used to build a file's directory entry.

Address (hex)

Name	64	128	Size	Default	Saved	Description †128 BackRAM
extclr:	20	20	1	$FB	No	exterior (border) color.
faultData:	84B6	84B6	1	$00	Yes	Holds Information about mouse faults. Mouse faults occur when the mouse attempts to move outside the bounds set by mouseLeft, mouseRight, mouseTop, and mouseBottom. A fault is also signaled when the mouse is outside the current menu area. The bits for signaling are used as follows: Bit Fault Constant for bit access --- ----- ----------------------- b7: mouse fault up OFFTOP_BIT b6: mouse fault down OFFBOTTOM_BIT b5: mouse fault left OFFLEFT_BIT b4: mouse fault right OFFRIGHT_BIT b3: menu fault OFFMENU_BIT
fileHeader:	8100	8100	256	$00	No	Header Block buffer for a GEOS file.
fileSize:	8499	8499	2	None	No	Current size (in blocks) of a file. It is pulled in from and written to the file's directory entry.
fileTrScTab:	8300	8300	256	$00	No	Track and Sector chain for a file of maximum size of 32258 bytes.
fileWritten:	8498	8498	1	None	No	Flag indicating if the currently open file has been written to since the last update of its index table and the BAM.
firstBoot:	88C5	88C5	1	$00	No	This flag is changed from $00 to $FF when the deskTop comes up after booting.
fontData:	850C	850C	9	None	No	Buffer for saving the user active font table when going into menus.
fontTable:	26	26	8	Default Font	Yes	**fontTable** is a label for the beginning of variables for the current font in use. These variables are **baselineOffset**, **curSetWidth**. **curHeight**, **curIndexTable**, and **cardDataPntr**. For more information, see documentation on these variables.
graphMode:	3F	3F	1	None	No	Current video mode for C128. **40-Column: GR_40 ($00)** **80-Column: GR_80 ($80) (%10000000)** **sample usage graphMode** **bit graphMode** **bmi Do80ColStuff**

Address (hex)

Name	64	128	Size	Default	Saved	Description †128 BackRAM
grcntrl1:	D011	D011	1	None	No	graphics control register #1, ie msb raster /ECM /BMM /DEN /RSEL /y scroll bits. defined for use with above register ST_ECM = $40 ST_BCM = $20 ST_DEN = $10 ST_25ROW = $08
numDrives:	848D	848D	1	Actual	No	Number of drives in the system
turboFlags:	8492	8492	0	$00	No	Turbo state flags for drives 8 through 11 Flag Byte Layout. bit 7 = 1 Turbo is Loaded. bit 6 = 1 Turbo is Active bit 0-5 Always Zero. **diskOpenFlg** can be used as a base to index into this table by drive number. Example ldy **curDrive** lda **diskOpenFlg**,y

Note[3]: Next firstBoot pg 547

Unused Next Section

```
;Dumping Ground for Wheels info until it gets organized
; Wheels
; these are addresses to routines that are in the extended
; kernal that get loaded in at $5000 in groups.
```