This work is in a very early stage.

The Goal of this Document is

1. combine the collective works from:

The Hitchhikers to GEOS

by Berkely Softworks 1988

Geos Programmer’s Reference Guide by

by Alexander Donald Boyce 1986

Revised by Bo Zimmerman 1997

The Official GEOS Programmers Reference Guide

by Berkeley Softworks 1987

Information now available from the Dissembled GEOS Kernal

Information Obtained from my Disassembly of GEOS Applications.

If possible. All new KERNAL additions from Wheels 4.4 and from MP3+

MP3 is still being actively developed and it is open source so this should not be a problem. Wheels 4.4 internals has been dissembled so this should be manageable as well but I have not yet started on either of these additions.

Any and all other relevant information from others sources including from my own experience developing for GEOS. With appropriate credits given for all source Document s.

1. Make all information in the document Hyperlinked.
2. Remove all Apple Information from this collection. When I get geoAssembler ported into the Apple GEOS world There will be another document made from this one with all the apple information in it.

This will provide a one stop resource for programming information.

The slowest part of the process is converting the OCR Scanned originals back into editable document form that can be easily edited.

Volunteers are welcome to assist...

My goal is too add 1 page a day to this document until it is completed.

Paul B Murdaugh

Writer of Dual Top and the Landmark Series for GEOS

paulbmurdaugh@gmail.com

Status

Chapter 1. [**GEOS Kernal**](#Catagory_Listing) **(Growing)**

**Chapter 2.** [**Examples**](#MyFreeBlock) **(growing)**

**Chapter 3. Memory Map (Almost ready to merge in)**

**Chapter 4. 128 BackRAM Memory Map (planned)**

**Chapter 5. REU Bank 0 Memory Map (planned)**

**Appendix I [Disk Errors](#apDiskErrors) (done)**

# GEOS Kernal by Name

Name Addr Description Category Page

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| [[**AllocateBlock**](#AllocateBlock)](#AllocateBlock) | 9048 | Mark a disk block as in-use. | [**disk mid-level**](#disk_mid_level) | 31 |
| [**AppendRecord**](#AppendRecord) | C289 | Insert a new VLIR record after the current record. | [**disk VLIR**](#disk_vlir) | 71 |
| [BBMult](#BBMult) | C160 | Byte by byte (single-precision) unsigned multiply. | [**math**](#math) | 106 |
| [**Bell**](#Bell) | N/A | Play a bell sound | [**utility**](#Utility) | 150 |
| [**BldGDirEntry**](#BldGDirEntry) | C1F3 | Build a GEOS directory entry in memory. | [disk mid-level](#disk_mid_level) | 32 |
| [**BlkAllOC**](#BlkAllOC) | C1FC | Allocate sectors for a file. | [disk mid-level](#disk_mid_level) | 33 |
| [**BlockProcess**](#BlockProcesses) | C10C | Block process from running. Does not freeze timer. | [**process**](#process) | 141 |
| [Bmult](#Bmult) | C163 | Byte by word unsigned multiply. | [**math**](#math) | 107 |
| [BootGeos](#BootGeos) | C000 | Reboot GEOS | [**internal**](#internal) | 102 |
| [[**CalcBlksFree**](#CalcBlksFree)](#CalcBlksFree) | C1DB | Calculate total number of free disk blocks. | [disk mid-level](#disk_mid_level) | 34 |
| [[**CallRoutine**](#CallRoutine)](#CallRoutine) | C1D8 | pseudo-subroutine call. $0000 aborts call. | [**utility**](#Utility) | 151 |
| [**ChangeDiskDevice**](#ChangeDiskDevice) | C2BC | Change disk drive device number. | [disk very low-level](#disk_very_low_level) | 17 |
| [**ChkDkGEOS**](#ChkDkGEOS) | C1DE | Check if a disk is GEOS format. | [disk mid-level](#disk_mid_level) | 35 |
| [**ClearRam**](#ClearRam) | C178 | Clear memory to $00. | [**memory**](#memory) | 118 |
| [**CloseRecordFile**](#CloseRecordFile) | C277 | Close/Save currently open VLIR file. | [**disk VLIR**](#disk_vlir) | 72 |
| [**CmpFString**](#CmpFString) | C26E | Compare two fixed-length strings. | [**memory**](#memory) | 119 |
| [**CmpString**](#CmpString) | C26B | Compare two null-terminated strings. | [**memory**](#memory) | 120 |
| [**CopyFString**](#CopyFString) | C268 | Copy a fixed-length string. | [**memory**](#memory) | 121 |
| [**CopyString**](#CopyString) | C265 | Copy a null-terminated string. | [**memory**](#memory) | 122 |
| [**CRC**](#CRC) | C20E | Cyclic Redundancy Check calculation. | [**utility**](#Utility) | 152 |
| [**Dabs**](#Dabs) | C16F | Double-precision signed absolute value. | [**memory**](#memory) | 108 |
| [[**DeleteFile**](#DeleteFile)](#DeleteFile) | C238 | Delete file. | [disk high-level](#disk_high_level) | 56 |
| [**DeleteRecord**](#DeleteRecord) | C283 | Delete current VLIR record. | [**disk VLIR**](#disk_vlir) | 73 |
| [**Ddec**](#DDec) | C175 | Double-precision unsigned decrement. | [**math**](#math) | 109 |
| [[**Ddiv**](#Ddiv)](#Ddiv) | C169 | Double-precision unsigned division. | [**math**](#math) | 110 |
| [**DMult**](#DMult) | C166 | Double-precision unsigned multiply. | [**math**](#math) | 112 |
| [**Dnegate**](#Dnegate) | C172 | Double-precision signed negation. | [**math**](#math) | 113 |
| [**DoBOp**](#DoBOp) | C2EC | (128) Back-RAM memory primitive | [**memory**](#memory) | 123 |
| [**DoDlgBox**](#DoDlgBox) | C256 | Display and begin interaction w/dialog box. | [**dialog box**](#dialog_box) | 14 |
| [[[[[[[[**DoIcons**](#DoIcons)](#UpdateRecordFile)](#UpdateRecordFile)](#UpdateRecordFile)](#WriteRecord)](#UpdateRecordFile)](#UpdateRecordFile)](#UpdateRecordFile) | C15A | Display and begin interaction with icons. | [****icon/menu****](#icon_menu) | 93 |
| [[**DoInlineReturn**](#DoInlineReturn)](#DoInlineReturn) | C2A4 | Return from inline subroutine. | [****utility****](#Utility) | 153 |
| [[[[[[[[**DoMenu**](#DoMenu)](#UpdateRecordFile)](#UpdateRecordFile)](#UpdateRecordFile)](#WriteRecord)](#UpdateRecordFile)](#UpdateRecordFile)](#UpdateRecordFile) | C151 | Display and begin interaction with menus. | [****icon/menu****](#icon_menu) | 94 |
| [**DoRAMOp**](#DoRamOp) | C2D4 | RAM-expansion unit access primitive. | [**memory**](#memory) | 130 |
| [**DoneWithIO**](#DoneWithIO) | C25F | Restore system after serial I/O. | [disk very low-level](#disk_very_low_level) | 18 |
| [**DrawPoint**](#DrawPoint) | C133 | Draw, clear, or recover a single screen point. | [**graphics**](#graphics) | 90 |
| [**DSDiv**](#DSDiv) | C16C | Double-precision signed division. | [**math**](#math) | 114 |
| [**DShiftLeft**](#DShiftLeft) | C15D | Double-precision left shift (zeros shifted in). | [**math**](#math) | 115 |
| [**DShiftRight**](#DShiftRight) | C262 | Double-precision right shift (zeros shifted in). | [**math**](#math) | 116 |
| [**EnableProcess**](#EnableProcess) | C109 | Make a process runnable immediately. | [**process**](#process) | 144 |
| [**EnterDeskTop**](#EnterDeskTop) | C22C | Leave application and return to GEOS deskTop. | [disk high-level](#disk_high_level) | 57 |
| [**EnterTurbo**](#EnterTurbo) | C214 | Activate disk turbo on current drive. | [disk very low-level](#disk_very_low_level) | 19 |
| [**ExitTurbo**](#ExitTurbo) | C232 | Deactivate disk turbo on current drive. | [disk very low-level](#disk_very_low_level) | 20 |
| [**FastDelFile**](#FastDelFile) | C244 | Quick file delete (requires full track/sector list). | [disk mid-level](#disk_mid_level) | 36 |
| [**FetchRAM**](#FetchRam) | C2CB | Transfer data from RAM-expansion unit. | [**memory**](#memory) | 131 |
| [**FillRam**](#FillRam) | C17B | Fill memory with a particular byte. | [**memory**](#memory) | 124 |
| [[[[**FindBAMBit**](#FindBAMBit)](#FastDelFile)](#FindBAMBit)](#FastDelFile) | C2AD | Get allocation status of particular disk block. | [disk mid-level](#disk_mid_level) | 37 |
| [[**FindFile**](#FindFile)](#FindFile) | C20B | Search for a particular file. | [disk high-level](#disk_high_level) | 58 |
| [[**FindFTypes**](#FindFTypes)](#FindFile) | C23B | Find all files of a particular GEOS type. | [disk high-level](#disk_high_level) | 59 |
| [[[[[[[[FirstInit](#FirstInit)](#BootGeos)](#Firstlnit)](#BootGeos)](#Firstlnit)](#BootGeos)](#Firstlnit)](#BootGeos) | C271 | Initialize GEOS variables. | [**internal**](#internal) | 103 |
| [[[[**FollowChain**](#FollowChain)](#FastDelFile)](#FindBAMBit)](#FastDelFile) | C205 | Follow chain of sectors, building track/sector table. | [disk mid-level](#disk_mid_level) | 38 |
| [**FreeBlock**](#FreeBlock) | C2B9 | Mark a disk block as not-in-use in **BAM**. | [disk mid-level](#disk_mid_level) | 39 |
| [[**FreeFile**](#FreeFile)](#FreeFile) | C226 | Free all blocks associated with a file. | [disk mid-level](#disk_mid_level) | 40 |
| [**FreezeProcess**](#FreezeProcess) | C112 | Pause a process countdown timer. | [**process**](#process) | 142 |
| [**GetlstDirEntry**](#GetlstDirEntry) | 9030 | Loads in the first directory block. | [disk mid-level](#disk_mid_level) | 40 |
| [**GetBlock**](#GetBlock) | C1E4 | Read single disk block into memory. | [disk low-level](#disk_low_level) | 28 |
| [**GetCharWidth**](#GetCharWidth) | C1C9 | Calculate width of char without style attributes. | [**text**](#text) | 157 |
| [**GetDirHead**](#GetDirHead) | C247 | Read directory header into memory. | [disk mid-level](#disk_mid_level) | 43 |
| [**GetFile**](#GetFile) | C208 | Load GEOS file. | [disk high-level](#disk_high_level) | 61 |
| [[[**GetFHdrInfo**](#GetFHdrInfo)](#GetFHdrlnfo)](#GetFHdrlnfo) | C229 | Read a GEOS file header into **fileHeader**. | [disk mid-level](#disk_mid_level) | 44 |
| [[[[**GetFreeDirBlk**](#GetFreeDirBlk)](#GetFHdrlnfo)](#GetFHdrInfo)](#GetFHdrlnfo) | C1F6 | Find an empty directory slot. | [disk mid-level](#disk_mid_level) | 45 |
| [**GetNextChar**](#GetNextChar) | C2A7 | Get next character from keyboard queue. | [**text**](#text) | 158 |
| [**GetNxtDirEntry**](#GetNxtDirEntry) | 9033 | Get directory entry other than first. | [disk mid-level](#disk_mid_level) | 42 |
| [**GetOffPageTrSc**](#GetOffPageTrSc) | 9036 | Get track and sector of off-page directory. | [disk mid-level](#disk_mid_level) | 47 |
| [[**GetPtrCurDkNm**](#GetPtrCurDkNm)](#FindFile) | C298 | Return pointer to current disk name. | [disk mid-level](#disk_mid_level) | 63 |
| [[[[**GetRandom**](#GetRandom)](#CallRoutine)](#GetRandom)](#CallRoutine) | C187 | Calculate new random number. | [**utility**](#Utility) | 154 |
| [**GetRealSize**](#GetRealSize) | C1B1 | Calculate actual character size with attributes. | [**text**](#text) | 159 |
| [[GetSerialNumber](#GetSerialNumber)](#BootGeos) | C196 | Return GEOS serial number. | [**internal**](#internal) | 104 |
| [**GetString**](#GetString) | C1BA | Get string input from user. | [**text**](#text) | 160 |
| **i\_**[**FillRam**](#FillRam) | C1B4 | Inline [**FillRam**](#FillRam). | [**memory**](#memory) | 124 |
| [**i­\_MoveData**](#MoveData) | C1B7 | Inline [**MoveData**](#MoveData). | [**memory**](#memory) | 127 |
| [**InitForIO**](#InitForIO) | C25C | Prepare system for serial I/O. | [disk very low-level](#disk_very_low_level) | 21 |
| [InitMouse](#InitMouse) | FE80 | Initialize input device. | [**input driver**](#input_driver) | 99 |
| [**InitProcesses**](#InitProcesses) | C103 | Initialize processes. | [**process**](#process) | 143 |
| [**InitRam**](#InitRam) | C181 | Initialize memory areas from table. | [**memory**](#memory) | 125 |
| [**InsertRecord**](#InsertRecord) | C286 | Insert new VLIR record in front of current record. | [**disk VLIR**](#disk_vlir) | 74 |
| [[**LdApplic**](#LdApplic)](#LdApplic) | C21D | Load GEOS application. | [disk mid-level](#disk_mid_level) | 48 |
| [**LdDeskAcc**](#LdDeskAcc) | C217 | Load GEOS desk accessory. | [disk mid-level](#disk_mid_level) | 50 |
| [[**LdFile**](#LdFile)](#LdDeskAcc) | C211 | Load GEOS data file. | [disk mid-level](#disk_mid_level) | 52 |
| [**MoveBData**](#MoveBData) | C2E3 | 128 BackRAM memory move routine. | [**memory**](#memory) | 126 |
| [**MoveData**](#MoveData) | C17E | Intelligent memory block move. | [**memory**](#memory) | 127 |
| [**NewDisk**](#NewDisk) | C1E1 | Initialize a drive. | [disk mid-level](#disk_mid_level) | 53 |
| [**NextRecord**](#NextRecord) | C27A | Make next VLIR the current record. | [**disk VLIR**](#disk_vlir) | 75 |
| [**OpenDisk**](#OpenDisk) | C2A1 | Open disk in current drive. | [disk high-level](#disk_high_level) | 63 |
| [**OpenRecordFile**](#OpenRecord) | C274 | Open VLIR file on current disk. | [**disk VLIR**](#disk_vlir) | 76 |
| [**PointRecord**](#PointRecord) | C280 | Make specific VLIR record the current record. | [**disk VLIR**](#disk_vlir) | 77 |
| [**PreviousRecord**](#PreviousRecord) | C27D | Make previous VLIR record the current record. | [**disk VLIR**](#disk_vlir) | 78 |
| [**PutBlock**](#PutBlock) | C1E7 | Write single disk block from memory. | [disk low-level](#disk_low_level) | 29 |
| [**PutDecimal**](#PutDecimal) | C184 | Format and display an unsigned double-precision nbr. | [**text**](#text) | 163 |
| [**ReadBlock**](#ReadBlock) | C21A | Get disk block primitive. | [disk mid-level](#disk_mid_level) | 23 |
| [**ReadByte**](#ReadByte) | C2B6 | Read a File 1 byte at a time. | [disk mid-level](#disk_mid_level) | 54 |
| [**ReadLink**](#ReadLink) | **904B** | **Read track/sector link.** | [disk mid-level](#disk_mid_level) | 24 |
| [**ReadRecord**](#ReadRecord) | C28C | Read current VLIR record into memory. | [**disk VLIR**](#disk_vlir) | 79 |
| [[[[[[[[[[[[[[[[**RecoverMenu**](#RecoverMenu)](#UpdateRecordFile)](#UpdateRecordFile)](#UpdateRecordFile)](#WriteRecord)](#UpdateRecordFile)](#UpdateRecordFile)](#UpdateRecordFile)](#RecoverMenu)](#UpdateRecordFile)](#UpdateRecordFile)](#UpdateRecordFile)](#WriteRecord)](#UpdateRecordFile)](#UpdateRecordFile)](#UpdateRecordFile) | C154 | Recover single menu from background buffer. | [**icon/menu**](#icon_menu) | 96 |
| [[[[[[[[[[[[[[[[**ReDoMenu**](#ReDoMenu)](#UpdateRecordFile)](#UpdateRecordFile)](#UpdateRecordFile)](#WriteRecord)](#UpdateRecordFile)](#UpdateRecordFile)](#UpdateRecordFile)](#ReDoMenu)](#UpdateRecordFile)](#UpdateRecordFile)](#UpdateRecordFile)](#WriteRecord)](#UpdateRecordFile)](#UpdateRecordFile)](#UpdateRecordFile) | C193 | Reactivate menus at the current level. | [**icon/menu**](#icon_menu) | 97 |
| [**RenameFile**](#RenameFile) | C259 | Rename GEOS disk file. | [disk mid-level](#disk_mid_level) | 65 |
| [**RestartProcess**](#RestartProcess) | C106 | Unblock, unfreeze, and restart process. | [**process**](#process) | 145 |
| [**SaveFile**](#SaveFile) | C1ED | Save Memory to create a GEOS file. | [disk high-level](#disk_high_level) | 67 |
| [**SetDevice**](#SetDevice) | C2B0 | Establish communication with a new serial device. | [disk high-level](#disk_high_level) | 68 |
| [**SetGEOSDisk**](#SetGEOSDisk) | C1EA | Convert normal CBM disk into GEOS format disk. | [disk high-level](#disk_high_level) | 69 |
| [SetMouse](#SetMouse) | FE89 | Reset input device scanning circuitry. | [**input driver**](#input_driver) | 100 |
| [**SetMsePic**](#SetMsePic) | C2DA | Set and preshift new soft-sprite mouse picture. | [**mouse/sprite**](#mouse_sprite) | 136 |
| [**SetNewMode**](#SetNewMode) | C2DD | Change GEOS 128 graphics mode (40/80 switch). | [**graphics**](#graphics) | 91 |
| [**Sleep**](#Sleep) | C199 | Put current subroutine to sleep for a specified time. | [**process**](#process) | 146 |
| [**SmallPutChar**](#SmallPutChar) | C202 | Fast character print routine. | [**text**](#text) | 162 |
| [**StartASCII**](#StartASCII) | 7912 | Begin ASCII mode printing. | [**print driver**](#print_driver) | 138 |
| [**StashRAM**](#StashRAM) | C2C8 | Transfer memory to RAM-expansion unit. | [**memory**](#memory) | 132 |
| [**SwapBData**](#SwapBData) | C2E6 | 128 memory swap between front/back ram. | [**memory**](#memory) | 128 |
| [**SwapRAM**](#SwapRAM) | C2CE | RAM-expansion unit memory swap. | [**memory**](#memory) | 133 |
| [**RstrFrmDialog**](#RstrFrmDialog) | C2BF | Exits from a dialog box. | [**dialog box**](#dialog_box) | 15 |
| [**TempHideMouse**](#TempHideMouse) | C2D7 | Hide soft-sprites before direct screen access. | [**mouse/sprite**](#mouse_sprite) | 137 |
| [[[[[[[[[[[[[[[[[[[[**ToBasic**](#ToBasic)](#CallRoutine)](#GetRandom)](#CallRoutine)](#GetRandom)](#CallRoutine)](#GetRandom)](#CallRoutine)](#ToBasic)](#CallRoutine)](#GetRandom)](#CallRoutine)](#GetRandom)](#CallRoutine)](#GetRandom)](#CallRoutine)](#GetRandom)](#CallRoutine)](#GetRandom)](#CallRoutine) | C241 | Pass Control to Commodore BASIC. | [**utility**](#Utility) | 155 |
| [**UnblockProcess**](#UnblockProcess) | C10F | Unblock a blocked process, allowing it to run again. | [**process**](#process) | 146 |
| [**UnfreezeProcess**](#UnfreezeProcess) | C115 | Unpause a frozen process timer. | [**process**](#process) | 148 |
| [[[[**UpdateRecordFile**](#UpdateRecordFile)](#UpdateRecordFile)](#UpdateRecordFile)](#UpdateRecordFile) | C295 | Update currently open VLIR file without closing | [**disk VLIR**](#disk_vlir) | 80 |
| [**VerifyBData**](#VerifyBData) | **C2E9** | 128 BackRAM verify. | [**memory**](#memory) | 129 |
| [**VerifyRAM**](#VerifyRAM) | C2D1 | RAM-expansion unit verify. | [**memory**](#memory) | 134 |
| [**VerWriteBlock**](#VerWriteBlock) | C223 | Disk block verify primitive. | [disk very low-level](#disk_very_low_level) | 25 |
| [**WriteBlock**](#WriteBlock) | C220 | Write disk block primitive. | [disk very low-level](#disk_very_low_level) | 26 |
| [[[[[[[[**WriteRecord**](#WriteRecord)](#UpdateRecordFile)](#UpdateRecordFile)](#UpdateRecordFile)](#WriteRecord)](#UpdateRecordFile)](#UpdateRecordFile)](#UpdateRecordFile) | C28F | Write current VLIR record to disk. | [**disk VLIR**](#disk_vlir) | 81 |

# GEOS Kernal by Category

dialog box

------------------- ---- -------------------------------------------------- ---

[**DoDlgBox**](#DoDlgBox) C256 Display and begin interaction w/dialog box. 14

**[RstrFrmDialog](#RstrFrmDialog)** C2BF Exits from a dialog box. 15

disk very Low level

------------------- ---- -------------------------------------------------- ---

[**ChangeDiskDevice**](#ChangeDiskDevice)C2BC Change disk drive device number. 17

**[DoneWithIO](#DoneWithIO)** C25F Restore system after serial I/O. 18

**[EnterTurbo](#EnterTurbo)** C214 Activate disk turbo on current drive. 19

**[ExitTurbo](#ExitTurbo)** C232 Deactivate disk turbo on current drive. 20

**[InitForIO](#InitForIO)** C25C Prepare system for serial I/O. 21

**[PurgeTurbo](#PurgeTurbo)** C235 Remove disk turbo from current drive. 21

[**ReadBlock**](#ReadBlock)C21A Get disk block primitive. 23

**[ReadLink](#ReadLink) 904B Read track/sector link.** 24

**[VerWriteBlock](#VerWriteBlock)** C223 Disk block verify primitive. 25

**[WriteBlock](#WriteBlock)** C220 Write disk block primitive. 26

disk low level

------------------- ---- -------------------------------------------------- ---

[**GetBlock**](#GetBlock) C1E4 Read single disk block into memory. 28

**[PutBlock](#PutBlock)** C1E7 Write single disk block from memory. 29

disk mid-level

------------------- ---- -------------------------------------------------- ---

[[**AllocateBlock**](#AllocateBlock)](#AllocateBlock) 9048 Mark a disk block as in-use.31

[**BldGDirEntry**](#BldGDirEntry) C1F3 Build a GEOS directory entry in memory.32

**[BlkAllOC](#BlkAllOC)** C1FC Allocate sectors for a file.33

[**[CalcBlksFree](#CalcBlksFree)**](#CalcBlksFree) C1DB Calculate total number of free disk blocks.34

**[ChkDkGEOS](#ChkDkGEOS)** C1DE Check if a disk is GEOS format.35

**[FastDelFile](#FastDelFile)** C244 Quick file delete (requires full track/sector list). 36

[[[**[FindBAMBit](#FastDelFile)**](#FindBAMBit)](#FastDelFile)](#FindBAMBit) C2AD Get allocation status of particular disk block. 37

[[[**[FollowChain](#FastDelFile)**](#FollowChain)](#FastDelFile)](#FindBAMBit) C205 Follow chain of sectors, building track/sector table. 38

**[FreeBlock](#FreeBlock)** C2B9 Mark a disk block as not-in-use in **BAM**.39

**[FreeFile](#FreeFile)** C226 Free all blocks associated with a file. 40

**[GetlstDirEntry](#GetlstDirEntry)** 9030 Loads in the first directory block.40

**[GetNxtDirEntry](#GetNxtDirEntry)** 9033 Get directory entry other than first.42

**[GetDirHead](#GetDirHead)**  C247 Read track 18 sector 0. 43

[[**[GetFHdrInfo](#GetFHdrlnfo)**](#GetFHdrInfo)](#GetFHdrlnfo) C229 Read a GEOS file header into fileHeader. 44

[[[**[GetFreeDirBlk](#GetFHdrlnfo)**](#GetFreeDirBlk)](#GetFHdrlnfo)](#GetFHdrInfo) C1F6 Find an empty directory slot. 45

**[GetOffPageTrSc](#GetOffPageTrSc)** 9036 Get track and sector of off-page directory.47

[**[LdApplic](#LdApplic)**](#LdApplic) C21D Load GEOS application. 48

**[LdDeskAcc](#LdDeskAcc)** C217 Load GEOS desk accessory. 50

[**[LdFile](#LdDeskAcc)**](#LdFile) C211 Load GEOS data file. 52

**[NewDisk](#NewDisk)** C1E1 Initialize a drive.53

NxtBlkAlloc $C24D Allocate sectors for a file. -319

PutDirHead $C24A Write to track 18 sector 0. -309

**[ReadByte](#ReadByte)** C2B6 Read a File 1 byte at a time.54

ReadFile $C1FF Load a chain of blocks into memory. -415

SetGDirEntry $C1F0 Create a directory entry on disk. -417

SetNextFree $C292 Find and allocate a disk block. -317

StartAppl $C22F Run a program that is in memory. -807

WriteFile $C1F9 Save memory to preallocated sectors. -404

disk high level

------------------- ---- -------------------------------------------------- ---

[[**DeleteFile**](#DeleteFile)](#DeleteFile) C238 Delete file.56

[**EnterDeskTop**](#EnterDeskTop) C22C Leave application and return to GEOS deskTop.57

[**[FindFile](#FindFile)**](#FindFile) C20B Search for a particular file.58

[**[FindFTypes](#FindFile)**](#FindFTypes) C23B Find all files of a particular GEOS type. 59

**[GetFile](#GetFile)** C208 Load GEOS file. 61

[**[GetPtrCurDkNm](#FindFile)**](#GetPtrCurDkNm) C298 Return pointer to current disk name. 63

**[OpenDisk](#OpenDisk)** C2A1 Open disk in current drive. 63

**[RenameFile](#RenameFile)** C259 GEOS disk file. 65

RstrAppl $C23E Load the SWAPFILE -409

[**SaveFile**](#SaveFile) $C1ED Save Memory to create a GEOS file. 67

**[SetDevice](#SetDevice)** C2B0 Establish communication with a new serial device. 68

**[SetGEOSDisk](#SetGEOSDisk)** C1EA Convert normal CBM disk into GEOS format disk. 69

disk VLIR

------------------- ---- -------------------------------------------------- ---

[**AppendRecord**](#AppendRecord) C289 Insert a new VLIR record after the current record. 71

**[CloseRecordFile](#CloseRecordFile)** C277 Close/Save currently open VLIR file. 72

**[DeleteRecord](#DeleteRecord)** C283 Delete current VLIR record. 73

**[InsertRecord](#InsertRecord)** C286 Insert new VLIR record in front of current record. 74

**[NextRecord](#NextRecord)** C27A Make next VLIR the current record. 75

**[OpenRecordFile](#OpenRecord)** C274 Open VLIR file on current disk. 76

**[PointRecord](#PointRecord)** C280 Make specific VLIR record the current record. 77

**[PreviousRecord](#PreviousRecord)** C27D Make previous VLIR record the current record. 78

**[ReadRecord](#ReadRecord)** C28C Read current VLIR record into memory. 79

[[[**[UpdateRecordFile](#UpdateRecordFile)**](#UpdateRecordFile)](#UpdateRecordFile)](#UpdateRecordFile) C295 Update currently open VLIR file without closing. 80

[[[[[[[**[WriteRecord](#UpdateRecordFile)**](#WriteRecord)](#UpdateRecordFile)](#UpdateRecordFile)](#UpdateRecordFile)](#WriteRecord)](#UpdateRecordFile)](#UpdateRecordFile) C28F Write current VLIR record to disk. 81

icon/menu

------------------- ---- -------------------------------------------------- ---

[[[[[[[**[DoIcons](#UpdateRecordFile)**](#DoIcons)](#UpdateRecordFile)](#UpdateRecordFile)](#UpdateRecordFile)](#WriteRecord)](#UpdateRecordFile)](#UpdateRecordFile) C15A Display and begin interaction with icons.93

[[[[[[[**[DoMenu](#UpdateRecordFile)**](#DoMenu)](#UpdateRecordFile)](#UpdateRecordFile)](#UpdateRecordFile)](#WriteRecord)](#UpdateRecordFile)](#UpdateRecordFile) C151 Display and begin interaction with menus. 94

DoPreviousMenu $C190 Close current menu 1-702

GotoFirstMenu $C1BD Close all menu levels 1-703

RecoverAllMenus $C157 Erase all menus 1-706

[[[[[[[[[[[[[[[**[RecoverMenu](#UpdateRecordFile)**](#RecoverMenu)](#UpdateRecordFile)](#UpdateRecordFile)](#UpdateRecordFile)](#WriteRecord)](#UpdateRecordFile)](#UpdateRecordFile)](#UpdateRecordFile)](#RecoverMenu)](#UpdateRecordFile)](#UpdateRecordFile)](#UpdateRecordFile)](#WriteRecord)](#UpdateRecordFile)](#UpdateRecordFile) C154 Recover single menu from background buffer. 96

[[[[[[[[[[[[[[[**[ReDoMenu](#UpdateRecordFile)**](#ReDoMenu)](#UpdateRecordFile)](#UpdateRecordFile)](#UpdateRecordFile)](#WriteRecord)](#UpdateRecordFile)](#UpdateRecordFile)](#UpdateRecordFile)](#ReDoMenu)](#UpdateRecordFile)](#UpdateRecordFile)](#UpdateRecordFile)](#WriteRecord)](#UpdateRecordFile)](#UpdateRecordFile) C193 Reactivate menus at the current level. 97

input driver

------------------- ---- -------------------------------------------------- ---

[InitMouse](#InitMouse) FE80 Initialize input device. 99

[SetMouse](#SetMouse) FE89 Reset input device scanning circuitry. 100

[SetMouse](#SetMouse)Pic $c2da Set and preshift new soft-sprite mouse picture.

SlowMouse $fe83 Reset mouse velocity variables.

UpdateMouse $fe86 Update mouse variables from input device.

internal

------------------- ---- -------------------------------------------------- ---

[BootGeos](#BootGeos) C000 Reboot GEOS. 102

[[[[[[[[FirstInit](#BootGeos)](#FirstInit)](#BootGeos)](#Firstlnit)](#BootGeos)](#Firstlnit)](#BootGeos)](#Firstlnit) C271 Initialize GEOS variables. 103

[[GetSerialNumber](#BootGeos)](#GetSerialNumber) C196 Return GEOS serial number. 104

InterruptMain $C100 Main interrupt level processing. 1-805

MainLoop $C1C3 GEOS's main loop. 1-804

Panic $C2C2 Report system errors. 1-711

ResetHandle $C003 internal Bootstrap entry point

graphics

------------------- ---- -------------------------------------------------- ---

BitmapUp $C1AB Draw a click box -621

i\_BitmapUp $C1AB Draw a click box / inline -622

BitmapClip $C2AA Draw a coded image -623

BitOtherClip $C2C5 Draw a coded image with user patches -624

DisablSprite $C1D5 Turn off a sprite -628

DrawLine $C130 Draw/Erase/Copy an arbitrary line -615

[**DrawPoint**](#DrawPoint) $C133 Draw, clear, or recover a single screen point.90

DrawSprite $C1C6 Copy a sprite data block -625

EnablSprite $C1D2 Turn on a sprite -627

FrameRectangle $C127 Draw an outline in a pattern -604

i\_FrameRectangle $C1A2 Draw a solid outline with inline data -605

GetScanLine $C13C Compute memory address of a row on the screen -618

GraphicsString $C136 Process a graphic command table -601

i\_GraphicsString $C1A8 Process a graphic command table / inline -602

HorizontalLine $C118 Draw a horizontal line in a pattern -616

InvertLine $C11B Reverse video a horizontal line -614

ImprintRectangle $C250 Copy a box from screen 2 to screen 1 -610 i\_ImprintRectangle $C253 Copy a box from screen 2 to screen 1 / inline -611

InvertRectangle $C12A Reverse video a box -612

PosSprite $C1CF Position a sprite -626

RecoverLine $C11E Copy a line from screen 2 to screen 1 -613

Rectangle $C124 Fill a box with a pattern -606

i\_Rectangle $C19F Fill a box with a pattern / inline -607

RecoverRectangle $C12D Copy a box from screen 1 to screen 2 -608

i\_RecoverRectangle $C1A5 Copy a box from screen 1 to screen 2 / inline -609

[**SetNewMode**](#SetNewMode) $C2DD Change GEOS 128 graphics mode (40/80 switch).91

SetPattern $C139 Select a fill pattern -603

TestPoint $C13F Test the value of a pixel -619

VerticalLine $C121 Draw a vertical line in a pattern -617

math

------------------- ---- -------------------------------------------------- ---

[BBMult](#BBMult) C160 Byte by byte (single-precision) unsigned multiply. 106

[Bmult](#Bmult) C163 Byte by word unsigned multiply. 107

**[Dabs](#Dabs)** C16F Double-precision signed absolute value. 108

**[Ddec](#DDec)** C175 Double-precision unsigned decrement. 109

[[**Ddiv**](#Ddiv)](#Ddiv)C169 Double-precision unsigned division. 110

**[DMult](#DMult)** C166 Double-precision unsigned multiply. 112

**[Dnegate](#Dnegate)** C172 Double-precision signed negation. 113

**[DSDiv](#DSDiv)** C16C Double-precision signed division. 114

**[DShiftLeft](#DShiftLeft)** C15D Double-precision left shift (zeros shifted in). 115

**[DShiftRight](#DShiftRight)** C262 Double-precision right shift (zeros shifted in). 116

memory

------------------- ---- -------------------------------------------------- ---

[**ClearRam**](#ClearRam)$C178 Clear memory to $00. 118

[**CmpFString**](#CmpFString)$C26E Compare two fixed-length strings. 119

[**CmpString**](#CmpString)$C26B Compare two null-terminated strings. 120

[**CopyFString**](#CopyFString)$C268 Copy a fixed-length string. 121

[**CopyString**](#CopyString)$C265 Copy a null-terminated string. 122

[**DoBOp**](#DoBOp)$C2EC (128) Back-RAM memory primitive 123

[**DoRAMOp**](#DoRamOp) $C2D4 RAM-expansion unit access primitive. 130

[**FetchRAM**](#FetchRam) C2CB Transfer data from RAM-expansion unit. 131

[**FillRam**](#FillRam)$C17B Fill memory with a particular byte. 124

i\_[**FillRam**](#FillRam)$C1B4 Inline **FillRam**. 124

[**InitRam**](#InitRam)$C181 Initialize memory areas from table. 125

[**MoveBData**](#MoveBData)$C2E3128 BackRAM memory move routine. 126

[**MoveData**](#MoveData)$C17E Intelligent memory block move 127

[**i­\_MoveData**](#MoveData)$C1B7 Inline MoveData. 127

[**StashRAM**](#StashRAM) $C2C8 Transfer memory to RAM-expansion unit. 132

[**SwapBData**](#SwapBData)$C2E6 128 memory swap between front/back ram. 128

[**SwapRAM**](#SwapRAM) $C2CE Swap memory with an REU memory block. 133

[**VerifyBData**](#VerifyBData) **$C2E9** 128 BackRAM verify. 129

[**VerifyRAM**](#VerifyRAM) $C2D1 RAM-expansion unit verify. 134

mouse/Sprite

------------------- ---- -------------------------------------------------- ---

ClearMouseMode $C19C Reset the mouse -815

HideOnlyMouse $C2F2 (128) Temporarily remove soft-sprite mouse pointer.

IsMseInRegion $C2B3 Check if mouse is inside a window -710

MouseUp $C18A Turn on the mouse -813

MouseOff $C18D Turn off the mouse -814

**[SetMsePic](#SetMsePic)** C2DA Set and preshift new soft-sprite mouse picture.136

StartMouseMode $C14E Initialize the mouse -812

[**TempHideMouse**](#TempHideMouse) $C2D7 Hide soft-sprites before direct screen access.137

printer driver

------------------- ---- -------------------------------------------- -----

GetDimensions $790C

InitForPrint $7900

PrintASCI $790F

PrintBuffer $7906

SetNLO $7915

**[StartASCII](#StartASCII)** 7912 Begin ASCII mode printing. 138

StartPrint $7903

StopPrint $7909

process

------------------- ---- -------------------------------------------------- ---

[**BlockProcess**](#BlockProcesses) C10C Block process from running. Does not freeze timer. 141

[**EnableProcess**](#EnableProcess) C109 Make a process runnable immediately. 144

[**FreezeProcess**](#FreezeProcess) C112 Pause a process countdown timer. 142

[**InitProcesses**](#InitProcesses) C103 Initialize processes. 143

**[RestartProcess](#RestartProcess)** C106 Unblock, unfreeze, and restart process. 145

**[Sleep](#Sleep)** C199 Put current routine to sleep for a specified time. 146

**[UnblockProcess](#UnblockProcess)** C10F Unblock a blocked process, allowing it to run again. 146

**[UnfreezeProcess](#UnfreezeProcess)** C115 Unpause a frozen process timer 148

text

------------------- ---- -------------------------------------------- -----

[**GetCharWidth**](#GetCharWidth) $C1C9 Calculate width of char without style attributes. 157

[**GetNextChar**](#GetNextChar) $C2A7 Get next character from keyboard queue. 158

[**GetRealSize**](#GetRealSize) $C1B1 Calculate actual character size with attributes. 159

**[GetString](#GetString" \o "GetString)**  C1BA Get string input from user. 160

InitTextPrompt $C1C0 Create the text cursor sprite -208

LoadCharSet $C1CC Load and activate a new font -212

PromptOff $C29E Turn off the text cursor -210

PromptOn $C29B Turn on the text cursor -209

PutChar $C145 Display a character -201

**[PutDecimal](#PutDecimal)** $C184 Format and display an unsigned double-precision nbr. 163

PutString $C148 Display a text string -204

i\_PutString $C1AE Display a text string with inline data -205

UseSystemFont $C14B Select the BSW font -211

[**SmallPutChar**](#SmallPutChar) $C202 Fast character print routine. 162

utility

------------------- ---- -------------------------------------------- -----

**[Bell](#Bell)** N/A Play a bell sound 150

[**[CallRoutine](#CallRoutine)**](#CallRoutine) C1D8 pseudo-subroutine call. $0000 aborts call. 151

**[CRC](#CRC)** C20E Cyclic Redundancy Check calculation. 152

[**[DoInlineReturn](#DoInlineReturn)**](#DoInlineReturn) C2A4 Return from inline subroutine. 153

[[[**[GetRandom](#CallRoutine)**](#GetRandom)](#CallRoutine)](#GetRandom) C187 Calculate new random number. 154

[[[[[[[[[[[[[[[[[[[**[ToBasic](#CallRoutine)**](#ToBasic)](#CallRoutine)](#GetRandom)](#CallRoutine)](#GetRandom)](#CallRoutine)](#GetRandom)](#CallRoutine)](#ToBasic)](#CallRoutine)](#GetRandom)](#CallRoutine)](#GetRandom)](#CallRoutine)](#GetRandom)](#CallRoutine)](#GetRandom)](#CallRoutine)](#GetRandom)  C241 Pass Control to Commodore BASIC. 155

Wheels Kernal

------------------- ---- -------------------------------------------- -----

[**GetNewKernal**](#GetNewKernal) $9D80 Load New Kernal Group

[**RstrKernal**](#RstrKernal) $9D83 Unload Kernal Group

KG­\_REU 0

-------------

GetRAMBam $5000

PutRAMBam $5003

AllocA11RAM $5006

AllocRAMBlock $5009

FreeRAMBlock $500C

GetRAMInfo $500F

RamBlkAlloc $5012

RemoveDrive $5015

SvRamDevice $5018

DelRamDevice $501B

RamDevinfo $501E

KG\_DEVICE 1

-------------

DevNumChange $5000

SwapDrives $5003

KG­\_DISK 2

-------------

NSetGEOSDisk $5000

DBFormat $5003

FormatDisk $5006

DBEraseDisk $5009

EraseDisk $500C

KG­\_READFILE 3

-------------

OReadFile $5000

KG­\_WRITEFILE 4

-------------

OWriteFile $5000

KG­\_DIRECTORY 5

-------------

ChgParType $5000

ChPartition $5003

ChSubdir $5006

ChDiskDirectory $5009

GetFEntries $500C

TopDirectory $500F

UpDirectory $5012

DownDirectory $5015

GoPartition $5018

ChPartOnly $501E

FindRamLink $5027

KG\_MKDIR 6

-------------

MakeDirectory $5000

MakeSysDir $5003

KG­\_VALDISK 7

---------

ValDisk $5000

KG­\_CPYDISK 8

---------

CopyDisk $5000

TestCompatibility $5003

KG­­\_COPY 9

-------------

CopyFile $5000

KG­\_DESKTOP 10

---------

NewDesktop $5000

OEnterDesktop $5003

InstallDriver $5006

FindDesktop $5009

FindAFile $500c

KG\_[[[[[[[[**ToBasic**](#ToBasic)](#CallRoutine)](#GetRandom)](#CallRoutine)](#GetRandom)](#CallRoutine)](#GetRandom)](#CallRoutine) 11

---------

K[[[[[[[[**ToBasic**](#ToBasic)](#CallRoutine)](#GetRandom)](#CallRoutine)](#GetRandom)](#CallRoutine)](#GetRandom)](#CallRoutine) $5000

**Structures**

disk

-------------

[**Directory Entry**](#Directory_Entry)

**Constants**

errors

-------------

examples

disk

-------------

[**CheckDiskSpace**](#exCheckDiskSpace)

# Chapter 1 GEOS Kernal

## dialog box

------------------- ---- -------------------------------------------- -----

[**DoDlgBox**](#DoDlgBox) C256 Display and begin interaction w/dialog box. 14

[**RstrFrmDialog**](#RstrFrmDialog)C2BF Exits from a dialog box. 15

### DoDlgBox: (C64,C128) [**C256**](#hC256)

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

**Parameters**: [**r0**](#zr0)DIALOG —pointer to dialog box definition (word).

**r5-rl0** can be used to send parameters to a dialog box.

When using **DB**[**GetFile**](#GetFile)**S**

**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:** Whenusing **DB**[**GetFile**](#GetFile)**S and bit 7 of r7L is set.**

**r5** FILTERPtr 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**: [**r0**](#zr0)**L** return code: typically the number of the system icon clicked on to exit.

*Note: returns when dialog box exits through* [***RstrFrmDialog***](#RstrFrmDialog)*.*

**Destroys: n/a**

**Description:** [**DoDlgBox**](#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**](#zr0)), 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**](#RstrFrmDialog), passing the number of the system icon selected in **sysDBData**. [**RstrFrmDialog**](#RstrFrmDialog) restores the system state and copies **sysDBData** to [**r0**](#zr0)**L**.

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**](#RstrFrmDialog).

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

**Structure:** [**DIALOG**](#DIALOG)

**Example**:

**See also**: [**RstrFrmDialog**](#RstrFrmDialog)

### **RstrFrmDialog**: (C64,C128) [**C2BF**](#hC2BF)

**Function**: Exits from a dialog box, restoring the system to the state prior to the call to [**DoDlgBox**](#DoDlgBox).

**Parameters**: none.

**Returns**: Returns to point where [**DoDlgBox**](#DoDlgBox)was called. System context is restored. [**r0**](#zr0)**L** contains **sysDBData** return value**.**

**Destroys:** assume a, x, y, **r0H-rl5**

**Description:** [**RstrFrmDialog**](#RstrFrmDialog) allows a custom dialog box routine to exit from the a dialog box. [**RstrFrmDialog**](#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**](#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**](#RstrFrmDialog) restores the sp register to value it contained at the call to [**DoDlgBox**](#DoDlgBox)just before returning. This allows [**RstrFrmDialog**](#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**](#DIALOG)

**Example**

## [**disk very low-level**](#very_low_level_disk)

------------------- ---- -------------------------------------------- -----

[**ChangeDiskDevice**](#ChangeDiskDevice)C2BC Change disk drive device number. 17

[**DoneWithIO**](#DoneWithIO)C25F Restore system after serial I/O. 18

[**EnterTurbo**](#EnterTurbo)C214 Activate disk turbo on current drive. 19

[**ExitTurbo**](#ExitTurbo)C232 Deactivate disk turbo on current drive. 20

[**InitForIO**](#InitForIO)C25C Prepare system for serial I/O. 21

[**PurgeTurbo**](#PurgeTurbo)C235 Remove disk turbo from current drive. 21

[**ReadBlock**](#ReadBlock)C21A Get disk block primitive. 23

[**ReadLink**](#ReadLink) **904B Read track/sector link.** 24

[**VerWriteBlock**](#VerWriteBlock)C223 Disk block verify primitive. 25

[**WriteBlock**](#WriteBlock)C220 Write disk block primitive. 26

### ChangeDiskDevice: (C64, C128) [**C2BC**](#hC2BC)

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

**Parameters**: **a** NEWDEVNUM — new device number to give current drive (byte). [[[[**curDrive**](#d_curDrive)](#curDirHead)](#d_curDevice)](#curDirHead) drive whose device number will change.

**Uses:** [[[[**curDrive**](#d_curDrive)](#curDirHead)](#d_curDevice)](#curDirHead)drive whose device number will change.

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

**Alters:** [[[[**curDrive**](#d_curDrive)](#curDirHead)](#d_curDevice)](#curDirHead)NEWDEVNUM

[[**curDevice**](#d_curDevice)](#curDirHead)NEWDEVNUM

**Destroys**: **a,y**

**Description**:[**ChangeDiskDevice**](#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**](#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**](#ChangeDiskDevice) **is used on a RAMdisk,** [[[[**curDrive**](#d_curDrive)](#curDirHead)](#d_curDevice)](#curDirHead) **and** [[**curDevice**](#d_curDevice)](#curDirHead) **both**

**change. However, because of the nature of the RAMdisk driver, the RAMdisk does not respond as this new device.**

**Example:**

**See also**: [**SetDevice**](#SetDevice)

### DoneWithIO: (C64, C128) [**C25F**](#hC25F)

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

**Parameters**: none.

**Returns**: nothing.

**Destroys**: a,y

**Description**:[**DoneWithIO**](#DoneWithIO) **restores the state of the system after a call to** [**InitForIO**](#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**](#InitForIO)and [**DoneWithIO**](#DoneWithIO).

**Example:** [**MyPutBlock**](#exMyPutBlock)

**See also**: [**InitForIO**](#InitForIO)

### EnterTurbo: (C64, C128) [**C214**](#hC214)

**Function**: Activate disk drive turbo mode

**Parameters**: none.

**Uses:** [[[[**curDrive**](#d_curDrive)](#curDirHead)](#d_curDevice)](#curDirHead)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**](#EnterTurbo) **activates the turbo software in the current drive. If the turbo software has not yet been downloaded to the drive,** [**EnterTurbo**](#EnterTurbo) **will download it. The turbo software allows GEOS to perform high-speed serial disk access.**

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

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

**Example**: [**MyPutBlock**](#exMyPutBlock)

**See also**: [**WriteBlock**](#WriteBlock)**,** [**ExitTurbo**](#ExitTurbo)**,** [**PurgeTurbo**](#PurgeTurbo)**.**

### **ExitTurbo**: (C64, C128) [C232](#hC232)

**Function**: Deactivate disk drive turbo mode.

**Parameters**: none.

**Uses:** [[[[**curDrive**](#d_curDrive)](#curDirHead)](#d_curDevice)](#curDirHead)currently active disk drive.

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

**Destroys**: **a,y**

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

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

**Example:**

**See also**: [**EnterTurbo**](#EnterTurbo), [**PurgeTurbo**](#PurgeTurbo).

### InitForIO: (C64, C128) [****C25C****](#hC25C)

**Function**: Prepare for I/O across the serial bus

**Parameters**: none.

**Returns**: nothing.

**Destroys**: a,y

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

**Example:** [**MyPutBlock**](#exMyPutBlock)

**See also**: [**DoneWithIO**](#DoneWithIO)**,** [**SetDevice**](#SetDevice)

### **PurgeTurbo**: (C64, C128) [****C235****](#hC235)

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

**Parameters: none**

**Uses:** [[[[**curDrive**](#d_curDrive)](#curDirHead)](#d_curDevice)](#curDirHead)currently active disk drive.

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

**Destroys**: **a,y**

**Description**:[**PurgeTurbo**](#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**](#EnterTurbo), [**ExitTurbo**](#ExitTurbo).

### ReadBlock: (C64, C128) [****C12A****](#hC12A)

**Function:** Very low-level read block from disk.

**Parameters: rlL TRACK—valid track number (byte),**

**rlH SECTOR—valid sector on track (byte).**

**r4 BUFFER — address of buffer of BLOCKSIZE bytes to read block into (word).**

**Uses:** [[[[**curDrive**](#d_curDrive)](#curDirHead)](#d_curDevice)](#curDirHead)currently active disk drive.

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

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

**Destroys**: **a,y**

**Description**:[**ReadBlock**](#ReadBlock) **reads the block at the specified TRACK and SECTOR into BUFFER. If the disk is shadowed,** [**ReadBlock**](#ReadBlock) **will read from the shadow memory.** [**ReadBlock**](#ReadBlock) **is a pared down version of GetBlock. It expects the application to have already called** [**EnterTurbo**](#EnterTurbo) **and** [**InitForIO**](#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**](#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**](#ReadBlock) **can function as the foundation of specialized, high-speed disk routines.**

**Example: MyGetBlock**

**See also**: [**GetBlock**](#GetBlock), [**WriteBlock**](#WriteBlock), [**VerWriteBlock**](#VerWriteBlock).

### ReadLink: (C64, C128) [****904B****](#h904B)

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

**Parameters: rlL TRACK** — track number (byte),

**rlH SECTOR** — sector on track (byte).

**r4 BUFFER** —address of buffer of at least **BLOCKSIZE** bytes, usually points to [**diskBlkBuf**](#d_diskBlkBuf) (word).

**Uses:** [[[[**curDrive**](#d_curDrive)](#curDirHead)](#d_curDevice)](#curDirHead)currently active disk drive.

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

**Destroys**: **a,y**

**Description**:[**ReadLink**](#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**](#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**](#DeleteFile)](#DeleteFile) **and** [[[[**FollowChain**](#FollowChain)](#FastDelFile)](#FindBAMBit)](#FastDelFile) **will automatically take advantage of this capability of 1581 drives.**

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

**Note:** Does not work in 1541 Drivers. Use [**ReadBlock**](#ReadBlock)instead.

**Example:**

**See also**: [**ReadBlock**](#ReadBlock), **FollowChain**

### VerWriteBlock: (C64, C128) ****[C223](#hC223)****

**Function:** Very low-level verify block on disk.

**Parameters: rlL** TRACK — track number (byte).

**rlH** 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**](#d_curDrive)](#curDirHead)](#d_curDevice)](#curDirHead)currently active disk drive.

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

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

**Destroys**: **a,y**

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

[**VerWriteBlock**](#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**](#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**](#VerWriteBlock) **can function as the foundation of specialized, high-speed disk routines.**

[**VerWriteBlock**](#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: My**[**PutBlock**](#exMyPutBlock)

**See also**: [**WriteBlock**](#WriteBlock)**,** [**PutBlock**](#PutBlock)

### WriteBlock: (C64, C128) [****C220****](#hC220)

**Function:** Very low-level write block to disk.

**Parameters: rlL** TRACK — valid track number (byte).

**rlH** SECTOR—valid sector on track (byte).

**r4** BUFFER — address of buffer of BLOCKSIZE bytes that contains data to write out (word).

**Uses:** [[[[**curDrive**](#d_curDrive)](#curDirHead)](#d_curDevice)](#curDirHead)currently active disk drive.

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

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

**Destroys**: **a,y**

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

[**WriteBlock**](#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**](#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](#WriteBlock) can function as the foundation of specialized, high-speed disk routines.**

**Example: My**[**PutBlock**](#exMyPutBlock)

**See also**: [**PutBlock**](#PutBlock), [**ReadBlock**](#ReadBlock), [**VerWriteBlock**](#VerWriteBlock).

## disk low-level

------------------- ---- -------------------------------------------- -----

[**GetBlock**](#GetBlock) C1E4 Read single disk block into memory. 28

[**PutBlock**](#PutBlock) C1E7 Write single disk block from memory. 29

### GetBlock: (C64, C128) [****C1E4****](#hC1E4)

**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).

**rlL** TRACK — track number (byte).

**r1H** SECTOR — sector number on track (byte).

**Uses:** [[[[**curDrive**](#d_curDrive)](#curDirHead)](#d_curDevice)](#curDirHead)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**

**(1581 drive, r1, r4)**

AAnote: *Need to confirm is this is still true*

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

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

The Commodore 1581 driver has a bug that causes its [**GetBlock**](#GetBlock)to trash rl and **r4**.

**Example:**

**See also**: [**PutBlock**](#PutBlock)**,** [**WriteBlock**](#WriteBlock)**,** [**BlkAllOC**](#BlkAllOC)**.**

### PutBlock: (C64, C128) [****C1E7****](#hC1E7)

**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**](#d_curDrive)](#curDirHead)](#d_curDevice)](#curDirHead)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**](#PutBlock) **writes a block from BUFFER to the disk.** [**PutBlock**](#PutBlock) **is useful for implementing disk utility programs and new file structures.**

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

**Note3:** [**PutBlock**](#PutBlock) **does no boundary check on the buffer. If the buffer is less the then BLOCKSIZE ($100) bytes,** [**PutBlock**](#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**](#GetBlock)**,** [**WriteBlock**](#WriteBlock)**,** [**BlkAllOC**](#BlkAllOC)**.**

## disk mid-level

------------------- ---- -------------------------------------------- -----

[[**AllocateBlock**](#AllocateBlock)](#AllocateBlock) 9048 Mark a disk block as in-use.31

[**BldGDirEntry**](#BldGDirEntry) C1F3 Create a directory entry in memory.32

[**BlkAllOC**](#BlkAllOC) C1FC Allocate sectors for a file.33

[[**CalcBlksFree**](#CalcBlksFree)](#CalcBlksFree) C1DB Compute number of free blocks on disk34

[**ChkDkGEOS**](#ChkDkGEOS) C1DE Check if a disk is GEOS format.35

[**FastDelFile**](#FastDelFile) $C244 Delete a temporary file. -407

[[**FindBAMBit**](#FindBAMBit)](#FastDelFile) $C2AD Check if a disk sector is in use. -316

[[[[**FollowChain**](#FollowChain)](#FastDelFile)](#FindBAMBit)](#FastDelFile) $C205 Create a list of sectors used by a file. -420

[**FreeBlock**](#FreeBlock) $C2B9 Free a BAM bit in memory.

[**FreeFile**](#FreeFile) $C226 Free a file's sectors. -408

**GetlstDirEntry** $9030 Loads in the first directory block.

GetNxtDirEntry 9033 Get directory entry other than first.

[**GetDirHead**](#GetDirHead) $C247 Read track 18 sector 0. -308

[[[**GetFHdrInfo**](#GetFHdrInfo)](#GetFHdrlnfo)](#GetFHdrlnfo) $C229 Get a file's load address. -419

[[[[**GetFreeDirBlk**](#GetFreeDirBlk)](#GetFHdrlnfo)](#GetFHdrInfo)](#GetFHdrlnfo) $C1F6 Find a hole in the directory. -416

GetOffPageTrSc 9036 Get track and sector of off-page directory.

[**LdApplic**](#LdApplic) $C21D Load and run a file, given a directory entry. -412

[**LdDeskAcc**](#LdDeskAcc) $C217 Load a file with memory swapping. -413

[[**LdFile**](#LdFile)](#LdDeskAcc) $C211 Load a file, given a directory entry. -411

[**NewDisk**](#NewDisk) C1E1 Initialize a drive.

NxtBlkAlloc $C24D Allocate sectors for a file. -319

PutDirHead $C24A Write to track 18 sector 0. -309

[**ReadByte**](#ReadByte) $C2B6 Read a File 1 byte at a time.

ReadFile $C1FF Load a chain of blocks into memory. -415

SetGDirEntry $C1F0 Create a directory entry on disk. -417

SetNextFree $C292 Find and allocate a disk block. -317

StartAppl $C22F Run a program that is in memory. -807

WriteFile $C1F9 Save memory to preallocated sectors. -404

### **[AllocateBlock](#AllocateBlock)**: (C64, C128) [****9048****](#h9048)

**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**](#d_curDrive)](#curDirHead)](#d_curDevice)](#curDirHead)drive that disk is in.

[**curDirHead**](#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**](#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**](#AllocateBlock)](#AllocateBlock) **allocates a single block on this disk by setting the appropriate flag in the block allocation map** (**BAM).**

**See also**: **SetNextFree**, [**BlkAllOC**](#BlkAllOC), **FreeBlock**.

**If the sector is already allocated then a BAD\_BAM error is returned.** [**AllocateBlock**](#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**](#AllocateBlock)**. All other device drivers, however, do. The following subroutine will properly allocate a block on any device, including the 1541.**

[**NewAllocateBlock**](#exNewAllocateBlock)

**Example:** [[**CallNewAlloc**](#exCallNewAlloc)](#AllocateBlock)

### **BldGDirEntry**: (C64, C128) [****C1F3****](#hC1F3)

**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**](#BlkAllOC)](#BlkAllOC)can be used to build such a list (word).

**r9** FILEHDR — pointer to GEOS file header (word).

**Uses:** [[[[**curDrive**](#d_curDrive)](#curDirHead)](#d_curDevice)](#curDirHead)drive that disk is in.

**Returns**: r6 pointer to first non-reserved block in track/sector table ([**BldGDirEntry**](#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**](#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). [**SaveFile**](#SaveFile)calls **SetGDirEntry**, which calls [**BldGDirEntry**](#BldGDirEntry)as part of its normal processing.

**See also**: **SetGDirEntry**

**Example:** [**MySetGDirEntry**](#exMySetGDirEntry)

### **BlkAllOC**: (C64, C128) [****C1FC****](#hC1FC)

**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**](#d_curDrive)](#curDirHead)](#d_curDevice)](#curDirHead)drive that disk is in.

[**curDirHead**](#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**:

**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**](#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](#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**](#CalcBlksFree) **to ensure that enough free blocks exist on the disk. If there are not enough free blocks to accommodate the data,** [**BlkAllOC**](#BlkAllOC) **returns an INSUFFICIENT\_SPACE error without allocating any blocks. Otherwise,** [**BlkAllOC**](#BlkAllOC) **calls SetNextFree to allocate the proper number of unused blocks.**

[**BlkAllOC**](#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**](#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.**

**See also**: **NxtBlkAlloc, SetNextFree,** [[[[**GetFreeDirBlk**](#GetFreeDirBlk)](#GetFHdrlnfo)](#GetFHdrInfo)](#GetFHdrlnfo)**, FreeBlock.**

**Example:** [**GrabSomeBlocks**](#exGrabSomeBlocks)

### **[CalcBlksFree](#CalcBlksFree)**: (C64, C128) [****C1DB****](#hC1DB)

**Function**: Calculate total number of free blocks on disk.

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

**Uses:** [[[[**curDrive**](#d_curDrive)](#curDirHead)](#d_curDevice)](#curDirHead)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**](#CalcBlksFree)](#CalcBlksFree) **calculates the number of free blocks available on the disk. An application can call** [[**CalcBlksFree**](#CalcBlksFree)](#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**](#BlkAllOC)**, call** [[**CalcBlksFree**](#CalcBlksFree)](#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**](#CalcBlksFree)](#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**](#curDirHead). Calling [[**CalcBlksFree**](#CalcBlksFree)](#CalcBlksFree)requires first loading **r5** with the address of [**curDirHead**](#curDirHead).

**LoadW** **r5**, #[**curDirHead**](#curDirHead)

jsr [**CalcBlksFree**](#CalcBlksFree)

When checking the total number of blocks (both allocated and free) on a particular disk device, call [[**CalcBlksFree**](#CalcBlksFree)](#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**](#curDirHead) ; point to the directory header

jsr [**CalcBlksFree**](#CalcBlksFree) ; r3 comes back with total number of blocks

; on this device

**See also**: **NxtBlkAlloc, SetNextFree,** [[[[**GetFreeDirBlk**](#GetFreeDirBlk)](#GetFHdrlnfo)](#GetFHdrInfo)](#GetFHdrlnfo)**, FreeBlock.**

**Example:** [**CheckDiskSpace**](#exCheckDiskSpace)

### **ChkDkGEOS**: (C64, C128) [****C1DE****](#hC1DE)

**Function**: Check Commodore disk for GEOS format.

**Parameters**: **r5** DIRHEAD — address of directory header, should always point to [**curDirHead**](#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**](#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**](#OpenDisk) **automatically calls** [**ChkDkGEOS**](#ChkDkGEOS)**. As long as OpenDisk is used before reading a new disk, applications should have no need to call** [**ChkDkGEOS**](#ChkDkGEOS)

**See also**: **SetGEOSDisk**

**Example:**

jsr [**GetDirHead**](#GetDirHead) ; read in the directory header

txa ; check status

bne 99$ ; exit on error

LoadW r5,#[**curDirHead**](#curDirHead) ; point to directory header

jsr [**ChkDkGEOS**](#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

### ****[FastDelFile](#FastDelFile)****: (C64, C128) [****C244****](#hC244)

**Function**: Special Commodore version of [**DeleteFile**](#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**](#d_curDrive)](#curDirHead)](#d_curDevice)](#curDirHead)

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

[**curDirHead**](#curDirHead)BAM updated to reflect newly freed blocks.

**dir2Head** (BAM for 1571 and 1581 drives only)

**dir3Head** (BAM for 1581 drive only)

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

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

**Description**:[[**FastDelFile**](#FastDelFile)](#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**](#FastDelFile)](#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**](#FastDelFile)](#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**](#DeleteFile) **or** [**FreeFile**](#FreeFile) **for deleting VLIR files or DeleteRecord for deleting a single record.**

[[**FastDelFile**](#FastDelFile)](#FastDelFile) **calls** [**GetDirHead**](#GetDirHead) **before freeing any blocks. This will overwrite any BAM and directory header in memory.**

**Note:** [[**FastDelFile**](#FastDelFile)](#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**](#exDeleteDirEntry)

**See also**: [**FreeFile**](#FreeFile), [**DeleteFile**](#DeleteFile)

**Examples**: **[DeleteDirEntry](#exDeleteDirEntry),** [**ReadAndDelete**](#exReadAndDelete)

### [****[FindBAMBit](#FastDelFile)****](#FindBAMBit): (C64, C128) [****C2AD****](#hC2AD)

**Function**: Get disk block allocation status.

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

**r6H** SECTOR — sector number of block (byte).

**Uses:** [[[[**curDrive**](#d_curDrive)](#curDirHead)](#d_curDevice)](#curDirHead)

[**curDirHead**](#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**](#curDirHead) for BAM byte.

**r8H** mask for isolating BAM bit.

a BAM byte masked with r8H.

**r7H** offset from [**curDirHead**](#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**](#FindBAMBit)](#FastDelFile)](#FindBAMBit)](#FastDelFile) **accesses the BAM of the current disk "in** [**curDirHead**](#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**](#FindBAMBit)](#FastDelFile)](#FindBAMBit)](#FastDelFile) **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**](#FindBAMBit)](#FastDelFile)](#FindBAMBit)](#FastDelFile)**.**

**bne BlockIsFree ;branch if block is free**

* **or -**

**beq BlockInUse ;branch if block is in-use**

**Note:** [[[[**FindBAMBit**](#FindBAMBit)](#FastDelFile)](#FindBAMBit)](#FastDelFile) **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**](#AllocateBlock)](#AllocateBlock)**.**

**See also**: [[**AllocateBlock**](#AllocateBlock)](#AllocateBlock)**, FreeBlock,** [**GetDirHead**](#GetDirHead) **, PutDirHead**

**Examples**:

**LoadB** **r6L**,#TRACK ; get track and sector number

**LoadB r6H,#SECTOR**

**jsr** [[[[**FindBAMBit**](#FindBAMBit)](#FastDelFile)](#FindBAMBit)](#FastDelFile) **; get allocation status**

**beq BlocklnUse ; branch if already in use**

### [[[****[FollowChain](#FastDelFile)****](#FollowChain)](#FastDelFile)](#FindBAMBit): (C64, C128) [****C205****](#hC205)

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

**Parameters**: **rlL** START\_TRACK — track number of starting block (byte).

**rlH** START\_SEC — sector number of starting block (byte).

**r3** TSTABLE — pointer to buffer for building out track and sectortable of chain, usually points to fileTrScTab (word).

**Uses:** [[[[**curDrive**](#d_curDrive)](#curDirHead)](#d_curDevice)](#curDirHead)

**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**](#d_diskBlkBuf) **used for temporary block storage.**

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

**Description**:[[[[[[[[**FollowChain**](#FollowChain)](#FastDelFile)](#FindBAMBit)](#FastDelFile)](#FollowChain)](#FastDelFile)](#FindBAMBit)](#FastDelFile) **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**](#FollowChain)](#FastDelFile)](#FindBAMBit)](#FastDelFile) **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**](#FollowChain)](#FastDelFile)](#FindBAMBit)](#FastDelFile) **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**](#FollowChain)](#FastDelFile)](#FindBAMBit)](#FastDelFile) **builds a standard track/sector table compatible with routines such as WriteFile and** [[**FastDelFile**](#FastDelFile)](#FastDelFile)**.**

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

**Examples**:

**LoadB** **r1L**,#START\_TR ; start track

**LoadB r6H,#**START\_**SC ; and sector**

**LoadW r3,#fileTricTab ; buffer for table**

**jsr** [[[[[[[[**FollowChain**](#FollowChain)](#FastDelFile)](#FindBAMBit)](#FastDelFile)](#FollowChain)](#FastDelFile)](#FindBAMBit)](#FastDelFile) **; get allocation status**

**txa ; set status flags**

**bne ; branch if error**

### FreeBlock: (C64, C128) [****C2B9****](#hC2B9)

**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**](#d_curDrive)](#curDirHead)](#d_curDevice)](#curDirHead)

[**curDirHead**](#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**](#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**](#FreeBlock) **tries to free (deallocate) the block number passed in r6. If the block is already free, then** [**FreeBlock**](#FreeBlock) **returns a BAD­\_BAM error.**

**Note**: [**FreeBlock**](#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 My**[**FreeBlock**](#MyFreeBlock)

**See also**: [[**FreeFile**](#FreeFile)**,**](#SetDevice) **AllocateBlock**

**Example: My**[**FreeBlock**](#MyFreeBlock)

### ****[FreeFile](#FreeFile)****: (C64, C128) [****C226****](#hC226)

**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**](#d_curDrive)](#curDirHead)](#d_curDevice)](#curDirHead)

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

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

**r6L, r6H** unchanged.

**Alters:** [**diskBlkBuf**](#d_diskBlkBuf)used for temporary block storage.

[**curDirHead**](#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**](#FreeFile)](#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**](#FindFile), [**GetlstDirEntry**](#GetlstDirEntry)and [**GetNxtDirEntry**](#GetNxtDirEntry). [[**FreeFile**](#FreeFile)](#FreeFile)is called automatically by **DeleteFile**.

[[**FreeFile**](#FreeFile)](#FreeFile) **tries to free (deallocate) the block number passed in r6. If the block is already free, then** [**FreeBlock**](#FreeBlock) **returns a BAD­\_BAM error.**

[[**FreeFile**](#FreeFile)](#FreeFile)calls [**GetDirHead**](#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**](#FreeFile)](#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**](#FreeFile)](#FreeFile)then calls **PutDirHead** to write out the new BAM.

When using [**GetlstDirEntry**](#GetlstDirEntry)and [**GetNxtDirEntry**](#GetNxtDirEntry), do not pass [[**FreeFile**](#FreeFile)](#FreeFile)a

pointer into [**diskBlkBuf**](#d_diskBlkBuf). Copy the full directory entry (**DIRENTRY\_SIZE**

bytes) from [**diskBlkBuf**](#d_diskBlkBuf) to another buffer (such as **dirEntryBuf**) and pass

[[**FreeFile**](#FreeFile)](#FreeFile)the pointer to that buffer. Otherwise when [[**FreeFile**](#FreeFile)](#FreeFile)uses

[**diskBlkBuf**](#d_diskBlkBuf) it will corrupt the directory entry.

**See also**: [[**DeleteFile**](#DeleteFile)](#DeleteFile), **FreeDir**, [**FreeBlock**](#FreeBlock)**.**

Because [[**FreeFile**](#FreeFile)](#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.

### GetlstDirEntry: (C64, C128) [****9030****](#h9030)

**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**](#d_curDrive)](#curDirHead)](#d_curDevice)](#curDirHead)

**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**](#d_diskBlkBuf).

**Alters:** [**diskBlkBuf**](#d_diskBlkBuf)directory block.

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

**Description**: **[GetlstDirEntry](#GetlstDirEntry) reads in the first directory block of the current directory and returns with r5 pointing to the first directory entry.** [**GetlstDirEntry**](#GetlstDirEntry) **is called by routines like** [[**FindFTypes**](#FindFTypes)](#FindFile) **and** [**FindFile**](#FindFile)**.**

To get a pointer to subsequent directory entries, call [**GetNxtDirEntry**](#GetNxtDirEntry).

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

[**GetlstDirEntry**](#GetlstDirEntry)did not appear in the jump table until version 1.3. An

application running under version 1.2 can access [**GetlstDirEntry**](#GetlstDirEntry)by calling directly into the Kernal. The following subroutine will work on Commodore GEOS vl.2 and later:

;\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

; MyGetlstDirEntry — Call instead of GetlstDirEntry

; to work on GEOS vl.2 and later

;\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

;EQUATE: vl.2 entry point directly into Kernal. Must

;do a version check before calling.

o\_GetlstDirEntry = $c9f7 ; exact entry point

MyGetlstDirEntry**:**

lda **version** ; check version number

cmp #$13

bcc 10$  **;** branch < vl.3

jmp **GetlstDirEntry ;** direct call

10$

jmpo\_GetlstDirEntry **;** go through jump table

**See also**: **GetNxtDirEntry**, [[**FindFTypes**](#FindFTypes)](#FindFile).

**Example:**

### GetNxtDirEntry: (C64, C128) [****9033****](#h9033)

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

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

**Uses:** [[[[**curDrive**](#d_curDrive)](#curDirHead)](#d_curDevice)](#curDirHead)

[**diskBlkBuf**](#d_diskBlkBuf)must be unaltered from previous call to [**GetlstDirEntry**](#GetlstDirEntry)or [**GetNxtDirEntry**](#GetNxtDirEntry).

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

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

**r5** pointer to next directory entry within [**diskBlkBuf**](#d_diskBlkBuf).

ynon-zero if end of directory reached

**Alters:** [**diskBlkBuf**](#d_diskBlkBuf)directory block.

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

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

[**GetNxtDirEntry**](#GetNxtDirEntry)did not appear in the jump table until version 1.3. An

application running under version 1.2 can access [**GetNxtDirEntry**](#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**](#GetNxtDirEntry) **;** direct call

10$

jmpo\_GetNxtDirEntry **;** go through jump table

**See also**: [**GetlstDirEntry**](#GetlstDirEntry), [[**FindFTypes**](#FindFTypes)](#FindFile).

**Example:**

### ****[GetDirHead](#GetDirHead)**** : (C64, C128) [****C247****](#hC247)

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

**Parameters**: none.

**Uses:** [[[[**curDrive**](#d_curDrive)](#curDirHead)](#d_curDevice)](#curDirHead)

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

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

**r4** pointer to [**curDirHead**](#curDirHead).

**Alters:** [**curDirHead**](#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**](#GetDirHead)](#GetDirHead)**reads the full directory header (256 bytes) into the buffer at** [**curDirHead**](#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**](#GetDirHead)](#GetDirHead)calls [**GetBlock**](#GetBlock)to read in the directory header block into the buffer at [**curDirHead**](#curDirHead). The directory header block contains the directory header and the disk BAM (block allocation map). Typically, applications don't call [[**GetDirHead**](#GetDirHead)](#GetDirHead)because the most up-to-date directory header is almost always in memory (at [**curDirHead**](#curDirHead)), [**OpenDisk**](#OpenDisk)calls [[**GetDirHead**](#GetDirHead)](#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**](#GetDirHead)](#GetDirHead)is called by routines such as [**OpenDisk**](#OpenDisk), [**SetGEOSDisk**](#SetGEOSDisk), and [**OpenRecordFile**](#OpenRecord), etc.

**See also**: **PutDirHead**

**Example:**

### [[****[GetFHdrInfo](#GetFHdrlnfo)****](#GetFHdrInfo)](#GetFHdrlnfo): (C64, C128) [****C229****](#hC229)

**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**](#d_curDrive)](#curDirHead)](#d_curDevice)](#curDirHead)

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

**Returns**: xerror ($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**](#GetFHdrInfo)](#GetFHdrlnfo)](#GetFHdrInfo)](#GetFHdrlnfo)](#GetDirHead)**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**](#FindFile), [**GetlstDirEntry**](#GetlstDirEntry)and [**GetNxtDirEntry**](#GetNxtDirEntry). [[[[[**GetFHdrInfo**](#GetFHdrInfo)](#GetFHdrlnfo)](#GetFHdrlnfo)](#GetFHdrlnfo)](#GetDirHead)is called by routines such as [[[[**LdFile**](#LdFile)](#LdDeskAcc)](#LdFile)](#LdDeskAcc) just prior to calling **ReadFile** (to load in a sequential file or record zero of a VLIR).

[[[[[**GetFHdrInfo**](#GetFHdrInfo)](#GetFHdrlnfo)](#GetFHdrlnfo)](#GetFHdrlnfo)](#GetDirHead)gets the block number (Commodore track/sector) of the GEOS

file header by looking at the **OFF\_GHDR\_PTR** word in the directory entry.

**See also**:

**Example:**

### [[[****[GetFreeDirBlk](#GetFHdrlnfo)****](#GetFreeDirBlk)](#GetFHdrlnfo)](#GetFHdrInfo): (C64, C128) [****C1F6****](#hC1F6)

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

**Parameters**: **rl0L** 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**](#d_curDrive)](#curDirHead)](#d_curDevice)](#curDirHead)

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

[**curDirHead**](#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**: xerror ($00 = no error).

**FULL\_DIRECTORY**

**rl0L** page number of empty directory slot**.**

**rl** block (track/sector) number of directory block in[**diskBlkBuf**](#d_diskBlkBuf)**.**

**y** index to empty directory slot in[**diskBlkBuf**](#d_diskBlkBuf)**.**

**Alters:** [**curDirHead**](#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**](#GetFreeDirBlk)](#GetFHdrlnfo)](#GetFHdrInfo)](#GetFHdrlnfo)](#GetFreeDirBlk)](#GetFHdrlnfo)](#GetFHdrInfo)](#GetFHdrlnfo)](#GetDirHead)**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**](#GetFreeDirBlk)](#GetFHdrlnfo)](#GetFHdrInfo)](#GetFHdrlnfo)](#GetFreeDirBlk)](#GetFHdrlnfo)](#GetFHdrInfo)](#GetFHdrlnfo)](#GetDirHead)**starts searching for an empty slot beginning with page number DIRPAGE. If** [[[[[[[[[**GetFreeDirBlk**](#GetFreeDirBlk)](#GetFHdrlnfo)](#GetFHdrInfo)](#GetFHdrlnfo)](#GetFreeDirBlk)](#GetFHdrlnfo)](#GetFHdrInfo)](#GetFHdrlnfo)](#GetDirHead)**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**](#GetFreeDirBlk)](#GetFHdrlnfo)](#GetFHdrInfo)](#GetFHdrlnfo)](#GetFreeDirBlk)](#GetFHdrlnfo)](#GetFHdrInfo)](#GetFHdrlnfo)](#GetDirHead)**won't find empty directory slots on lower pages and extra directory blocks may be allocated needlessly.**

[[[[[[[[[**GetFreeDirBlk**](#GetFreeDirBlk)](#GetFHdrlnfo)](#GetFHdrInfo)](#GetFHdrlnfo)](#GetFreeDirBlk)](#GetFHdrlnfo)](#GetFHdrInfo)](#GetFHdrlnfo)](#GetDirHead)**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**](#GetFreeDirBlk)](#GetFHdrlnfo)](#GetFHdrInfo)](#GetFHdrlnfo) will return a **FULL\_DIRECTORY** error.

[[[[[[[[[**GetFreeDirBlk**](#GetFreeDirBlk)](#GetFHdrlnfo)](#GetFHdrInfo)](#GetFHdrlnfo)](#GetFreeDirBlk)](#GetFHdrlnfo)](#GetFHdrInfo)](#GetFHdrlnfo)](#GetDirHead)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**](#GetFreeDirBlk)](#GetFHdrlnfo)](#GetFHdrInfo)](#GetFHdrlnfo)](#GetFreeDirBlk)](#GetFHdrlnfo)](#GetFHdrInfo)](#GetFHdrlnfo)](#GetDirHead)does not automatically write out the **BAM**. See **PutDirHead**

for more information on writing out the **BAM**.

**See also**: [[**AllocateBlock**](#AllocateBlock)](#AllocateBlock)**,** [**FreeBlock**](#FreeBlock)**,** [**BlkAllOC**](#BlkAllOC)

**Example:** [**MySetGDirEntry**](#exMySetGDirEntry)

### **GetOffPageTrSc**: (C64, C128) [****9036****](#h9036)

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

**Parameters**: none.

**Uses:** [[[[**curDrive**](#d_curDrive)](#curDirHead)](#d_curDevice)](#curDirHead)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.

**rlL** track of off-page directory.

**r1H** sector of off-page directory.

**r4** pointer to [**curDirHead**](#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**](#GetOffPageTrSc) **reads the directory header into the buffer at** [**curDirHead**](#curDirHead) **and calls** [**ChkDkGEOS**](#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**](#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.**

**See also**: **PutDirHead**

**Example:**

; Put off-page block into [**diskBlkBuf**](#d_diskBlkBuf)

jsr [**GetOffPageTrSc**](#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**](#d_diskBlkBuf) ; get off-page block

jsr ; return with error in x

99$ rts

### ****[LdApplic](#LdApplic)****: (C64, C128) [****C21D****](#hC21D)

**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 O: 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**](#d_curDrive)](#curDirHead)](#d_curDevice)](#curDirHead)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**](#LdApplic).

**r2** as originally passed to [**LdApplic**](#LdApplic) (use **dataDiskName**).

**r3** as originally passed to [**LdApplic**](#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**](#LdApplic)](#LdApplic) **is a mid-level application loading routine called by the higher level** [**GetFile**](#GetFile)**. Given a directory entry of a GEOS application file,** [**LdApplic**](#LdApplic) **will attempt load it into memory and optionally run it.** [**LdApplic**](#LdApplic) **calls** [[**LdFile**](#LdFile)](#LdDeskAcc) **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**](#LdApplic) **directly but will go indirectly through** [**GetFile**](#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**](#LdApplic)](#LdApplic)returns to the caller, it does so before calling **StartAppl** to warmstart GEOS.

**See also**: [**GetFile**](#GetFile)**,** [[**LdDeskAcc**](#LdDeskAcc)](#LdDeskAcc)**, StartAppl**

**Example:**

### ****[LdDeskAcc](#LdDeskAcc)****: (C64, C128) [****C217****](#hC217)

**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**](#d_curDrive)](#curDirHead)](#d_curDevice)](#curDirHead)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:

**rl0L** as originally passed to [[**LdDeskAcc**](#LdDeskAcc)](#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**](#LdDeskAcc)](#LdDeskAcc) **is a mid-level desk accessory loading routine called by the higher level** [**GetFile**](#GetFile)**. Given a directory entry of a GEOS desk accessory file,** [**LdDeskAcc**](#LdDeskAcc) **will attempt load it into memory and run it. When the user closes the desk accessory, control returns to the calling application.**

[**LdDeskAcc**](#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**](#LdDeskAcc) **directly, but will go indirectly through** [**GetFile**](#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:

**See also**: [**GetFile**](#GetFile)**,** [[**LdApplic**](#LdApplic)](#LdApplic)**, RstrAppl,** [**RstrFrmDialog**](#RstrFrmDialog)**.**

bit 7: 1 force desk accessory to save foreground screen area and restore it on return to application.

1. 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.

1. 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:**

### [[[****[LdFile](#LdDeskAcc)****](#LdFile)](#LdDeskAcc)](#LdFile): (C64, C128) [****C211****](#hC211)

**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**](#d_curDrive)](#curDirHead)](#d_curDevice)](#curDirHead)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**+O 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**](#LdFile)](#LdDeskAcc) is in an Unusable state already so this is to be expected.

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

**Most applications will not call** [[[[[[**LdFile**](#LdFile)](#LdDeskAcc)](#LdFile)](#LdDeskAcc)](#LdFile)](#LdDeskAcc) **directly, but will go indirectly through** [**GetFile**](#GetFile)**.**

All versions of [[**LdFile**](#LdFile)](#LdDeskAcc) 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**](#GetFile) to achieve the same effect.

**See also**: [**GetFile**](#GetFile)**,** [[**LdApplic**](#LdApplic)](#LdApplic)**,** [[**LdDeskAcc**](#LdDeskAcc)](#LdDeskAcc)**.**

### **NewDisk**: (C64, C128) [****C1E1****](#hC1E1)

**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**](#d_curDrive)](#curDirHead)](#d_curDevice)](#curDirHead)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**](#zr0)**-r3**

**Description**:[**NewDisk**](#NewDisk) **informs the disk drive turbo software that a new disk has been**

**inserted into the drive. It first calls** [**EnterTurbo**](#EnterTurbo) **then sends an initialize command to the turbo code. If the disk is shadowed, the shadow memory is also cleared.**

[**NewDisk**](#NewDisk) **gets called automatically when** [**OpenDisk**](#OpenDisk) **opens a new disk. An**

**application that does not deal with anything but the low-level disk routines might want to call** [**NewDisk**](#NewDisk) **instead of** [**OpenDisk**](#OpenDisk) **to avoid the unnecessary overhead associated with reading the directory header and initializing internal file-level variables.**

**Note**: [**NewDisk**](#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**](#NewDisk) **during the application's initialization will stop the disk.**

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

**Calls**:2 [**EnterTurbo**](#EnterTurbo), [**InitForIO**](#InitForIO), [**DoneWithIO**](#DoneWithIO)

**See also**: [[**OpenDisk**](#OpenDisk)**,**](#SetDevice)[**SetDevice**](#SetDevice)

**Example:**

### **ReadByte**: (C64, C128) [****C2B6****](#hC2B6)

**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**](#ReadByte), usually a pointer to [**diskBlkBuf**](#d_diskBlkBuf) (word).

**r5** $0000 (word).

**Uses:** [[[[**curDrive**](#d_curDrive)](#curDirHead)](#d_curDevice)](#curDirHead)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**](#ReadByte).

**Destroys**: **y**

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

[**ReadByte**](#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**](#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**](#ReadByte)is especially useful for displaying very large bitmaps with **BitOtherClip**

**Note**: Reading a chain a byte at a time under Commodore GEOS 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.

**See also**: [[**OpenDisk**](#OpenDisk)**,**](#SetDevice)[**SetDevice**](#SetDevice)

**Example:**

## disk high-level

------------------- ---- -------------------------------------------- -----

[[**DeleteFile**](#DeleteFile)](#DeleteFile) C238 Delete a file.56

[**EnterDeskTop**](#EnterDeskTop) C22C Load and run DESKTOP.57

[[**FindFile**](#FindFile)](#FindFile) $C20B Lookup a file in the directory

[[**FindFTypes**](#FindFTypes)](#FindFile) $C23B Create a table of file names -401

[**GetFile**](#GetFile) $C208 Load a file, given a file name -410

[[**GetPtrCurDkNm**](#GetPtrCurDkNm)](#FindFile) $C298 Compute address of disk's name -315

[**OpenDisk**](#OpenDisk) $C2A1 Open a disk

[**RenameFile**](#RenameFile) $C259 Rename a file -405

RstrAppl $C23E Load the SWAPFILE -409

[**SaveFile**](#SaveFile) $C1ED Save memory to a file

[**SetDevice**](#SetDevice) $C2B0 Select a drive

[**SetGEOSDisk**](#SetGEOSDisk) C1EA Convert a disk to GEOS format

### ****[DeleteFile](#DeleteFile)****: (C64, C128) ****[C238](#hC238)****

**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**](#zr0)FILENAME — pointer to null-terminated name of file to delete

**Uses:** [[[[**curDrive**](#d_curDrive)](#curDirHead)](#d_curDevice)](#curDirHead)

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

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

**Alters:** [**diskBlkBuf**](#d_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**](#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**](#zr0)**-r9.**

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

[[**DeleteFile**](#DeleteFile)](#DeleteFile) first calls [[**FindFile**](#FindFile)](#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**](#hC22C)

**Function:** Standard application exit to GEOS deskTop.

**Parameters:** none.

**Returns:** *never returns to application.*

**Description:** [**EnterDeskTop**](#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**](#EnterDeskTop) will prompt the user to insert a disk with a copy of the deskTop on it.

**Example:**

**See also: RstrAppl,** [**GetFile**](#GetFile)**.**

### ****[FindFile](#FindFile)****: (C64, C128) [**C20B**](#hC20B)

**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**](#d_curDrive)](#curDirHead)](#d_curDevice)](#curDirHead)

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

**$886E1** Flag byte

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

**r1** track/sector of directory block containing entry;

**r5** Pointer todirectory entry within diskBIkBuf.

**Alters: dirEntryBuf** directory entry of file if found.

[**diskBlkBuf**](#d_diskBlkBuf)contains directory block where FILENAME found.

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

**Description**: **Given a null-terminated filename,** [[**FindFile**](#FindFile)](#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**](#exLoadBASIC)

**See also: GetlstDirEntry, GetNxtDirEntry,** [[**FindFTypes**](#FindFTypes)](#FindFile)

### [****[FindFTypes](#FindFile)****](#FindFTypes): (C64, C128) [**C23B**](#hC23B)

**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**](#d_curDrive)](#curDirHead)](#d_curDevice)](#curDirHead)

**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**](#d_diskBlkBuf)used as temporary buffer for directory blocks.

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

**Description**:[[[[**FindFTypes**](#FindFTypes)](#FindFile)](#FindFTypes)](#FindFile) **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**](#FindFTypes)](#FindFile) **first clears enough of the area at BUFFER to hold MAXFILES**

**filenames then calls GetlstDirEntry 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**](#FindFTypes)](#FindFile)](#FindFTypes)](#FindFile) **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:**

### ****[GetFile](#GetFile)****: (C64, C128) [****C208****](#hC208)

**See also:** [[**FindFile**](#FindFile)](#FindFile)**, GetlstDirEntry, GetNxtDirEntry.**

**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**](#d_curDrive)](#curDirHead)](#d_curDevice)](#curDirHead)

**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**](#GetFile).

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

**r3** as originally passed to [**GetFile**](#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**](#GetFile).

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**](#GetFile)](#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**](#OpenRecord) **and** [**ReadRecord**](#ReadRecord)**). Most applications will use** [**GetFile**](#GetFile) **to load and execute desk accessories when the user clicks on an item in the geos menu. Some applications will use** [**GetFile**](#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)**.**

[**GetFile**](#GetFile) **first calls** [**FindFile**](#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**](#GetFile) **calls** [**LdDeskAcc**](#LdDeskAcc)**. If the file is type APPLICATION or type**

**AUTO\_EXEC,** [**GetFile**](#GetFile) **calls** [**LdApplic**](#LdApplic)**. All other file types are loaded with**

**the generic** [[[[[[**LdFile**](#LdFile)](#LdDeskAcc)](#LdFile)](#LdDeskAcc)](#LdFile)](#LdDeskAcc)**.**

**The following GEOS constants can be used to set the LOAD\_OPT parameter**

**when loading an application:**

|  |  |
| --- | --- |
| **ST\_LD\_AT\_ADDR** | **Load at address: load application at the address passed in r7 as opposed to the address in the file header.** |
| **ST\_LD\_DATA** | **Load data file: application is being passed the name of a data file to load.** |
| **ST\_PR\_DATA** | **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.**

1. **not necessary for desk accessory to save color memory.**

**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.)**

**See** [**LdDeskAcc Note**](#DANote) **for more information.**

**Example:**

**See also:** [[[[[[**LdFile**](#LdFile)](#LdDeskAcc)](#LdFile)](#LdDeskAcc)](#LdFile)](#LdDeskAcc)**, LdDeskAcc,** [**LdApplic**](#LdApplic)

### [****[GetPtrCurDkNm](#FindFile)****](#GetPtrCurDkNm): (C64, C128) ****[C298](#hC298)****

**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**](#d_curDrive)](#curDirHead)](#d_curDevice)](#curDirHead)

**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**](#GetPtrCurDkNm)](#FindFile)](#GetPtrCurDkNm)](#FindFile) **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**](#GetPtrCurDkNm)](#FindFile) **would return the address of DrACurDkNm. If drive B is the current drive then** [[**GetPtrCurDkNm**](#GetPtrCurDkNm)](#FindFile) **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**](#GetPtrCurDkNm)](#FindFile) **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**](#GetPtrCurDkNm)](#FindFile) **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:**

**See also:**

### OpenDisk: (C64, C128) [****C2A1****](#hC2A1)

**Function:** Open the disk in the current drive

**Parameters:** None:

**Uses:** [[[[**curDrive**](#d_curDrive)](#curDirHead)](#d_curDevice)](#curDirHead)drive that disk is in. Set by call [**SetDevice**](#SetDevice)

**driveType** type of drive to open (for shadowing information)

**Calls**: [**NewDisk**](#NewDisk), [**GetDirHead**](#GetDirHead), [**ChkDkGEOS**](#ChkDkGEOS), [[**GetPtrCurDkNm**](#GetPtrCurDkNm)](#FindFile)

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

**r5** pointer to disk name buffer as returned from [[**GetPtrCurDkNm**](#GetPtrCurDkNm)](#FindFile). This is a pointer to one of the DrxCurDkNm arrays.

**Alters: Dn*x*CurDkNm** current disk name array contains disk name

[**curDirHead**](#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**](#zr0)**-r4.**

**Description:** [**OpenDisk**](#OpenDisk)initiates access to the disk in the current drive. [**OpenDisk**](#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**](#OpenDisk)immediately after calling [**SetDevice**](#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**](#OpenDisk)first calls [**NewDisk**](#NewDisk)to tell the disk drive a new disk has been inserted (if the disk is shadowed, the shadow memory is also cleared). [**GetDirHead**](#GetDirHead) is then called to load the disk's header block and BAM into [**curDirHead**](#curDirHead). With a valid header block in memory, [**ChkDkGEOS**](#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**](#OpenDisk)copies the disk name string from [**curDirHead**](#curDirHead) to the disk name buffer returned by [[**GetPtrCurDkNm**](#GetPtrCurDkNm)](#FindFile).

**Note**: This Routine calls [**GetDirHead**](#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**](#FreeFile)**,** [**FreeBlock**](#FreeBlock)**,** [**SetDevice**](#SetDevice)**.**

### ****[RenameFile](#RenameFile)****: (C64, C128) [**C259**](#hC259)

**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**](#d_curDrive)](#curDirHead)](#d_curDevice)](#curDirHead)drive that disk is in. Set by call [**SetDevice**](#SetDevice)

**driveType** type of drive to open (for shadowing information)

**Calls**: [**NewDisk**](#NewDisk), [**GetDirHead**](#GetDirHead), [**ChkDkGEOS**](#ChkDkGEOS), [[**GetPtrCurDkNm**](#GetPtrCurDkNm)](#FindFile)

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

**Alters:** [**diskBlkBuf**](#d_diskBlkBuf)used for temporary block storage.

**dirEntryBuf** old directory entry.

[**curDirHead**](#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, r**[**4**](#zr0)**-r6.**

**Description:** [[**RenameFile**](#RenameFile)](#RenameFile)searches the current directory for OLDFILE and changes the name string in the directory entry to NEWFILE.

[**RenameFile**](#RenameFile) first calls [**FindFile**](#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 [**GetlstDirEntry**](#GetlstDirEntry)and [**GetNxtDirEntry**](#GetNxtDirEntry)to establish the old file

name, do not pass [[**RenameFile**](#RenameFile)](#RenameFile)a pointer into [**diskBlkBuf**](#d_diskBlkBuf). Copy the file name from [**diskBlkBuf**](#d_diskBlkBuf) to another buffer (such as **dirEntryBuf**) and pass [[**FreeFile**](#FreeFile)](#FreeFile)the pointer to that buffer. Otherwise when [[**FreeFile**](#FreeFile)](#FreeFile)uses [**diskBlkBuf**](#d_diskBlkBuf) it will corrupt the file name.

**Note3:** This Routine calls [[**FindFile**](#FindFile)](#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**](#RenameFile).

**Example:**

**See also:** [[**FreeFile**](#FreeFile)](#FreeFile)**,** [**FreeBlock**](#FreeBlock)**.**

### 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.

**Required Offsets into Header to Set**

.word $00 = Pointer to Filename

.byte $69 = DOS File Type

.byte $70 = GEOS Filetype

.byte $71 = GEOS Structure (SEQ or VLIR)

.word $72 = Memory to Save Start Address

(Set to $0000 for an empty file)

.word $74 = Memory to Save End Address

(Set to $FFFF for an empty file)

**r10L** DIRPAGE Directory page to begin searching for an empty directory slot.

**Uses:** [[[[**curDrive**](#d_curDrive)](#curDirHead)](#d_curDevice)](#curDirHead)

**year, month, day, hours, minutes** for date-stamping file**.**

**curType** GEOS 64 vl.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**](#d_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**](#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**](#zr0)**-r8**

**Description**: [**SaveFile**](#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**](#SaveFile)calls **SetGDirEntry** and [**BlkAllOC**](#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**

**Note1,3: 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).**

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

**See also**: [**GetFile**](#GetFile)**,** [**OpenRecordFile**](#OpenRecord)**.**

**Example:**

### SetDevice: (C64, C128) [****C2B0****](#hC2B0)

**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**](#d_curDevice)](#curDirHead)currently active device.

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

**Alters:** [[**curDevice**](#d_curDevice)](#curDirHead)new current device number.

[[[[**curDrive**](#d_curDrive)](#curDirHead)](#d_curDevice)](#curDirHead)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**](#SetDevice) **changes the active device and is used primarily to switch from one disk drive to another.** [**SetDevice**](#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)**.**

[**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**](#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**](#SetDevice) **calls** [**ExitTurbo**](#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**](#SetDevice) **to change the active device. An application should never directly modify** [[[[**curDrive**](#d_curDrive)](#curDirHead)](#d_curDevice)](#curDirHead) **or** [[**curDevice**](#d_curDevice)](#curDirHead)**.**

**Example:**

**See also:** [**OpenDisk**](#OpenDisk)**,** [**ChangeDiskDevice**](#ChangeDiskDevice)

### SetGEOSDisk: (C64, C128) [****C1EA****](#hC1EA)

**Function**: Convert Commodore disk to GEOS format.

**Parameters**: none.

**Uses:** [[[[**curDrive**](#d_curDrive)](#curDirHead)](#d_curDevice)](#curDirHead)

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

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

**Alters:** [**curDirHead**](#curDirHead)directoryheader is read from disk.

**dir2Head** (BAM for 1571 and 1581 drives only)

**dir3Head** (BAM for 1581 drive only)

**Destroys**: **a,y**

**Description**: [**SetGEOSDisk**](#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**](#OpenDisk)returns the **isGEOS** flag set to FALSE. Typically the user is prompted before the conversion.

[**SetGEOSDisk**](#SetGEOSDisk)expects the disk to have been previously opened with

[**OpenDisk**](#OpenDisk). It first calls [**GetDirHead**](#GetDirHead) to read the directory header into memory then calls [[**CalcBlksFree**](#CalcBlksFree)](#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**](#ChkDkGEOS)

## disk VLIR

------------------- ---- -------------------------------------------- -----

[**AppendRecord**](#AppendRecord) C289 Add a VLIR chain

[**CloseRecordFile**](#CloseRecordFile) C277 Close a VLIR file

[**DeleteRecord**](#DeleteRecord) C283 Remove a VLIR chain

[**InsertRecord**](#InsertRecord) C286 Insert a VLIR chain

[**NextRecord**](#NextRecord) C27A Move to next VLIR chain

[**OpenRecordFile**](#OpenRecord) C274 Open a VLIR file

[**PointRecord**](#PointRecord) C280 Go to a specific VLIR chain

[**PreviousRecord**](#PreviousRecord) C27D Move to previous VLIR chain

[**ReadRecord**](#ReadRecord) C28C Load a VLIR chain

[[[[**UpdateRecordFile**](#UpdateRecordFile)](#UpdateRecordFile)](#UpdateRecordFile)](#UpdateRecordFile) C295 Update a VLIR file

[[[[[[[[**WriteRecord**](#WriteRecord)](#UpdateRecordFile)](#UpdateRecordFile)](#UpdateRecordFile)](#WriteRecord)](#UpdateRecordFile)](#UpdateRecordFile)](#UpdateRecordFile) C28F Save memory to a VLIR chain

### **AppendRecord:** (C64, C128) [****C289****](#hC289)

**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**](#d_curDrive)](#curDirHead)](#d_curDevice)](#curDirHead)drive that disk is in. Set by call [**SetDevice**](#SetDevice)

**fileWritten†** if FALSE, assumes record just opened (or updated) and

reads BAM/VBM into memory. *ANOTE: Confirm*

[[[[[[**curRecord**](#d_curRecord)](#curDirHead)](#curDirHead)](#curDirHead)](#curDirHead)](#curDirHead)Current record number

**fileHeader** VLIR index table.

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

[**curDirHead**](#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**](#d_curRecord)](#curDirHead)](#curDirHead)](#curDirHead)](#curDirHead)](#curDirHead)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**](#CloseRecordFile)*to write the changes to disk.*

**Destroys: a,y,** [**r0**](#zr0)**L,** **r**[**1**](#zr0)**L, r4**

**Preparatory routines1:** [**OpenRecordFile**](#OpenRecord)

**Description:** [**AppendRecord**](#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**](#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**](#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**](#WriteRecord)](#UpdateRecordFile)](#UpdateRecordFile)](#UpdateRecordFile).

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

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

**See also:** [**InsertRecord**](#InsertRecord), [**DeleteRecord**](#DeleteRecord), [**PointRecord**](#PointRecord)

**Example:** [**SaveRecord**](#ExSaveRecord)

### **CloseRecordFile**: (C64, C128) [****C277****](#hC277)

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

**Parameters:** none.

**Uses**:[[[[**curDrive**](#d_curDrive)](#curDirHead)](#d_curDevice)](#curDirHead)drive that disk is in. Set by call [**SetDevice**](#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 vl.3 and later for detecting REU shadowing

[**curDirHead**](#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**](#CloseRecordFile)*to write the changes to disk.*

**Destroys: a,y,** **r**[**1**](#zr0)**, r4, r5.**

**Preparatory routines1:** [**OpenRecordFile**](#OpenRecord)

**Description:** **[CloseRecordFile](#CloseRecordFile)** first calls [[**UpdateRecordFile**](#UpdateRecordFile)](#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**](#UpdateRecordFile)](#UpdateRecordFile).

**See also:** [**OpenRecordFile**](#InsertRecord), **UpdateRecordFile**

**Example:** [**SaveRecord**](#ExSaveRecord)

### **DeleteRecord**: (C64, C128) [****C283****](#hC283)

**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**](#d_curDrive)](#curDirHead)](#d_curDevice)](#curDirHead)drive that disk is in. Set by call [**SetDevice**](#SetDevice)

**fileWritten†** if FALSE, assumes record just opened (or updated) and

reads BAM into memory.

[[[[[[**curRecord**](#d_curRecord)](#curDirHead)](#curDirHead)](#curDirHead)](#curDirHead)](#curDirHead)Current record number

**fileHeader** VLIR index table.

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

[**curDirHead**](#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**](#d_curRecord)](#curDirHead)](#curDirHead)](#curDirHead)](#curDirHead)](#curDirHead)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**](#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**](#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**](#DeleteRecord)does not update the BAM and VLIR file information on the disk. Call **CloseRecordFile** or [[**UpdateRecordFile**](#UpdateRecordFile)](#UpdateRecordFile) to update the file when done modifying.

**See also:** [**AppendRecord**](#AppendRecord), [**InsertRecord**](#InsertRecord)

**Example:**

### **InsertRecord**: (C64, C128) ****[C286](#hc286)****

**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**](#d_curDrive)](#curDirHead)](#d_curDevice)](#curDirHead)drive that disk is in. Set by call [**SetDevice**](#SetDevice)

**fileWritten†** if FALSE, assumes record just opened (or updated) and

reads BAM/VBM into memory.

[[[[[[**curRecord**](#d_curRecord)](#curDirHead)](#curDirHead)](#curDirHead)](#curDirHead)](#curDirHead)Current record number

**fileHeader** VLIR index table.

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

[**curDirHead**](#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**](#d_curRecord)](#curDirHead)](#curDirHead)](#curDirHead)](#curDirHead)](#curDirHead)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,** [**r0**](#zr0)**L**

**Preparatory routines1:** [**OpenRecordFile**](#OpenRecord)

**Description:** [**InsertRecord**](#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**](#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**](#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.

**See also:** [**ReadRecord**](#ReadRecord), [[[[**WriteRecord**](#WriteRecord)](#UpdateRecordFile)](#UpdateRecordFile)](#UpdateRecordFile), [**CloseRecordFile**](#CloseRecordFile)**, UpdateRecordFile**

**Example:** [**SaveRecord**](#ExSaveRecord)

### **NextRecord**: (C64, C128) **[C27A](#hC27A)**

**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**](#d_curRecord)](#curDirHead)](#curDirHead)](#curDirHead)](#curDirHead)](#curDirHead)new record number

**Destroys:** nothing

**Preparatory routines1:** [**OpenRecordFile**](#OpenRecord)

**Description:** [**NextRecord**](#NextRecord)makes the current record plus one the new current record. A

subsequent call to [**ReadRecord**](#ReadRecord)or [[[[**WriteRecord**](#WriteRecord)](#UpdateRecordFile)](#UpdateRecordFile)](#UpdateRecordFile) will operate with this record.

If the record does not exist, then [**NextRecord**](#NextRecord)returns an then [**NextRecord**](#NextRecord)returns an **INV\_RECORD** (invalid record) error.

**See also:** [**PreviousRecord**](#PreviousRecord)**,** [**PointRecord**](#PointRecord)**.**

**Example:** [**SaveRecord**](#ExSaveRecord)

### **OpenRecordFile**: (C64, C128) [**C274**](#hC274)

**Function:** Open an existing VLIR file for access.

**Parameters:** [**r0**](#zr0)FILENAME—pointer to null-terminated name of file (word).

**Uses**:[[[[**curDrive**](#d_curDrive)](#curDirHead)](#d_curDevice)](#curDirHead)drive that disk is in. Set by call SetDevice

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

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

**STRUCT­\_MISMATCH**

**rlL** track/Sector of VLIR Block

**r1H**

**r5** pointer into [**diskBlkBuf**](#d_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**](#d_curRecord)](#curDirHead)](#curDirHead)](#curDirHead)](#curDirHead)](#curDirHead)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**](#SetDevice)**,**](#OpenRecord)[**OpenDisk**](#OpenDisk)

**Description:** Before accessing the data in a VLIR file, an application must call

[**OpenRecordFile**](#OpenRecord). [**OpenRecordFile**](#OpenRecord)searches the current directory for

FILENAME and, if it finds it, loads the index table into fileHeader.

[**OpenRecordFile**](#OpenRecord)initializes the GEOS VLIR variables (both local and global) to allow other VLIR routines such as [[[[**WriteRecord**](#WriteRecord)](#UpdateRecordFile)](#UpdateRecordFile)](#UpdateRecordFile) 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**](#OpenRecord)

will return a **STRUCT\_MISMATCH** error.

**Note**: An application can create an empty VLIR file with [**SaveFile**](#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**](#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.

**See also:** [**ReadRecord**](#ReadRecord), [[[[**WriteRecord**](#WriteRecord)](#UpdateRecordFile)](#UpdateRecordFile)](#UpdateRecordFile), [**CloseRecordFile**](#CloseRecordFile)**, UpdateRecordFile**

**Example:** [**SaveRecord**](#ExSaveRecord)

### **PointRecord**: (C64, C128) [****C280****](#hC280)

**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**](#d_curRecord)](#curDirHead)](#curDirHead)](#curDirHead)](#curDirHead)](#curDirHead)new record number

**Destroys:** nothing

**Preparatory routines1:** [**OpenRecordFile**](#OpenRecord)

**Description:** [**PointRecord**](#PointRecord)makes RECORD the current record so that a subsequent call to **ReadRecord** or [[[[**WriteRecord**](#WriteRecord)](#UpdateRecordFile)](#UpdateRecordFile)](#UpdateRecordFile) 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**](#PointRecord)returns an **INV\_RECORD** (invalid record) error.

**See also:** [**NextRecord**](#NextRecord)**,** [**PreviousRecord**](#PreviousRecord)**.**

**Example:** [**SaveRecord**](#ExSaveRecord)

### **PreviousRecord:** (C64, C128) [****C27D****](#hC27D)

**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**](#d_curRecord)](#curDirHead)](#curDirHead)](#curDirHead)](#curDirHead)](#curDirHead)new record number

**Destroys:** nothing

**Preparatory routines1:** [**OpenRecordFile**](#OpenRecord)

**Description:** [**PreviousRecord**](#PreviousRecord)makes the current record minus one the new current record. A subsequent call to [**ReadRecord**](#ReadRecord)or [[[[**WriteRecord**](#WriteRecord)](#UpdateRecordFile)](#UpdateRecordFile)](#UpdateRecordFile) will operate with this record.

If the record does not exist, then [**PreviousRecord**](#PreviousRecord)returns an **INV\_RECORD** (invalid record) error.

**See also:** [**NextRecord**](#NextRecord)**,** [**PointRecord**](#PointRecord)**.**

**Example:** [**SaveRecord**](#ExSaveRecord)

### **ReadRecord**: (C64, C128) [****C28C****](#hC28C)

**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**](#d_curDrive)](#curDirHead)](#d_curDevice)](#curDirHead)drive that disk is in. Set by call [**SetDevice**](#SetDevice)

[[[[[[**curRecord**](#d_curRecord)](#curDirHead)](#curDirHead)](#curDirHead)](#curDirHead)](#curDirHead)Current record number

**fileHeader** VLIR index table. Table holds Track / Sector of first block of each record.

**curType** GEOS 64 vl.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**](#d_diskBlkBuf) to the application's buffer space, would have exceeded the size of **BUFFER**. The process of copying any extra data from [**diskBlkBuf**](#d_diskBlkBuf) to the end of **BUFFER** is left to the application. The data starts at [**diskBlkBuf**](#d_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 routines1:** [**OpenRecordFile**](#OpenRecord)

**Description:** [**ReadRecord**](#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**](#ReadRecord)calls **ReadFile** to load the chain of blocks into memory.

**See also:** [[[[[**WriteRecord**](#WriteRecord)](#UpdateRecordFile)](#UpdateRecordFile)](#UpdateRecordFile)](#NextRecord)**,** [**ReadFile**](#PointRecord)**.**

**Example:**

### [**[UpdateRecordFile](#UpdateRecordFile)**](#UpdateRecordFile): (C64, C128) [****C295****](#hC295)

**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**](#d_curDrive)](#curDirHead)](#d_curDevice)](#curDirHead)drive that disk is in. Set by call [**SetDevice**](#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 vl.3 and later for detecting REU shadowing

[**curDirHead**](#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**](#UpdateRecordFile)](#UpdateRecordFile)](#UpdateRecordFile)](#UpdateRecordFile)checks the **fileWritten** flag. If the flag is **TRUE**, which indicates the file has been altered since it was last updated, [[[[**UpdateRecordFile**](#UpdateRecordFile)](#UpdateRecordFile)](#UpdateRecordFile)](#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**](#UpdateRecordFile)](#UpdateRecordFile)](#UpdateRecordFile)](#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.

**See also:** [**CloseRecordFile**](#CloseRecordFile)**,** [**OpenRecordFile**](#OpenRecord)**.**

**Example:**

### [[[**[WriteRecord](#UpdateRecordFile)**](#WriteRecord)](#UpdateRecordFile)](#UpdateRecordFile)**:** (C64, C128) [****C28F****](#hC28F)

**Function:** Write data to the current VLER record.

**Parameters:** none.

**Uses:** [[[[**curDrive**](#d_curDrive)](#curDirHead)](#d_curDevice)](#curDirHead)drive that disk is in. Set by call [**SetDevice**](#SetDevice)

**fileWritten**if FALSE, assumes record just opened (or updated) and

[[[[[[**curRecord**](#d_curRecord)](#curDirHead)](#curDirHead)](#curDirHead)](#curDirHead)](#curDirHead)Current record number

**fileHeader** VLIR index table.

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

[**curDirHead**](#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**](#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**](#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**](#WriteRecord)](#UpdateRecordFile)](#UpdateRecordFile)](#UpdateRecordFile)](#WriteRecord)](#UpdateRecordFile)](#UpdateRecordFile)](#UpdateRecordFile) **writes data to the current record All blocks previously associated with the record are freed.** [**BlkAllOC**](#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**](#WriteRecord)](#UpdateRecordFile)](#UpdateRecordFile)](#UpdateRecordFile)](#WriteRecord)](#UpdateRecordFile)](#UpdateRecordFile)](#UpdateRecordFile) **does not write the BAM and internal VLIR file information to disk. Call** [**CloseRecordFile**](#CloseRecordFile) **or** [[[[**UpdateRecordFile**](#UpdateRecordFile)](#UpdateRecordFile)](#UpdateRecordFile)](#UpdateRecordFile) **when done to update the disk with this information.**

**Note:** [[[[[[[[**WriteRecord**](#WriteRecord)](#UpdateRecordFile)](#UpdateRecordFile)](#UpdateRecordFile)](#WriteRecord)](#UpdateRecordFile)](#UpdateRecordFile)](#UpdateRecordFile)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.

**See also:** [**ReadRecord**](#ReadRecord)**,** [**WriteFile**](#OpenRecord)**.**

**Example:**

## disk DRIVER

------------------- ---- -------------------------------------------- -----

AddDirBlock 9039 **Needs Documenting...**

[CallDrvRoutine](#UpdateRecordFile): [**9042**](#hC28F) **Needs Documenting...**

[CheckDrvStatus](#UpdateRecordFile): [**9045**](#hC28F) **Needs Documenting...**

Get[**diskBlkBuf**](#d_diskBlkBuf) 903C **Needs Documenting...**

Put[**diskBlkBuf**](#d_diskBlkBuf) 903F **Needs Documenting...**

[UpdateRecordFile](#UpdateRecordFile)JmpIndX: [**9D80**](#hC28F) **Jump Table Needs Documenting...**

### **AddDirBlock:** (C64, C128) [****9042****](#hC28F)

**Function:** Call Directly into Disk Driver

**Parameters:**

**Uses:**

**Returns:**

**Alters:**

**Destroys:**

**Description:**

**Note: Needs Documenting...**

**See also:**

**Example:**

### [**CallDrvRoutine**](#UpdateRecordFile)**:** (C64, C128) [****9042****](#hC28F)

**Function:** Call Directly into Disk Drive

**Parameters:**

**Uses:**

**Returns:**

**Alters:**

**Destroys:**

**Description:**

**Note: Needs Documenting...**

**See also:**

**Example:**

### **Get**[**diskBlkBuf**](#d_diskBlkBuf)**:** (C64, C128) [****9045****](#hC28F)

**Function:** Call Directly into Disk Driver

**Parameters:**

**Uses:**

**Returns:**

**Alters:**

**Destroys:**

**Description:**

**Note: Needs Documenting...**

**See also:**

**Example:**

### **Put**[**diskBlkBuf**](#d_diskBlkBuf)**:** (C64, C128) [****9045****](#hC28F)

**Function:** Call Directly into Disk Driver

**Parameters:**

**Uses:**

**Returns:**

**Alters:**

**Destroys:**

**Description:**

**Note: Needs Documenting...**

**See also:**

**Example:**

### [**CheckDrvStatus**](#UpdateRecordFile)**:** (C64, C128) [****9045****](#hC28F)

**Function:** Call Directly into Disk Driver

**Parameters:**

**Uses:**

**Returns:**

**Alters:**

**Destroys:**

**Description:**

**Note: Needs Documenting...**

**See also:**

**Example:**

### **JmpIndX:** (C64, C128) [****9D80****](#hC28F)

**Function:** Jump Table

**Parameters:**

**Uses:**

**Returns:**

**Alters:**

**Destroys:**

**Description:**

**Note: Needs Documenting...**

**See also:**

**Example:**

## graphics

------------------- ---- -------------------------------------------------- ---

BitmapUp $C1AB Draw a click box -621

i\_BitmapUp $C1AB Draw a click box / inline -622

BitmapClip $C2AA Draw a coded image -623

BitOtherClip $C2C5 Draw a coded image with user patches -624

DisablSprite $C1D5 Turn off a sprite -628

DrawLine $C130 Draw/Erase/Copy an arbitrary line -615

[**DrawPoint**](#DrawPoint) $C133 Draw/Erase/Copy a point on the screen90

DrawSprite $C1C6 Copy a sprite data block -625

EnablSprite $C1D2 Turn on a sprite -627

FrameRectangle $C127 Draw an outline in a pattern -604

i\_FrameRectangle $C1A2 Draw a solid outline with inline data -605

GetScanLine $C13C Compute memory address of a row on the screen -618

GraphicsString $C136 Process a graphic command table -601

i\_GraphicsString $C1A8 Process a graphic command table / inline -602

HorizontalLine $C118 Draw a horizontal line in a pattern -616

InvertLine $C11B Reverse video a horizontal line -614

ImprintRectangle $C250 Copy a box from screen 2 to screen 1 -610 i\_ImprintRectangle $C253 Copy a box from screen 2 to screen 1 / inline -611

InvertRectangle $C12A Reverse video a box -612

PosSprite $C1CF Position a sprite -626

RecoverLine $C11E Copy a line from screen 2 to screen 1 -613

Rectangle $C124 Fill a box with a pattern -606

i\_Rectangle $C19F Fill a box with a pattern / inline -607

RecoverRectangle $C12D Copy a box from screen 1 to screen 2 -608

i\_RecoverRectangle $C1A5 Copy a box from screen 1 to screen 2 / inline -609

[**SetNewMode**](#SetNewMode) $C2DD Set GEOS 128 to 40 or 80 column mode91

SetPattern $C139 Select a fill pattern -603

TestPoint $C13F Test the value of a pixel -619

VerticalLine $C121 Draw a vertical line in a pattern -617

### DrawPoint: (C64, C128) $C133

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

**Parameters**: **r3** XI — 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 Draw set pixel using **dispBufferOn**.

0 0 Erase clear pixel using **dispBufferOn**.

**Uses**: when setting or clearing pixels:

**dispBuffer0n**:

bit 7 — write to foreground screen if set.

bit 6 — write to background screen if set.

**Destroys:**

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

**Description**:

[**DrawPoint**](#DrawPoint)is a very versatile routine. It can copy a point from one screen to another, as well as draw or erase it. This routine is called by DrawLine to draw lines.

The carry (c) flag and sign (n) flag in the processor status register (s) are used to pass information to [**DrawPoint**](#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 # to set the sign (n) flag.
* Use lda #0 to clear the sign (n) flag.
* **128**: **DOUBLE\_W**,**ADD1­\_W** can be used on r3

**Example**:

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

### 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**](#zr0)**-r15**

**Description:** [**SetNewMode**](#SetNewMode)the Operation mode of the Commodore 128.

|  |
| --- |
| 40-column mode (graphMode == GR\_40) |

1: 8510 clock speed is slowed down to lMhz 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**](#exChangeMode)

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

## icon/menu

------------------- ---- -------------------------------------------- -----

[[[[[[[[**DoIcons**](#DoIcons)](#UpdateRecordFile)](#UpdateRecordFile)](#UpdateRecordFile)](#WriteRecord)](#UpdateRecordFile)](#UpdateRecordFile)](#UpdateRecordFile) C15A Display and begin interaction with icons.93

[[[[[[[[**DoMenu**](#DoMenu)](#UpdateRecordFile)](#UpdateRecordFile)](#UpdateRecordFile)](#WriteRecord)](#UpdateRecordFile)](#UpdateRecordFile)](#UpdateRecordFile) C151 Display and begin interaction with menus. 94

DoPreviousMenu $C190 Close current menu 1-702

GotoFirstMenu $C1BD Close all menu levels 1-703

RecoverAllMenus $C157 Erase all menus 1-706

[[[[[[[[[[[[[[[[**RecoverMenu**](#RecoverMenu)](#UpdateRecordFile)](#UpdateRecordFile)](#UpdateRecordFile)](#WriteRecord)](#UpdateRecordFile)](#UpdateRecordFile)](#UpdateRecordFile)](#RecoverMenu)](#UpdateRecordFile)](#UpdateRecordFile)](#UpdateRecordFile)](#WriteRecord)](#UpdateRecordFile)](#UpdateRecordFile)](#UpdateRecordFile) C154 Recover single menu from background buffer. 96

[[[[[[[[[[[[[[[[**ReDoMenu**](#ReDoMenu)](#UpdateRecordFile)](#UpdateRecordFile)](#UpdateRecordFile)](#WriteRecord)](#UpdateRecordFile)](#UpdateRecordFile)](#UpdateRecordFile)](#ReDoMenu)](#UpdateRecordFile)](#UpdateRecordFile)](#UpdateRecordFile)](#WriteRecord)](#UpdateRecordFile)](#UpdateRecordFile)](#UpdateRecordFile) C193 Reactivate menus at the current level. 97

### **DoIcons**: (C64, C128) [C15A](#hC15A)

**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**](#DoIcons)](#UpdateRecordFile)](#UpdateRecordFile)](#UpdateRecordFile)](#WriteRecord)](#UpdateRecordFile)](#UpdateRecordFile)](#UpdateRecordFile)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**](#DoIcons)](#UpdateRecordFile)](#UpdateRecordFile)](#UpdateRecordFile)](#WriteRecord)](#UpdateRecordFile)](#UpdateRecordFile)](#UpdateRecordFile)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**](#DoIcons)](#UpdateRecordFile)](#UpdateRecordFile)](#UpdateRecordFile)](#WriteRecord)](#UpdateRecordFile)](#UpdateRecordFile)](#UpdateRecordFile)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.

.

**See also:** [[[[[[[[**DoMenu**](#DoMenu)](#UpdateRecordFile)](#UpdateRecordFile)](#UpdateRecordFile)](#WriteRecord)](#UpdateRecordFile)](#UpdateRecordFile)](#UpdateRecordFile)**.**

**Example:** [**IconsUp**](#exIconsUp)

### [[[[[[[****[DoMenu](#UpdateRecordFile)****](#DoMenu)](#UpdateRecordFile)](#UpdateRecordFile)](#UpdateRecordFile)](#WriteRecord)](#UpdateRecordFile)](#UpdateRecordFile): (C64, C128) [C151](#hC151)

**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, **rO-r13**

**Description:** [[[[[[[[[[[[[[[[**DoMenu**](#DoMenu)](#UpdateRecordFile)](#UpdateRecordFile)](#UpdateRecordFile)](#WriteRecord)](#UpdateRecordFile)](#UpdateRecordFile)](#UpdateRecordFile)](#DoMenu)](#UpdateRecordFile)](#UpdateRecordFile)](#UpdateRecordFile)](#WriteRecord)](#UpdateRecordFile)](#UpdateRecordFile)](#UpdateRecordFile)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**](#DoMenu)](#UpdateRecordFile)](#UpdateRecordFile)](#UpdateRecordFile)](#WriteRecord)](#UpdateRecordFile)](#UpdateRecordFile)](#UpdateRecordFile) 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 **RecoverAllMenus**, which will erase any extant menus by recovering from the background buffer.

[[[[[[[[[[[[[[[[**DoMenu**](#DoMenu)](#UpdateRecordFile)](#UpdateRecordFile)](#UpdateRecordFile)](#WriteRecord)](#UpdateRecordFile)](#UpdateRecordFile)](#UpdateRecordFile)](#DoMenu)](#UpdateRecordFile)](#UpdateRecordFile)](#UpdateRecordFile)](#WriteRecord)](#UpdateRecordFile)](#UpdateRecordFile)](#UpdateRecordFile)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**](#DoMenu)](#UpdateRecordFile)](#UpdateRecordFile)](#UpdateRecordFile)](#WriteRecord)](#UpdateRecordFile)](#UpdateRecordFile)](#UpdateRecordFile) always forces the mouse to a new position. If you do not want the mouse moved, surround the call to [[[[[[[[**DoMenu**](#DoMenu)](#UpdateRecordFile)](#UpdateRecordFile)](#UpdateRecordFile)](#WriteRecord)](#UpdateRecordFile)](#UpdateRecordFile)](#UpdateRecordFile) with code to save and restore the mouse positions. The following code fragment will install menus without moving the mouse.

[[[[[[[[**DoMenu**](#DoMenu)](#UpdateRecordFile)](#UpdateRecordFile)](#UpdateRecordFile)](#WriteRecord)](#UpdateRecordFile)](#UpdateRecordFile)](#UpdateRecordFile)2:

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**](#DoMenu)](#UpdateRecordFile)](#UpdateRecordFile)](#UpdateRecordFile)](#WriteRecord)](#UpdateRecordFile)](#UpdateRecordFile)](#UpdateRecordFile) ; 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**](#DoIcons)](#UpdateRecordFile)](#UpdateRecordFile)](#UpdateRecordFile)](#WriteRecord)](#UpdateRecordFile)](#UpdateRecordFile)](#UpdateRecordFile)**, GotoFirstMenu, DoPreviousMenu, Re**[[[[[[[[**DoMenu**](#DoMenu)](#UpdateRecordFile)](#UpdateRecordFile)](#UpdateRecordFile)](#WriteRecord)](#UpdateRecordFile)](#UpdateRecordFile)](#UpdateRecordFile)**.**

### [[[[[[[[[[[[[[[****[RecoverMenu](#UpdateRecordFile)****](#RecoverMenu)](#UpdateRecordFile)](#UpdateRecordFile)](#UpdateRecordFile)](#WriteRecord)](#UpdateRecordFile)](#UpdateRecordFile)](#UpdateRecordFile)](#RecoverMenu)](#UpdateRecordFile)](#UpdateRecordFile)](#UpdateRecordFile)](#WriteRecord)](#UpdateRecordFile)](#UpdateRecordFile): (C64, C128) [C154](#hC154)

**Function:** Removes the current menu from the foreground screen by recovering from the background buffer.

**Parameters:** none.

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

**Description:** [[[[[[[[[[[[[[[[**RecoverMenu**](#RecoverMenu)](#UpdateRecordFile)](#UpdateRecordFile)](#UpdateRecordFile)](#WriteRecord)](#UpdateRecordFile)](#UpdateRecordFile)](#UpdateRecordFile)](#RecoverMenu)](#UpdateRecordFile)](#UpdateRecordFile)](#UpdateRecordFile)](#WriteRecord)](#UpdateRecordFile)](#UpdateRecordFile)](#UpdateRecordFile)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**](#RecoverMenu)](#UpdateRecordFile)](#UpdateRecordFile)](#UpdateRecordFile)](#WriteRecord)](#UpdateRecordFile)](#UpdateRecordFile)](#UpdateRecordFile)](#RecoverMenu)](#UpdateRecordFile)](#UpdateRecordFile)](#UpdateRecordFile)](#WriteRecord)](#UpdateRecordFile)](#UpdateRecordFile)](#UpdateRecordFile)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).

**See also:** [[[[[[[[**DoMenu**](#DoMenu)](#UpdateRecordFile)](#UpdateRecordFile)](#UpdateRecordFile)](#WriteRecord)](#UpdateRecordFile)](#UpdateRecordFile)](#UpdateRecordFile)**.**

**Example:**

### [[[[[[[[[[[[[[[****[ReDoMenu](#UpdateRecordFile)****](#ReDoMenu)](#UpdateRecordFile)](#UpdateRecordFile)](#UpdateRecordFile)](#WriteRecord)](#UpdateRecordFile)](#UpdateRecordFile)](#UpdateRecordFile)](#ReDoMenu)](#UpdateRecordFile)](#UpdateRecordFile)](#UpdateRecordFile)](#WriteRecord)](#UpdateRecordFile)](#UpdateRecordFile)**:** (C64, C128) [C193](#hC193)

**Function:** Reactivate menus at the current level.

**Parameters:** none.

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

**Description:** [[[[[[[[[[[[[[[[**ReDoMenu**](#ReDoMenu)](#UpdateRecordFile)](#UpdateRecordFile)](#UpdateRecordFile)](#WriteRecord)](#UpdateRecordFile)](#UpdateRecordFile)](#UpdateRecordFile)](#ReDoMenu)](#UpdateRecordFile)](#UpdateRecordFile)](#UpdateRecordFile)](#WriteRecord)](#UpdateRecordFile)](#UpdateRecordFile)](#UpdateRecordFile)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**](#ReDoMenu)](#UpdateRecordFile)](#UpdateRecordFile)](#UpdateRecordFile)](#WriteRecord)](#UpdateRecordFile)](#UpdateRecordFile)](#UpdateRecordFile)](#ReDoMenu)](#UpdateRecordFile)](#UpdateRecordFile)](#UpdateRecordFile)](#WriteRecord)](#UpdateRecordFile)](#UpdateRecordFile)](#UpdateRecordFile)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**](#ReDoMenu)](#UpdateRecordFile)](#UpdateRecordFile)](#UpdateRecordFile)](#WriteRecord)](#UpdateRecordFile)](#UpdateRecordFile)](#UpdateRecordFile)](#ReDoMenu)](#UpdateRecordFile)](#UpdateRecordFile)](#UpdateRecordFile)](#WriteRecord)](#UpdateRecordFile)](#UpdateRecordFile)](#UpdateRecordFile)will redraw the menu with the new text while leaving the menu open for another selection.

**See also:** [[[[[[[[**DoMenu**](#DoMenu)](#UpdateRecordFile)](#UpdateRecordFile)](#UpdateRecordFile)](#WriteRecord)](#UpdateRecordFile)](#UpdateRecordFile)](#UpdateRecordFile)**, GotoFirstMenu, DoPreviousMenu.**

**Example:**

## input driver

------------------- ---- -------------------------------------------- -----

[InitMouse](#InitMouse) $fe80 Initialize input device.

[SetMouse](#SetMouse) $fe89 Reset input device scanning circuitry.

[SetMouse](#SetMouse)Pic $c2da Set and preshift new soft-sprite mouse picture.

SlowMouse $fe83 Reset mouse velocity variables.

UpdateMouse $fe86 Update mouse variables from input device.

### **[InitMouse](#BBMult)**: (c64,C128) [**FE80**](#hFE80)

**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](#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**:

### **[SetMouse](#SetMouse)**: (C128) [**FE89**](#hFE89)

**See also**: **SlowMouse, UpdateMouse,** [SetMouse](#SetMouse)**, StartMouseMode,** **MouseUp.**

**Function**: Input device scan reset.

**Parameters**: none.

**Returns**: nothing.

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

**Description**: GEOS 128 calls [[SetMouse](#SetMouse)](#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](#SetMouse) routine that merely performs an rts. An application should never need to call [SetMouse](#SetMouse).

**Example**:

**See also**: **SlowMouse, UpdateMouse,** **Initmouse.**

## internal

------------------- ---- -------------------------------------------------- ---

[BootGeos](#BootGeos) C000 Reboot GEOS. 102

[[[[FirstInit](#FirstInit)](#BootGeos)](#Firstlnit)](#BootGeos) $C271 Initialize GEOS variables. 1-803

[[GetSerialNumber](#GetSerialNumber)](#BootGeos) C196 Return GEOS serial number. 104

InterruptMain $C100 Main interrupt level processing. 1-805

MainLoop $C1C3 GEOS's main loop. 1-804

Panic $C2C2 Report system errors. 1-711

ResetHandle $C003 internal Bootstrap entry point

### [**[BootGeos](#BBMult)**](#BootGeos): (c64,C128) [C000](#hC000)

**Function**: Restart GEOS from a non-GEOS application.

**Parameters**: none.

**Returns**: Does not return.

**Destroys:** n/a

**Description**: [[BootGeos](#BootGeos)](#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](#BootGeos) ($C000) and [BootGeos](#BootGeos)+$7f ($CO7f). The rest of the GEOS Kernal may be overwritten. To reboot GEOS, simply jmp [BootGeos](#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**](#d_CPU_DATA)](#d_CPU_DDR) 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](#BootGeos)

**Example**: [**RoadTrip**](#exRoadTrip)

**See also**:

### [[[[[**[FirstInit](#BootGeos)**](#FirstInit)](#BootGeos)](#Firstlnit)](#BootGeos)](#Firstlnit): (c64,C128) [**C271**](#hC271)

**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, **rO-r2**

**Description**: [[[[[[[[FirstInit](#FirstInit)](#BootGeos)](#Firstlnit)](#BootGeos)](#FirstInit)](#BootGeos)](#Firstlnit)](#BootGeos) 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](#FirstInit)](#BootGeos)](#Firstlnit)](#BootGeos)](#Firstlnit)](#BootGeos); a call to **UseSystemFont** may be necessary.

**Example**:

**See also**: **StartAppl**

### [**[GetSerialNumber](#BootGeos)**](#GetSerialNumber): (c64,C128) [**C196**](#hC196)

**See also**:

**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](#GetSerialNumber)](#BootGeos)](#GetSerialNumber)](#BootGeos) 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**:

## math

------------------- ---- -------------------------------------------------- ---

[BBMult](#BBMult) C160 Byte by byte (single-precision) unsigned multiply. 106

[Bmult](#Bmult) C163 Unsigned 16 bit by 8 bit multiply 107

[**Dabs**](#Dabs)C16F 16 bit absolute value 108

[**Ddec**](#DDec) C175 Decrement a 16 bit integer 109

[[**Ddiv**](#Ddiv)](#Ddiv)C169 Unsigned 16 bit division 110

[**DMult**](#DMult)C166 Unsigned 16 bit by 16 bit multiply 112

[**Dnegate**](#Dnegate)C172 Negate a 16 bit integer 113

[**DSDiv**](#DSDiv)C16C Signed 16 bit division 114

[**DShiftLeft**](#DShiftLeft)C15D Multiple 16 bit arithmetic shift left 115

[**DShiftRight**](#DShiftRight)C262 Multiple 16 bit logical shift right 116

### **[BBMult](#BBMult)**: (c64,C128) [C160](#hC160)

**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 = OPERANDl(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](#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**](#ex8BitMultiply)

### **BMult**: (c64,C128) **[C163](#hC163)**

**See also**: [**BMult**](#Bmult)**,** [**DMult**](#DMult)**,** [[**Ddiv**](#Ddiv)](#Ddiv)**, DSdiv**

**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 = OPERANDl(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**](#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](#Bmult) operates by clearing the high-byte of 0PERAND2 and calling [Bmult](#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**](#ex16x8Multiply)**,** [**ConvToUnits**](#exConvToUnits)

**See also**: [**BMult**](#Bmult)**,** [**DMult**](#DMult)**,** [[**Ddiv**](#Ddiv)](#Ddiv)**, DSdiv**

### **Dabs**: (c64,C128) [**C16F**](#hC16F)

**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**](#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**](#exDSmult)

**See also**: [**DNegate**](#DDec)

### **[Ddec](#Ddec):** (c64,C128) [[****C175****](#hC175)](#Ddec)

**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**](#Ddec)](#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**](#Ddec)](#DDec)is useful for loops which require a two-byte counter.

**Note3:** 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**](#Ddec)](#DDec)should be used when space is at a premium as it costs only 5 bytes to use. The kernal uses [[**Ddec**](#Ddec)](#DDec)in [**CRC**](#CRC)because space in the kernal is more valuable then the speed of the [**CRC**](#CRC)procedure that is not normally ever used in an inner loop. See Example [[**Ddec**](#Ddec)**vsDecW**](#exDDecVSDecW)

***Example****:* [**Kernal\_CRC**](#exKernal_CRC)**,** [[**Ddec**](#Ddec)**vsDecW**](#exDDecVSDecW)**,** [**DecCounter**](#exDecCounter)

### **[Ddiv](#Ddiv)**: (c64,C128) [[****C169****](#hC169)](#Ddec)

**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 = OPERANDl(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**](#Ddiv)](#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**](#Ddiv)](#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+l,y ; get high byte of divisor

bne 10$ ; if either non-zero, go divide

jmp DivideByZero ; else, flag error

10$

jmp [[**Ddiv**](#Ddiv)](#Ddiv)

There is no possibility of overflow (a result which cannot fit in 16 bits).

**Example**: [**ConvToUnits**](#exConvToUnits)**,** [**CheckDiskSpace**](#exCheckDiskSpace)

**See also**: [**DSDiv**](#DSDiv)**,** [**DMult**](#DMult)**,** [BBMult](#BBMult)**,** [**BMult**](#Bmult)

DivideBySeven: (Apple)

**Function**: Divide a byte value by 7

**Parameters**: [**r0L**](#zr0L) 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 [**r0**](#zr0)**L**

lsr

lsr

lsr

adc [**r0**](#zr0)**L**

ror

lsr

lsr

adc [**r0**](#zr0)**L**

ror

lsr

lsr

rts

### **DMult**: (c64,C128) [[****C166****](#hC166)](#Ddec)

**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 unchangedword pointed to by OPERAND contains the word result.

**Destroys**: **a,** **r6-r8**

**Description**: [**DMult**](#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**](#exDSmult)

**See also**: [**Bmult**](#Bmult)**,** [BBMult](#BBMult)**,** [[**Ddiv**](#Ddiv)](#Ddiv)**,** [**DSDiv**](#DSDiv)

### **Dnegate**: (c64,C128) [[****C172****](#hC172)](#Ddec)

**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**](#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**](#exDSmult)

**See also**: [**Dabs**](#Dabs)

### **DSDiv**: (c64,C128) [[****C16C****](#hC16C)](#Ddec)

**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**](#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**](#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**](#DSDiv) ;divide

**Example**:

### **DShiftLeft**: (c64,C128) [[****C15D****](#hC15D)](#Ddec)

**See also**: [[**Ddiv**](#Ddiv)](#Ddiv)**,** [**DMult**](#DMult)**,** [BBMult](#BBMult)**,** [**BMult**](#Bmult)

**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**](#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)

### **DShiftRight**: (c64,C128) [[****C262****](#hC262)](#Ddec)

**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**](#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**](#exMseToCardPos)**,** [**ConvToUnits**](#exConvToUnits)

**See also**: [**DShiftLeft**](#DShiftLeft)

## memory

------------------- ---- -------------------------------------------------- ---

[**ClearRam**](#ClearRam)$C178 Fill a memory region with zeroes 118

[**CmpFString**](#CmpFString)$C26E Memory block comparison 121

[**CmpString**](#CmpString)$C26B String compare 120

[**CopyFString**](#CopyFString)$C268 String copy 121

[**CopyString**](#CopyString)$C265 Memory block move 122

[**DoBOp**](#DoBOp)$C2EC (128) Back-RAM memory action primitive 123

[**DoRAMOp**](#DoRamOp) $C2D4 Perform any of the Below REU commands 130

[**FetchRAM**](#FetchRam) C2CB Retrieve memory from an REU 131

[**FillRam**](#FillRam)$C17B Memory block fill 124

i\_[**FillRam**](#FillRam)$C1B4 Memory block fill with inline data 124

[**InitRam**](#InitRam)$C181 Multiple memory location initialization 125

[**MoveBData**](#MoveBData)$C2E3(128) **M**ove data between front and back RAM 126

[**MoveData**](#MoveData)$C17E Intelligent memory block move 127

[**i­\_MoveData**](#MoveData)$C1B7 Inline Version inline data 127

[**StashRAM**](#StashRAM) $C2C8 Stash memory into an REU 132

[**SwapBData**](#SwapBData)$C2E6 (128) Swaps memory between Front and Back RAM 128

[**SwapRAM**](#SwapRAM) $C2CE Swap memory with an REU memory block 133

[**VerifyBData**](#VerifyBData) **$C2E9 (128)** Compares two regions of memory 129

[**VerifyRAM**](#VerifyRAM) $C2D1 Verify (compare) memory with REU 134

### **ClearRam**: (C64,C128) $C187

**Function**: Clear a region of memory to $00.

**Parameters**: [**ClearRam**](#ClearRam) COUNT — number of bytes to clear (0 - 64K) (word).

**r1** ADDR — address of area to clear (word).

**Returns**: nothing.

**Destroys**: **a,y,** [**r0**](#zr0)**,r1,r2L**

**Description**: [**ClearRam**](#ClearRam)clears COUNT bytes starting at ADDR to ADDRESS+COUNT. It useful for initializing .ramsect variable and data sections.

**Note**: Do not use [**ClearRam**](#ClearRam)to initialize [**r0**](#zr0)**-r2L.**

**Note:** Also, for when space is at a premium, it actually takes fewer bytes to call **i\_**[**FillRam**](#FillRam) with a fill value of $00.

**Note**:1 [**ClearRam**](#ClearRam)sets r2L to $00 and calls [**FillRam**](#FillRam).

**Example**: [**InitBuffers**](#exInitBuffers)

**See also**: [**FillRam**](#FillRam)**,** [**InitRam**](#InitRam)**.**

### **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**](#CmpFString)to function exactly like [**CmpString**](#CmpString), expecting a null terminated source string.

**Returns**: st status register flags reflect the result of the comparison.

**Destroys**: **a,x,y**

**Description**: [**CmpFString**](#CmpFString)compares the fixed-length string pointed to by SOURCE to the string of the same length pointed to by DEST.

[**CmpFString**](#CmpFString)with a LEN value of $00 causes the routine to act exactly like [**CmpString**](#CmpString).

[**CmpFString**](#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**](#exInitBuffers)

**See also**: [**CmpString**](#CmpString), [**CopyFString**](#CopyFString)

### **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**](#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**](#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**](#CmpString)cannot compare strings longer than 256 bytes (including the null). The compare process is aborted after 256 bytes.

**Example**: [**Find**](#exFind2)**2**

CopyFString: (C64,C128) $C268

**See also**: [**CmpFString**](#CmpFString)**,** [**CopyString**](#CopyString)**.**

**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**](#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**](#CopyFString)cannot copy a string longer than 255 bytes. A LEN value of $00 causes [**CopyFString**](#CopyFString)to act exactly like [**CopyString**](#CopyString).

**Note**: The source string may contain internal NULL'S. These will not terminate the copy operation.

**Example**: [**CopyBuffer**](#exCopyBuffer)

**See also**: [**CopyString**](#CopyString), [**CmpFString**](#CmpFString)**,** [**MoveData**](#MoveData)**.**

### **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**](#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**](#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**](#exCopyBuffer)

**See also**: [**CopyFString**](#CopyFString), [**CmpString**](#CmpString),[**MoveData**](#MoveData)**.**

### **DoBOp**: (C128) $C2EC

**Function**: Primitive for communicating with REU (RAM-Expansion Unit) devices.

**Parameters**: [**r0**](#zr0) 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***](#DoBOp) *MODE parameter closely matches the low nibble of the Do RAM Op CMD parameter.*

**Returns**: [**r0**](#zr0)**-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**](#DoBOp) is a generalized memory primitive for dealing with both memory banks on the Commodore 128. It is used by [**MoveBData**](#MoveBData), [**SwapBData**](#SwapBData), and [**VerifyBData**](#VerifyBData).

**Note**: [**DoBOp**](#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**](#MoveBData)**,** [**SwapBData**](#SwapBData)**,** [**VerifyBData**](#VerifyBData)**,** [**DoRAMOp**](#DoRamOp)

### **FillRam**: ,**I­\_FillRam:** (C64,C128) $C17B,$C1B4

**Function**: Fills a region of memory with a repeating byte value.

**Parameters**: Normal:

**[r0](#“zr0”)** COUNT — number of bytes to clear (0 - 64K) (word).

**rl** 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**](#zr0)**,r1**

**Description**: [**FillRam**](#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**](#FillRam)to initialize [**r0**](#zr0)**-r2L.**

**Note:**

**Example**: [**InitBuffers**](#exInitBuffers)

### **InitRam**: (C64,C128) $C181

**See also**: [**ClearRam**](#ClearRam)**,** [**InitRam**](#InitRam)**.**

**Function**: Table driven initialization for variable space and other memory areas.

**Parameters**: [**r0**](#zr0) TABLE —address of initialization table (word).

**Returns**: nothing.

**Destroys**: **a,x,y,** [**r0**](#zr0)**-r2L**

**Description**: [**InitRam**](#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**](#InitRam)expects the next address parameter.

**Note**: Do not use [**InitRam**](#InitRam)to initialize [**r0**](#zr0)**-r2L.**

**Example**:

**See also**: [**FillRam**](#FillRam)**,** [**ClearRam**](#ClearRam)**.**

### **MoveBData**: (C128) $C2E3

**Function**: Special version of [**MoveData**](#MoveData)that will move data within either front RAM or back RAM (or from one bank to the other).

**Parameters**: [**r0**](#zr0) 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**](#zr0)**-r3** unchanged.

**Destroys**: **a,x,y**

**Description**: [**MoveBData**](#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**](#MoveBData) is especially useful for copying data from front RAM to back RAM or from back RAM to front RAM.

[**MoveBData**](#MoveBData) uses the [**DoBOp**](#DoBOp) primitive by calling it with a MODE parameter of $00.

**Note**: [**MoveBData**](#MoveBData) should only be used to move data within the designated application areas of memory. [**MoveBData**](#MoveBData) is significantly slower than [**MoveData**](#MoveData)and should be avoided if the move will occur entirely within front RAM.

**Example**:

**See also**: [**MoveBData**](#MoveBData)**,** [**SwapBData**](#SwapBData)**,** [**VerifyBData**](#VerifyBData)**,** [**DoBOp**](#DoBOp).

### **MoveData**: ,I­\_MoveData: (C64,C128) $C2E3

**Function**: Moves a block data from one area to another.

**Parameters**: Normal:

[**r0**](#zr0) 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**](#zr0)**,R1,R2** unchanged.

**Destroys**: **a,y**

**Description**: [**MoveData**](#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**](#MoveData)option in the Configure program is enabled (GEOS vl.3 and later), [**MoveData**](#MoveData)will use part of bank 0 of the installed RAM-expansion unit for an ultrafast move operation. An application that calls [**MoveData**](#MoveData)in the normal manner will automatically take advantage of this selection. An application that relies upon a slower [[**MoveData**](#MoveData)](#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**](#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). [**MoveData**](#MoveData)should only be used to move data within the standard front RAM application space. Use [**MoveData**](#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**](#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**](#MoveData)on [**r0**](#zr0)-r6.

**Example**:

**See also**: [**MoveBData**](#MoveBData), [**CopyString**](#CopyString).

### **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**](#zr0) 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**](#zr0)**-r3** unchanged.

**Destroys**: **a,x, y**

**Description**:

[**SwapBData**](#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**](#SwapBData)is especially useful for swapping data from front RAM to back RAM or from back RAM to front RAM.

[**SwapBData**](#SwapBData)uses the [**DoBOp**](#DoBOp) primitive by calling it with a MODE parameter of $02.

**Note**: [**SwapBData**](#SwapBData)should only be used to swap data within the designated application areas of memory.

**Example**:

**See also**: [**MoveBData**](#MoveBData)**,** [**VerifyBData**](#VerifyBData)**,** [**DoBOp**](#DoBOp).

### **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**](#zr0) 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**](#zr0)**-r3** unchanged.

x $00 if data matches; $FF if mismatch.

**Destroys**: a,y

**Description**: [**VerifyBData**](#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**](#VerifyBData) uses the [**DoBOp**](#DoBOp) primitive by calling it with a MODE parameter of $03.

**Note**: [**VerifyBData**](#VerifyBData) should only be used to compare data within the designated application areas of memory.

**Example**:

**See also**: [**MoveBData**](#MoveBData)**,** [**SwapBData**](#SwapBData)**,** [**DoBOp**](#DoBOp).

### **DoRAMOp**: (c64 v1.3+,C128) $C2D4

**Function**: Primitive for communicating with REU (RAM-Expansion Unit) devices.

**Parameters**: [**r0**](#zr0) 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**](#zr0)**-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**](#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**](#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**](#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***](#DoRamOp)*CMD parameter closely matches the*

[***DoBOp***](#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**:

### **FetchRAM**: (c64 v1.3+,C128) [****C2CB****](#hC2CB)

**See also**: [**StashRAM**](#StashRAM)**,** [**FetchRAM**](#FetchRam)**,** [**SwapRAM**](#SwapRAM)**,** [**VerifyRAM**](#VerifyRAM),[**DoBOp**](#DoBOp)

**Function**: Primitive for transferring data from an REU

**Parameters**: [**r0**](#zr0) 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**](#zr0)**-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**](#FetchRam)moves a block of data from a REU BANK into Commodore memory.

[**FetchRAM**](#FetchRam)uses the [**DoRAMOp**](#DoRamOp)primitive by calling it with a CMD parameter of %10010001. $91

**Note**: Refer to [**DoRAMOp**](#DoRamOp)for notes and warnings.

**Example**:

**See also**: [**StashRAM**](#StashRAM)**,** [**SwapRAM**](#SwapRAM)**,** [**VerifyRAM**](#VerifyRAM)**,** [**DoRAMOp**](#DoRamOp)**,** [**MoveBData**](#MoveBData)

### **StashRAM**: (c64 v1.3+,C128) $C2C8

**Function**: Primitive for transferring data to an REU

**Parameters**: [**r0**](#zr0) 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**](#zr0)**-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**](#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**](#StashRAM)uses the DoRAMOp primitive by calling it with a CMD parameter of %10010000. $90

**Note**: Refer to [**DoRAMOp**](#DoRamOp) for notes and warnings.

**Example**:

**See also**: [**SwapRAM**](#SwapRAM)**,** [**FetchRAM**](#FetchRam)**,** [**VerifyRAM**](#VerifyRAM)**,** [**DoRAMOp**](#DoRamOp)**,** [**MoveBData**](#MoveBData)

### **SwapRAM**: (c64 v1.3+,C128) $C2CE

**Function**: Primitive for swapping data between Commodore memory and an REU.

**Parameters**: [**r0**](#zr0) 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**](#zr0)**-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**](#SwapRAM)swaps a block of data in an REU bank with a block of data in Commodore memory.

[**SwapRAM**](#SwapRAM)uses the [**DoRAMOp**](#DoRamOp) primitive by calling it with a CMD parameter

of % 10010010. $92

**Note**: Refer to [**DoRAMOp**](#DoRamOp) for notes and warnings.

**Example**:

**See also**: [**StashRAM**](#StashRAM)**,** [**FetchRAM**](#FetchRam)**,** [**VerifyRAM**](#VerifyRAM)**,** [**DoRAMOp**](#DoRamOp)**,** [**SwapBData**](#SwapBData)

### **VerifyRAM**: (c64 v1.3+,C128) $C2D1

**Function**: Verify (compare) data in main memory with data in an REU.

**Parameters**: [**r0**](#zr0) 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**](#zr0)**-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**](#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**](#VerifyRAM)uses the [**DoRAMOp**](#DoRamOp) primitive by calling it with a CMD parameter

of % 10010011. $93

**Note**: Refer to [**DoRAMOp**](#DoRamOp) for notes and warnings.

**Example**:

**See also**: [**SwapRAM**](#SwapRAM)**,** [**FetchRAM**](#FetchRam)**,** [**StashRAM**](#StashRAM)**,** [**DoRAMOp**](#DoRamOp)**,** [**VerifyBData**](#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**](#SetMsePic) C2DA Set and preshift new soft-sprite mouse picture.136

StartMouseMode $C14E Initialize the mouse 1-812

[**TempHideMouse**](#TempHideMouse) $C2D7 Hide soft-sprites before direct screen access.137

### **SetMsePic:** (C64, C128) $C184

**See also**: [**TempHideMouse**](#TempHideMouse), **HideOnlyMouse, DrawSprite**

**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**](#SetMsePic)(as opposed to a normal **DrawSprite**). [**SetMsePic**](#SetMsePic)pre-shifts the image data and lets the soft-sprite mouse routine know of the new image.

[**SetMsePic**](#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**](#SetMsePic)calls **HideOnlyMouse.**

**Note3: ARROW Equate - ARROW = $00**

**Example**:

### **TempHideMouse:** (C64, C128) $C184

**See also**: **HideOnlyMouse**

**Function**: Temporarily removes soft-sprites and the mouse pointer from the graphics screen.

**Parameters**: nothing.

**Uses**: **graphMode**

**Destroys: a, x**

**Description**:[**TempHideMouse**](#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**](#TempHideMouse) exits

immediately without affecting the hardware sprites.

**Example**:

## **print driver**

### **StartASCII:** (C64, C128) [****7912****](#h7912)

**See also**: **PrintASCII, StopPrint, StartPrint**

**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**](#StartASCII) enables ASCII text mode printing. An application calls [**StartASCII**](#StartASCII) at the beginning of each page. It assumes that **InitForPrint** has already been called to initialize the printer.

[**StartASCII**](#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**:

## process

------------------- ---- -------------------------------------------------- ---

[**BlockProcess**](#BlockProcesses) C10C Prevent a recurring timed event from running 141

[**EnableProcess**](#EnableProcess) C109 Force a recurring timed event to run 144

[**FreezeProcess**](#FreezeProcess) C112 Stop a recurring timed event's timer 142

[**InitProcesses**](#InitProcesses) C103 Initialize a table of recurring timed events 143

[**RestartProcess**](#RestartProcess) C106 Enable a recurring timed event 145

[**Sleep**](#Sleep) C199 Set up a time delay 146

[**UnblockProcess**](#UnblockProcess) C10F Allow a recurring timed event to execute 146

[**UnfreezeProcess**](#UnfreezeProcess) C115 Start a recurring timed event's timer 148

### **BlockProcess:** (C64, C128) [****C103****](#hC103)

**See also**: [**UnblockProcess**](#UnblockProcess)**,** [**FreezeProcess**](#FreezeProcess)

**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**](#BlockProcesses)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**](#UnblockProcess). [**BlockProcess**](#BlockProcesses)stops the process the **MainLoop** level. Refer to [**FreezeProcess**](#FreezeProcess)to stop the process at the Interrupt Level.

[**BlockProcess**](#BlockProcesses)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**](#BlockProcesses)has no effect.

**Example**:

SuspendClock:

ldx #CLOCK\_PROCESS ; x <- process number of the clock

jmp [**BlockProcess**](#BlockProcesses) ; block that particular process

### **FreezeProcess:** (C64, C128) ****[C112](#hC112)****

**See also**: [**UnfreezeProcess**](#UnfreezeProcess)**,** [**BlockProcess**](#BlockProcesses)

**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**](#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**](#EnableProcess). When a process is unfrozen with [**UnfreezeProcess**](#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**](#FreezeProcess)has no effect.

**Example**:

### **InitProcesses:** (C64, C128) [****C103****](#hC103)

**See also**: [**Sleep**](#Sleep)**,** [**RestartProcess**](#RestartProcess)

**Function**: Initialize and install a process data structure.

**Parameters**: a NUM\_PROC — number of processes in table (byte).

[**r0**](#zr0) **PTABLE —** pointer to process data structure to use (word).

**Returns**: [**r0**](#zr0)unchanged.

**Destroys:** a, x, y, **rl**

**Description**:[**InitProcesses**](#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**](#RestartProcess)after the call to [**InitProcesses**](#InitProcesses).

[**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**](#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**](#UnblockProcess)), use the values $00-$03. Do not call [**InitProcesses**](#InitProcesses)with a *NUM\_PROC* value of $00 or a *NUM\_PROC* value greater than **MAX\_PROCESSES** (the maximum number of processes allowable).

**Example**:

### **EnableProcess:** (C64, C128) [****C109****](#hC109)

**See also**: [**InitProcesses**](#InitProcesses)**,** [**RestartProcess**](#RestartProcess)**,** [**UnfreezeProcess**](#UnfreezeProcess)**,** [**UnblockProcess**](#UnblockProcess)

**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**](#EnableProcess)forces a process to become runnable on the next pass through **MainLoop**, independent of its timer value.

[**EnableProcess**](#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**](#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**](#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**](#RestartProcess). Any time the desired event-state is detected, a call to [**EnableProcess**](#EnableProcess)will generate an event on the next pass through MainLoop. [**EnableProcess**](#EnableProcess)can be called from Interrupt Level, which allows a condition to be detected at Interrupt Level but processed during **MainLoop.**

**Example**:

### **RestartProcess:** (C64, C128) [****C106****](#hC106)

**See also**: [**InitProcesses**](#InitProcesses)**,** [**EnableProcess**](#EnableProcess)**,** [**UnfreezeProcess**](#UnfreezeProcess)**,** [**UnblockProcess**](#UnblockProcess)**.**

**Function**: Reset a process's timer to its starting value then unblock and unfreeze the process.

**Parameters**: x PROCESS — process to restart (0 - n-l where n is the number of processes in the table) (byte).

**Returns**: [**r0**](#zr0)unchanged.

**Destroys:** a, x, y, **rl**

**Description**:[**RestartProcess**](#RestartProcess) **s**ets a process's countdown timer to its initialization value then unblocks and unfreezes it Use [**RestartProcess**](#RestartProcess)to initially start a process after a call to [**InitProcesses**](#InitProcesses)or to rewind a process to the beginning of its cycle.

**Note:** [**RestartProcess**](#RestartProcess)clears the runnable flag associated with the process, thereby losing any pending call to the process.

[**RestartProcess**](#RestartProcess)should always be used to start a process for the first time because [**InitProcesses**](#InitProcesses)leaves the value of the countdown timer in an unknown state.

**Example**:

### **Sleep:** (C64, C128) [****C199****](#hC199)

**Function**: Pause execution of a subroutine ("go to sleep") for a given time interval.

**Parameters**: [**r0**](#zr0) DELAY — number of vblanks to sleep (word),

**Returns**: does not return directly to caller (see description below).

**Destroys:** a, x, y

**Description**:[**Sleep**](#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**](#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)**.** [**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**](#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**](#Sleep) **with a jsr** [**Sleep**](#Sleep)**. The jsr places a return address on the stack and transfers the processor to the** [**Sleep**](#Sleep) **routine.**

**2:** [**Sleep**](#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**](#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**](#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**](#Sleep)**), and jsr's to this address. Because no context information is saved along with the** [**Sleep**](#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**](#Sleep)alters the normal flow of control.

**Example**: [**BeepThrice**](#exBeepThrice)

**See also**: [**InitProcesses**](#InitProcesses)**.**

### **UnblockProcess:** (C64, C128) [****C10F****](#hC10F)

**See also**: [**BlockProcess**](#BlockProcesses)**,** [**UnfreezeProcess**](#UnfreezeProcess) **,** [**EnableProcess**](#EnableProcess)**,** [**RestartProcess**](#RestartProcess)**.**

**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**](#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**](#RestartProcess) **to unblock the process.**

**Note:** If a process is not blocked, an unnecessary call to [**UnblockProcess**](#UnblockProcess)will have no effect.

**Example**:

### **UnfreezeProcess:** (C64, C128) [****C115****](#hC115)

**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**](#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**](#UnfreezeProcess) **will have no effect.**

**Note:** If a process is not frozen, a call to [**UnfreezeProcess**](#UnfreezeProcess)will have no effect.

**Example**:

**See also**: [**FreezeProcess**](#FreezeProcess), [**BlockProcess**](#BlockProcesses)

## Utility

------------------- ---- -------------------------------------------- -----

[**Bell**](#Bell) N/A Play a bell sound 150

[[**CallRoutine**](#CallRoutine)](#CallRoutine) C1D8 pseudo-subroutine call. $0000 aborts call. 151

[**CRC**](#CRC) C20E Cyclic Redundancy Check calculation. 152

[[**DoInlineReturn**](#DoInlineReturn)](#DoInlineReturn) C2A4 Return from inline subroutine. 153

[[[[**GetRandom**](#GetRandom)](#CallRoutine)](#GetRandom)](#CallRoutine) C187 Calculate new random number. 154

[[[[[[[[[[[[[[[[[[[[**ToBasic**](#ToBasic)](#CallRoutine)](#GetRandom)](#CallRoutine)](#GetRandom)](#CallRoutine)](#GetRandom)](#CallRoutine)](#ToBasic)](#CallRoutine)](#GetRandom)](#CallRoutine)](#GetRandom)](#CallRoutine)](#GetRandom)](#CallRoutine)](#GetRandom)](#CallRoutine)](#GetRandom)](#CallRoutine)  C241 Pass Control to Commodore BASIC. 155

### **Bell**: (Apple) ****[N/A](#hBELL)****

**Function**: Makes a brief beeping sound

**Parameters**: none.

**Returns**: nothing.

**Destroys: a**

**Description:** [**Bell**](#Bell) sounds a 1000 Hz signal. The sound lasts approximately 1/10th of a second.

**Note:** [**Bell**](#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**](#Bell)**:**

**PushB** [[**CPU\_DATA**](#d_CPU_DATA)](#d_CPU_DDR) **;switch to I/O space**

**LoadB** [[**CPU\_DATA**](#d_CPU_DATA)](#d_CPU_DDR)**,#**[[**IO\_IN**](#cIO_IN)](#RstrKernal)

**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**](#d_CPU_DATA)](#d_CPU_DDR) **;return to memory space**

**rts**

**See also**: [**Ddec**](#DDec)

**Example**: [**BeepThrice**](#exBeepThrice)

### ****[CallRoutine](#CallRoutine)****: (C64,C128) [****C1D8****](#hC1D8)

**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**](#CallRoutine)](#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**](#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**](#CallRoutine)](#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**](#CallRoutine)](#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**](#CallRoutine)](#CallRoutine) may be called from Interrupt Level (off of routines in **IntTopVector** and **IntBotVector**). Do not use [[**CallRoutine**](#CallRoutine)](#CallRoutine) to call inline (i\_) routines, as it will not return properly.

**See also**:

**Example**: [**HandleCommand**](#exHandleCommand):

### **CRC**: (c64,C128) [****C20E****](#hC20E)

**Function**: 16-bit cyclic redundancy check (CRC).

**Parameters**: [**r0**](#zr0)DATA **-** pointer to start of data (word).

**r1** LENGTH - of bytes to check (word).

**Returns**: **r2** CRCvalue for the specified range (word)**.**

**Destroys: a, y, r0-r3L**

**Description:** [**CRC**](#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**](#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**](#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**](#CRC)will produce the same value under all versions of GEOS.

**Note1:** 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**](#exKernal_CRC)

**MAGIC\_VALUE = $0317 ;** CRC **value that we’re looking for**

**DATA­\_SIZE=$2434 ; Size of data**

**.ramsect**

**buffer .block DATA­\_SIZE**

**.psect**

**Checksum:**

**See also**: [**Ddec**](#DDec)

**LoadW** [**r0**](#zr0)**,#buffer ;** [**r0**](#zr0) **<- data area to checksum**

**LoadW r1,#DATA­\_SIZE ; r1 <- bytes in buffer to check**

**jsr** [**CRC**](#CRC) **; r2 <-** CRC **value for data area**

**CmpWI r2,MAGIC­\_VALUE ; return status to caller**

**rts**

### [****[DoInlineReturn](#DoInlineReturn)****](#DoInlineReturn): (c64,C128) [****C2A4****](#hC2A4)

**Function**: Return from an inline subroutine.

**Parameters**: a DATABYTES — number of inline data bytes following the jsr plus one(byte).

stacktop 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**](#DoInlineReturn)](#DoInlineReturn)](#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**](#DoInlineReturn)](#DoInlineReturn)](#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**](#DoInlineReturn)](#DoInlineReturn)](#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**](#DoInlineReturn)](#DoInlineReturn)](#DoInlineReturn).

**Note:** [[[**DoInlineReturn**](#DoInlineReturn)](#DoInlineReturn)](#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**](#DoInlineReturn)](#DoInlineReturn)](#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**](#CallRoutine)](#CallRoutine) cannot be used to call an inline subroutine).

**Example**: [**i\_VerticalLine**](#exi_VerticalLine)

**See also**: [**Ddec**](#DDec)

### [[[[[****[GetRandom](#CallRoutine)****](#GetRandom)](#CallRoutine)](#GetRandom)](#CallRoutine)](#GetRandom): (C64,C128) [****C187****](#hC187)

**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**](#GetRandom)](#CallRoutine)](#GetRandom)](#CallRoutine)](#GetRandom)](#CallRoutine)](#GetRandom)](#CallRoutine)](#GetRandom)](#CallRoutine)](#GetRandom)](#CallRoutine)  produces a new pseudorandom (not truly random) number using the following linear congruential formula:

**random** = (2\*(**random**+l) // 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**](#GetRandom)](#CallRoutine)](#GetRandom)](#CallRoutine) 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**](#GetRandom)](#CallRoutine)](#GetRandom)](#CallRoutine) must be called manually.

**See also**:

**Example**:

### [[[[[[[****[ToBasic](#CallRoutine)****](#ToBasic)](#CallRoutine)](#GetRandom)](#CallRoutine)](#GetRandom)](#CallRoutine)](#GetRandom): (C64,C128) [****C187****](#hC187)

**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**](#ToBasic)](#CallRoutine)](#GetRandom)](#CallRoutine)](#GetRandom)](#CallRoutine)](#GetRandom)](#CallRoutine)](#ToBasic)](#CallRoutine)](#GetRandom)](#CallRoutine)](#GetRandom)](#CallRoutine)](#GetRandom)](#CallRoutine)](#GetRandom)](#CallRoutine)](#GetRandom)](#CallRoutine) 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**](#ToBasic)](#CallRoutine)](#GetRandom)](#CallRoutine)](#GetRandom)](#CallRoutine)](#GetRandom)](#CallRoutine) 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**](#d_CPU_DATA)](#d_CPU_DDR) to $37 (KRNL\_BAS\_IO\_IN) and jump to $C000

**Example**: [**LoadBASIC**](#exLoadBASIC)

**See also**: [[BootGeos](#BootGeos)](#BootGeos)

## text

------------------- ---- -------------------------------------------- -----

[**GetCharWidth**](#GetCharWidth) $C1C9 Get a character's width 157

[**GetNextChar**](#GetNextChar) $C2A7 Read a character from the keyboard 158

[**GetRealSize**](#GetRealSize) $C1B1 Get a character's stats 159

[**GetString**](#GetString)  C1BA Read a line of text from the user 160

InitTextPrompt $C1C0 Create the text cursor sprite -208

LoadCharSet $C1CC Load and activate a new font -212

PromptOff $C29E Turn off the text cursor -210

PromptOn $C29B Turn on the text cursor -209

PutChar $C145 Display a character -201

[**PutDecimal**](#PutDecimal) $C184 Display a 16 bit integer 163

PutString $C148 Display a text string -204

i\_PutString $C1AE Display a text string with inline data -205

UseSystemFont $C14B Select the BSW font -211

[**SmallPutChar**](#SmallPutChar) $C202 Draw a character on the screen 162

### **GetCharWidth**: (C64, C128) $C1C9

**See also**: [**GetRealSize**](#GetRealSize)

**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**](#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**](#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**](#TempHideMouse) exits

immediately without affecting the hardware sprites.

**Example**:

### **GetNextChar:** (C64, C128) $C2A7

**See also**: [**GetString**](#GetString)

**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**](#GetNextChar)removes the last character

from the queue, then the **KEYPRESS\_BIT** is cleared.

**Destroys:** x

**Description**:[**GetNextChar**](#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**:

### **GetRealSize:** (C64, C128) $C1B1

**See also**: [**GetCharWidth**](#GetCharWidth)

**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**](#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**](#GetRealSize) ; dimensions come back in x,y

### **GetString:** (C64, C128) [****C1BA****](#hC1BA)

**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**](#zr0) 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.

**rlL** 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).

**rlH** 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***](#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**](#d_string) pointer to start of string buffer**.**

any variables used by **PutChar.**

**Returns:** *from call to* [***GetString***](#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***](#GetString)*:*

[**r0**](#zr0)**-rl3, a, x, y.**

**Description**:[**GetString**](#GetString) installs a character handling routine into [**GetString**](#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**](#GetString) does:

1: Variables local to the [**GetString**](#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**](#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**](#GetString)

**LoadB** **keyData**,#CR ; load up a [Return]

lda **keyVector** ; and go through **keyVector**

ldx **keyVector**+1 ; so SystemStringService

jsr [**CallRoutine**](#CallRoutine) ; thinks it was pressed

This will also terminate the [**GetString**](#GetString) input.

**Note**: This note courtesy of Bill Coleman...Because [**GetString**](#GetString) runs off of **MainLoop**, it is a good idea to call [**GetString**](#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**](#GetString) like this:

jsr [**GetString**](#GetString)  ; Start [**GetString**](#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.

**Note2**: If the user manages to type off the end of the screen, specifically past **rightMargin**, [**GetString**](#GetString)  will stop echoing characters although it will still enter the characters into the buffer.

**See also**: PutChar, PutString, [**GetNextChar**](#GetNextChar).

### 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.

**rlH** unchanged

**Destroys: a, x, y, rlL, r2-rlO, rl2, rl3**

**Description**: [**SmallPutChar**](#SmallPutChar)is a bare bones version of PutChar. [**SmallPutChar**](#SmallPutChar)will not handle escape codes, does no margin faulting, and does not normalize the x coordinates on GEOS 128.

[**SmallPutChar**](#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**](#SmallPutChar)will only print the portion of the character lies within the margins. [**SmallPutChar**](#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 vl.4) — the entire character is clipped instead Full **leftmargin** clipping is supported on all other versions of GEOS: GEOS 64 vl.4 and above, GEOS 128 (both in 64 and 128 mode.

Like **PutChar**, 159 is the maximum CHAR value that [**SmallPutChar**](#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

### PutDecimal: (C64, C128) $C184

**Function**: Format and print a 16-bit positive integer.

**Parameters**: **a** FORMAT — formatting codes (byte) — see below.

[**r0**](#zr0)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.

**rlH** unchanged

**Destroys: a, x, y,** [**r0**](#zr0)**, rlL, r2-rlO, rl2, rl3**

**Description**:[**PutDecimal**](#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-bO 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**](#SmallPutChar)

# Wheels 4.4

## Wheels Kernal

------------------- ---- -------------------------------------------- -----

[**GetNewKernal**](#GetNewKernal) $9d80 Load New Kernal Group

[**RstrKernal**](#RstrKernal) $9d83 Unload Kernal Group

### **GetNewKernal:** (Wheels 4.4 64,128) $9D80

**See also**: [**RstrKernal**](#RstrKernal)

**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**](#GetNewKernal) allows access to the Extended Kernal available in 4.4.

If RUNFLAG is 0 [**GetNewKernal**](#GetNewKernal) behaves as a far jsrto thefirst 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**](#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**](#GetNewKernal)

### **RstrKernal:** (Wheels 4.4 64,128) $9D83

**See also**: [**GetNewKernal**](#GetNewKernal)

**Function**: Unload Extended Kernal group.

**Parameters**: none

**Destroys: a**

**Return: nothing**

**Alters:** Memory areafrom 5000-5FFF is restored to its previous contents.

**Description**:[**RstrKernal**](#RstrKernal) is used to restore the memory area 5000-5FFF after using [**GetNewKernal**](#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**](#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**](#RstrKernal) **; Remove Kernal Group, restoring 5000-5FFF ; to its previous contents.**

# **Examples**

## 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**](#exGetFPS):

jsr [**C64Model**](#exC64Model)

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**](#exC64Model):

;-- 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>

Note3: I believe this will also work on a 128 in 40 column mode. Need to test.

## math

-------------

### 8BitMultiply:

;\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

; [**8BitMultiply**](#ex8BitMultiply)**-** 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**](#ex8BitMultiply):

**MoveB** **rlL**,**r2L** ; **r2L** <- **rlL** copy of OPERAND1

ldx #**r2L** ; x <- source register address

ldy #**r1H** ; y <- destination register address

jsr [BBMult](#BBMult) ; **r2** <- **r2L** \* **r2H** do multiplication

rts

### 16x8Multiply:

;\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

; [**16x8Multiply**](#ex16x8Multiply) **- 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**](#ex16x8Multiply):

ldx #**r9** ; point to OPERAND1 in r9

**LoadB** **rlL**,#87 ; rl <- 87 (OPERAND2)

ldy #**rl** ; point to OPERAND2 in rl

jsr [Bmult](#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**](#zr0)- number to convert (in pixels)

;

; return: [**r0**](#zr0)- inches / centimeters

; **rlL** - tenths of an inch / millimeters.

; destroys: a, x, y, [**r0**](#zr0)-**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**](#exConvToUnits): ; First, convert [**r0**](#zr0)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**](#zr0)

ldy #2

jsr [**DShiftRight**](#DShiftRight)

.else

; For Centimeters, need to multiply by

; 20 1

; ------------- = ---

; 31.5 dots/cm 63

;

**LoadB** **rl**,#40 ; First multiply by 40

ldx #[**r0**](#zr0) ; (word value)

ldy #**rl** ; (byte value)

jsr [Bmult](#Bmult) ; [**r0**](#zr0)\* [**r0**](#zr0)\*40 (byte by word multiply)

**LoadW** **rl**,#63 ; then divide by 63

ldx #[**r0**](#zr0) ;

ldy #**rl** ;

jsr [[**Ddiv**](#Ddiv)](#Ddiv) ; [**r0**](#zr0)- r0/63

.endif

;-- Start of Common Code ; [**r0**](#zr0)\* result in l/20ths

**IncW** [**r0**](#zr0) ; add in one more l/20th, for rounding

**LoadW** **rl**,#20 ; now divide by 20 (to move decimal over one)

ldx #[**r0**](#zr0) ; dividend

ldy #**rl** ; divisor

jsr[[**Ddiv**](#Ddiv)](#Ddiv) ; [**r0**](#zr0)= [**r0**](#zr0)/20 ([**r0**](#zr0)= result in proper unit)

**MoveB** **r8L**,**rlL** ; rlL - l/20ths

lsr **rlL** ; and convert to l/10ths (rounded)

rts ; exit

### ­Kernal\_CRC:

;\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

; This is the actual Kernal Code for [**CRC**](#CRC).

;

; pass: [**r0**](#zr0)- pointer to start of data

; **r1** - # of bytes to check

;

; return: **r2** – [**CRC**](#CRC)Checksum

;

; destroys: a, x, y, [**r0**](#zr0), r1, r3L

;\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

[**Kernal\_CRC**](#exKernal_CRC):

ldy #$FF

sty **r2L**

STY **r2H**

iny

10$

lda #$80

sta **r3L**

20$

asl **r2L**

rol **r2H**

lda ([**r0**](#zr0)),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**](#zr0H)

50$

ldx #**r1**

**jsr** [[**Ddec**](#Ddec)](#DDec)

lda **r1L**

ora **r1H**

bne 10$

rts

**DecCounter:**

;\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

;

zCounter = $70

COUNT = $FFF0

[**DecCounter**](#exDecCounter):

**LoadW** zCounter,COUNT

10$

Jsr DoSomething

ldx #zCounter

jsr [[**Ddec**](#Ddec)](#DDec)

bne 10$

rts

### ****[Ddec](#Ddec)****vsDecW:

Size in Bytes vs Speed in Cycles of [[**Ddec**](#Ddec)](#DDec)and **DecW**

[[**Ddec**](#Ddec)](#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**](#Ddec)](#DDec)

zCounter=$70

.macro **DecW** dest

lda dest

bne dolow

dec dest+1

dolow:

dec dest

.endm

[[**Ddec**](#Ddec)](#DDec)code block.

Op Code Instruction Bytes Cycles

--------- --------------- ------ --------

A2 70 ldx #zCounter 2 2

20 0E C2 jsr [[**Ddec**](#Ddec)](#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**](#Ddec)](#DDec) ;Actual Kernal Code for [[**Ddec**](#Ddec)](#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**](#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**](#exDSmult):

lda zpage+l,x ;get sign of multiplicand (hi-byte)

eor zpage+l,y ;and compare with sign of multiplier

php ;save the result for when we come back

jsr [**Dabs**](#Dabs) ;multiplicand = abs(multiplicand)

stx r6L ;save multiplicand index

tya ;put multiplier index into x

tax ;for call to [**Dabs**](#Dabs)

jsr [**Dabs**](#Dabs) ;multiplier = abs(multiplier)

ldx r6L ;restore multiplier index

jsr [**DMult**](#DMult) ;do multiplication as if unsigned

plp ;get back sign of result

bpl 90$ ;ignore sign-change if result positive

jsr [**Dnegate**](#Dnegate) ;otherwise, make the result negative

90$

rts

## memory

-------------

[**CopyBuffer**](#exCopyBuffer)

### 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**](#exCopyBuffer)**:**

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**](#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**](#exCopyBuffer)**:**

**LoadW** [**r0**](#zr0),#SrcStr ; point to start of source String

**LoadW** **rl**,#DestBuff ; point to start of destination buffer

ldx #[**r0**](#zr0) ; x <- source register address

ldy #**rl** ; y <- destination register address

jsr [**CopyString**](#CopyString) ; DestBuff <- SrcStr (copy)

rts

### 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**](#exFind)**:**

LoadW **r2**,#NUM\_RECS ; **r2** <- total number of records

LoadW [**r0**](#zr0),#Key ; [**r0**](#zr0) <- pointer to keyword

LoadW **rl**,#Data ; **rl** <- pointer to start of search list

10$ ; DO

ldx #[**r0**](#zr0) ; **x** <- source string - key ,

ldy #**rl** ; **y** <- destination string - list

lda #REC\_SIZE ; **a** <- length of each record

jsr [**CmpFString**](#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:

[**Find**](#exFind2)**2:**

LoadW [**r0**](#zr0),#original ; [**r0**](#zr0) <- pointer to original string

LoadW **rl**,#copy ; **r1** <- pointer to copy

ldx #[**r0**](#zr0) ; **x**  <- source string =\* key

ldy #**rl** ; **y** <- destination string - list

jsr [**CmpString**](#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**](#exInitBuffers):

**LoadW** [**r0**](#zr0),#varStart ; clear variable space

**LoadW** **rl**,#(varEnd-varStart)

jsr [**ClearRam**](#ClearRam)

**LoadW** [**r0**](#zr0),#heapStart ; clear heap

**LoadW** **rl**,#(heapEnd-heapStart)

jmp [**ClearRam**](#ClearRam)

;\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

Alternate version. Using more space efficient **i\_**[**FillRam**](#FillRam)

[**InitBuffers**](#exInitBuffers)**:**

jsr **i\_**[**FillRam**](#FillRam) ; clear variable space

.word varStart

.word varEnd-varStart

.byte $AA ; With any value you choose

jsr **i\_**[**FillRam**](#FillRam) ; clear heap

.word heapStart

.word heapEnd-heapStart

.byte $00 ; Heap set to zero’s

rts

## disk

-------------

[**CheckDiskSpace**](#exCheckDiskSpace)

### 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**](#exCheckDiskSpace):

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**](#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**](#curDirHead) ; point to directory header

jsr [**CalcBlksFree**](#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**](#exDeleteDirEntry):

LoadW **r0**,rFileName

LoadW **r3**,#NullTrScTable ;pass dummy table

jmp [**FastDelFile**](#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**](#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**](#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**](#FastDelFile) to free all blocks in the file (including the file header block, which is the first entry in the table).

.endif

[**ReadAndDelete**](#exReadAndDelete)**:**

**MoveW** **r6**,**r0** ; save pointer for [**FastDelFile**](#FastDelFile)

Jsr [**FindFile**](#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**](#FastDelFile) ; file read OK, delete it!

99$

rts ; error in x

### GrabSomeBlocks:

;\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

;

; [**GrabSomeBlocks**](#exGrabSomeBlocks)— 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**](#exGrabSomeBlocks):

LoadW r2,BUF\_SIZE ; number of bytes to allocate

LoadW r6,**fileTrSecTab** ; buffer to build out table

jsr [**BlkAllOC**](#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:

;\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

;

; **My**[**FreeBlock**](#MyFreeBlock) — allocate specific block in BAM

; with any CBM device driver. And any GEOS Version

;

; pass:

; **r6L** = track #

; **r6h** = sector #

;

; **Note:**

**;** [**FreeBlock**](#FreeBlock) **was not added to the**

**; GEOS jump table until vl.3**

;\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

**My**[**FreeBlock**](#MyFreeBlock):

lda **version** ;check GEOS version number

cmp #$13 ; version Less then 1.3?

bcc 10$ ;

Jmp [**FreeBlock**](#FreeBlock) ; if not, go through jump table

10$

jsr [[**FindBAMBit**](#FindBAMBit)](#FastDelFile) ; 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**](#curDirHead),x ; flip BAM bit to make available

sta [**curDirHead**](#curDirHead),x ;

ldx **r7H** ; one more free block

inc [**curDirHead**](#curDirHead),x ;

ldx #0 ; **NO­\_ERROR**

rts

99$

ldx #[**BAD\_BAM**](#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**](#BldGDirEntry)

[[[[**GetFreeDirBlk**](#GetFreeDirBlk)](#GetFHdrlnfo)](#GetFHdrInfo)](#GetFHdrlnfo)

[**PutBlock**](#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**](#exMySetGDirEntry):

jsr [**BldGDirEntry**](#BldGDirEntry) ; build directory entry for GEOS file

jsr [[[[**GetFreeDirBlk**](#GetFreeDirBlk)](#GetFHdrlnfo)](#GetFHdrInfo)](#GetFHdrlnfo) ; 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**](#d_diskBlkBuf) for dir entry

clc

adc #[[**diskBlkBuf**](#d_diskBlkBuf) ; and get absolute address in buffer

sta **r5L**

lda #][**diskBlkBuf**](#d_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**](#d_diskBlkBuf) ; write out the new directory entry

jsr [**PutBlock**](#PutBlock)

txa ; get error status

bne 99$ ; if error, exit

clc

rts ; Success exit

99$

sec

rts ; Error exit

[**TimeStampEntry**](#exTimeStampEntry):

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:

;\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

; **My**[**PutBlock**](#MyFreeBlock) — Write [**diskBlkBuf**](#d_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**](#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 **My**[**PutBlock**](#exMyPutBlock)

**bcs 99$**

**rts ;return good status in carry**

**99$**

**... ;Error Handler or let caller handle error**

rts

**My**[**PutBlock**](#exMyPutBlock):

jsr [**EnterTurbo**](#EnterTurbo) ; go into turbo mode

txa ; check for error in X

bne 99$ ; branch if error found

jsr [**InitForIO**](#InitForIO) ; prepare for serial I/O

jsr [**WriteBlock**](#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 **Ver**[**WriteBlock**](#VerWriteBlock) ; verify block we wrote

txa ; set status flags

bne 99$ ; branch if error found

80$

jsr [**DoneWithIO**](#DoneWithIO) ; restore after I/O done

clc

rts ; No Errors

99$

jsr [**DoneWithIO**](#DoneWithIO) ; restore after I/O done

sec

rts ; Error Status exit

### MyReadBlock:

;\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

;

; **My**[**ReadBlock**](#MyFreeBlock) — Read sector from disk into [**diskBlkBuf**](#d_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 **My**[**ReadBlock**](#exMyPutBlock)

**bcs 99$**

**rts ;return good status in carry**

**99$**

**... ;Error Handler or let caller handle error**

rts

**My**[**ReadBlock**](#exMyReadBlock):

jsr [**EnterTurbo**](#EnterTurbo) ; go into turbo mode

txa ; check for error in X

bne 99$ ; branch if error found

jsr [**InitForIO**](#InitForIO) ; prepare for serial I/O

jsr [**ReadBlock**](#ReadBlock) ; primitive read block

jsr [**DoneWithIO**](#DoneWithIO) ; restore after I/O done (x is preserved in [**DoneWithIO**](#DoneWithIO))

txa ; get error result of [**ReadBlock**](#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**](#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**](#exNewAllocateBlock):

ldy [[[[**curDrive**](#d_curDrive)](#curDirHead)](#d_curDevice)](#curDirHead) ; 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**](#AllocateBlock) ; else, use driver routine

1541$

jsr [[**FindBAMBit**](#FindBAMBit)](#FastDelFile) ; 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**](#curDirHead),x ; and with BAM byte to clear

; bit and show as allocated

sta [**curDirHead**](#curDirHead), x ; and store back.

ldx **r7H** ; get base of track9s entry

dec [**curDirHead**](#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**](#exNewAllocateBlock) ; (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)+l

.psect

[**SaveRecord**](#ExSaveRecord):

LoadW [**r0**](#zr0), #filename ; pointer to filename

jsr [**OpenRecordFile**](#OpenRecord) ; open VLIR file

txa ; check open status

bne ; exit on error

lda appendPoint ; get record to append to

jsr [**PointRecord**](#PointRecord) ; go to that record

txa ; check point status

bne 99$ ; exit on error

jsr [**AppendRecord**](#AppendRecord) ; append a record at this point

LoadW **r7**, #bufStart ; point at data buffer

LoadW **r2**, #BUFLENGTH ; bytes in buffer (bufEnd-bufStart)

jsr [[[[**WriteRecord**](#WriteRecord)](#UpdateRecordFile)](#UpdateRecordFile)](#UpdateRecordFile) ; write buffer to record

txa ; get write status

bne 99$ ; exit on error

jsr [**CloseRecordFile**](#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

### RoadTrip:

;Show Leaving GEOS to use all of the resources of the machine and returning again via

;rebooting by either REU disk.

BYTESTOSAVE=$80 ; no. of bytes to save at [BootGeos](#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**](#exRoadTrip):

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](#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**](#d_CPU_DATA)](#d_CPU_DDR) ;

$200

ldx #BYTESTOSAVE ; restore bytes GEOS needs to restart

10$ ; STARTLOOP

lda GEOS\_save-l,x ; copy a byte

sta [BootGeos](#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](#BootGeos)

## graphics

-------------

[**ChangeMode**](#exChangeMode)

### ChangeMode:

GREYPAT=2

[**ChangeMode**](#exChangeMode):

jsr GreyScreen ; grey out old screen

lda **graphMode** ; switch mode by flipping

eor #%10000000 ; 40/80 bit

jsr [**SetNewMode**](#SetNewMode) **; and calling** [**SetNewMode**](#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**-l) | **DOUBLE\_W** | **ADD1\_W**

.byte **SC\_PIX\_HEIGHT**-1

.byte **NULL**

rts

### MseToCardPos:

.if **COMMENT**

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

**MseToCardPos**

converts current mouse positions to card position

pass: Nothing

uses: **mouseXPos**, **mouseYPos**

Returns: [**r0**](#zr0)**L** mouse card x-position (byte)

[**r0H**](#zr0H) mouse card y-position (byte)

Destroys: **a,x,y**

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

.endif

[**MseToCardPos**](#exMseToCardPos):

php ; save current interrupt disable status

sei ; disable interrupts so mouseXPos doesn't change\*

**MoveW** **mouseXPos**, [**r0**](#zr0) ; copy mouse x-position to zp work reg ([**r0**](#zr0))

lda **mouseYPos** ; get mouse y-position

plp ; reset interrupt status asap.

ldx #**r0** ; divide x-position ([**r0**](#zr0)) by 8

ldy #3 ; (shift right 3 times)

jsr [**DShiftRight**](#DShiftRight) ; this gives us the card x-position in [**r0**](#zr0)**L**

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**](#zr0H)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 [**r0**](#zr0)**L** it is for a different **mouseXPos** reading giving unpredictable results.

**Note3:** 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.

## icons/menu

-------------

[**ChangeMode**](#exChangeMode)

### IconsUp:

[**IconsUp**](#exIconsUp):

LoadB **dispBufferOn**,#(**ST\_WR\_FORE** | **ST\_WR\_BACK**) ;draw to both buffers

LoadW **r0**,#IconTable

jsr [[[[[[[[**DoIcons**](#DoIcons)](#UpdateRecordFile)](#UpdateRecordFile)](#UpdateRecordFile)](#WriteRecord)](#UpdateRecordFile)](#UpdateRecordFile)](#UpdateRecordFile) ;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**](#DoIcons)](#UpdateRecordFile)](#UpdateRecordFile)](#UpdateRecordFile)](#WriteRecord)](#UpdateRecordFile)](#UpdateRecordFile)](#UpdateRecordFile) ; install. Let [[[[[[[[**DoIcons**](#DoIcons)](#UpdateRecordFile)](#UpdateRecordFile)](#UpdateRecordFile)](#WriteRecord)](#UpdateRecordFile)](#UpdateRecordFile)](#UpdateRecordFile)rts

## mouse/sprite

-------------

[**ChangeMode**](#exChangeMode)

### ArrowUp:

;\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

; Put up a new mouse picture

;\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

**ArrowUp**:

LoadW [**r0**](#zr0), #DnArrow ;point at new image

jsr [**SetMsePic**](#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 %0000000000000

## utility

------------------- --------------------------------------------------------- ---

[**BeepThrice**](#exBeepThrice)Beep three times. Runs off the MainLoop by using [**Sleep**](#Sleep)

[**HandleCommand**](#exHandleCommand) 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**](#Sleep)

;\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

.if TARGET\_NTSC

FRAME­\_RATE=60

.else

FRAME­\_RATE=50

.endif

BELL\_INTERVAL = (FRAME\_RATE/10) ;approximately. 1/10 second.

[**BeepThrice**](#exBeepThrice):

jsr [**Bell**](#Bell) ; sound the bell

LoadW [**r0**](#zr0),BELL­\_INTERVAL ;

jsr [**Sleep**](#Sleep) ; pause a bit

jsr [**Bell**](#Bell) ; sound the bell again

LoadW [**r0**](#zr0),BELL­\_INTERVAL

jsr [**Sleep**](#Sleep) ; pause a bit

jmp [**Bell**](#Bell) ; sound the bell again and let bell rts

Note3: see [**GetFPS**](#exGetFPS) **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**](#exHandleCommand):

cpy #TOT\_CMDS ; check command # against last cmd num

bcs 99$ ; exit if command is invalid

lda CMDtabL,y ; get low byte of routine address

ldx CMDtabH,y ; get high byte routine address

jsr [**CallRoutine**](#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

### i\_VerticalLine:

;\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

; Inline version of VerticalLine.

; Pass:

; .word xl

; .word x2

; .byte y1

;\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

IVERT\_BYTES = 5 number of inline bytes in call

[**i\_VerticalLine**](#exi_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**](#DoInlineReturn) ; jump to inline return. Do not jsr!

### 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**](#exLoadBASIC):

**LoadW** **r6**,basicProg ; Find Basic Program to run

jsr [**FindFile**](#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**](#ToBasic)](#CallRoutine)](#GetRandom)](#CallRoutine)](#GetRandom)](#CallRoutine)](#GetRandom)](#CallRoutine)

99$

sec

rts

# **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 Dolcons 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

Publishlcon:



PUBLISHW = **PicW**

PUBLISHH = **PicH**

PUBLISHX = WRITEX + WRITEW + I\_SPACE

PUBLISHY = WRITEY

;The actual icon data structure to pass to Dolcons 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)/IESIZE ; 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**](#DoIcons)](#UpdateRecordFile)](#UpdateRecordFile)](#UpdateRecordFile)](#WriteRecord)](#UpdateRecordFile)](#UpdateRecordFile)](#UpdateRecordFile)Display and activate an icon table. |

[[[[[[[[**DoIcons**](#DoIcons)](#UpdateRecordFile)](#UpdateRecordFile)](#UpdateRecordFile)](#WriteRecord)](#UpdateRecordFile)](#UpdateRecordFile)](#UpdateRecordFile)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 Dolcons with the address of a different icon structure. In either case, the old icons are not erased from the screen by GEOS.

Dolcons 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**](#exIconsUp)

**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**.

;\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

Nolcons:

**LoadW** **r0**,#DummyIconTable ; point to dummy icon table

jmp [[[[[[[[**DoIcons**](#DoIcons)](#UpdateRecordFile)](#UpdateRecordFile)](#UpdateRecordFile)](#WriteRecord)](#UpdateRecordFile)](#UpdateRecordFile)](#UpdateRecordFile) ; install. Let [[[[[[[[**DoIcons**](#DoIcons)](#UpdateRecordFile)](#UpdateRecordFile)](#UpdateRecordFile)](#WriteRecord)](#UpdateRecordFile)](#UpdateRecordFile)](#UpdateRecordFile)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\_NVERT | $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

; \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

IconEventl:

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 **Dolcons**, the icon will not be drawn. If an icon is disabled after the call to Dolcons, 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 Dolcons 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:

# structures

## dialog/Icons/Menus/Graphics

### DIALOG:

**Note2:** 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)

**Note1:** standard window size: columns 72-263

rows 40-135

**Note1:** If the shadow pattern is zero, then no shadow is drawn.

**Note1:** Icon descriptors are stored in a table at $880C

**Note3:** Maximum # of Dialog Icons is 8. This can be worked around by drawing your own images and detecting mouse clicks over the images.

**Note1:** The following is a list of global variables stored by the window processor:

[[[[[**curPattern**](#curDirHead)](#d_curPattern)](#d_curPattern)](#d_curPattern)](#d_curPattern) **string baselineOffset** [[[[[[[[**curSetWidth**](#d_currentMode)](#curDirHead)](#curDirHead)](#curDirHead)](#d_curRecord)](#curDirHead)](#curDirHead)](#curDirHead) **curHeight** [[[[[[[[**curIndexTable**](#d_curIndexTable)](#curDirHead)](#d_curDevice)](#curDirHead)](#d_curIndexTable)](#curDirHead)](#d_curDevice)](#curDirHead) **cardDataPntr currentMode** [**dispBufferOn**](#d_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.

**Note3:** 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** as this will be a VERY large x coordinate. **DBUSERICON** Structures DO need **DOUBLE\_B** for width if the user icon is not a native 80 col icon.

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** |
| **DB**[**GetFile**](#GetFile)**S** | 16 | .byte **DB**[**GetFile**](#GetFile)**S**  .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-bO number of items in menu/sub-menu (up to **MAX\_M\_ITEMS**).

## disk

-------------

### Directory Entry:

|  |  |  |  |
| --- | --- | --- | --- |
| Offset Hex|Dec | Constant | # Bytes | Description |
| 00 | **OFF\_CFILE\_TYPE** | 1  Bit 7  Bit 6  Bit 0-2 | DOS file type  1=File Closed  Write Protect bit  **ST\_WR\_PR** %01000000  Commodore file Type  **DEL**=0  **SEQ**=1  **PRG**=2  **USR**=3 (GEOS uses USR)  **REL**=4 (Not permitted under GEOS)  **CBM**=5 |
| 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 | 19 | **OFF\_GHDR\_PTR** | 2 | track/sector of GEOS Header block |
| $15 | 21 | **OFF\_GSTRUC\_TYPE** | 1 | GEOS file structure type  **SEQUENTIAL**=0  **VLIR**=1 |
| $16 | 22 | **OFF\_GFILE\_TYPE** | 1 | GEOS file type indicator  **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 | 23 | **OFF\_YEAR** | 5 | Y/M/D/H/M |
| $1C | 28 | **OFF\_SIZE** | 2 | File Size in blocks |

# Appendex

## hardware

-------------

dialog

### 6510 data register: (64,128) 01

[[**CPU\_DATA**](#d_CPU_DATA)](#d_CPU_DDR)

;Machine Power on Default KRNL\_BAS\_IO\_IN

;GEOS Default RAM\_64K

;GEOS During serial I/O [[**IO\_IN**](#cIO_IN)](#RstrKernal)

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**](#cIO_IN)](#RstrKernal)](#cIO_IN)](#RstrKernal)

|  |
| --- |
| FFFF |
| E000 |
| D000 |
| C000 |
| A000 |
| 0100 |
| 0 |

|  |  |  |
| --- | --- | --- |
| 8k KERNAL ROM | 8K RAM | 8K RAM |
| I/O | 4K RAM | I/O |
| 4K RAM | 4K RAM | 4K RAM |
| 8K BASIC | 8K RAM | 8K RAM |
| 24K RAM | 24K RAM | 24K RAM |
| Zero Page | Zero Page | Zero Page |

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

### 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

|  |
| --- |
| Configuration Register |

;MMUReg Bits

|  |  |
| --- | --- |
| Bits | Description |
| 0 | D000-DFFF   1. I/O 2. 1 RAM or Character ROM |
| 1 | 4000-7FFF   1. BASIC ROM low 2. RAM |
| 2-3 | 8000-BFFF   1. Basic ROM 2. Internal Function ROM   10 External Function ROM  11 RAM |
| 4-5 | C000-CFFF,E000-EFFF   1. Kernal ROM 2. Internal Function ROM   10 External Function ROM  11 RAM |
| 6-7 | Bank Select   1. Bank 0 2. Bank 1   10 Bank 2  11 Bank 3 |

;- 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

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

|  |
| --- |
| RAM Configuration Register |

;MMUReg Bits

|  |  |
| --- | --- |
| Bits | Description |
| 0-1 | Size of Common Ram   1. 1k 2. 4k   10 8k  11 16K |
| 2-3 | Common Ram Location   1. Disabled 2. Bottom   10 Top  11 Both |
| 4-5 | Not Used |
| 6-7 | Bank Select for VIC   1. Bank 0   01 Bank 1  10 Bank 2  11 Bank 3 |

;- 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

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 | 400 | 6000 | 8000 | A000 | BF40 | D000 | E000 |
| [**Zero Page**](#apZeroPage) | Stack Page | deskTopVars | AppRAM | Back Screen | Disk Buffers | Fore  Screen | Kernal  Low | I/O or Kernal | Kernal  High |

### Zero Page

=================================================================================

00 [**CPU\_DDR**](#d_CPU_DDR) 6510 data direction register

01 [[[[[[**CPU\_DATA**](#d_CPU_DATA)](#d_CPU_DDR)](#CPU_DATA)](#d_curPattern)](#d_curPattern)](#d_curPattern) Built-in 6510 I/O port, bit oriented

02-21 [**r0-r15**](#pseudoregisters) GEOS Kernal zero Page pseudoregisters

22-23 [[[[[**curPattern**](#curDirHead)](#d_curPattern)](#d_curPattern)](#d_curPattern)](#d_curPattern) Pointer to fill pattern data

24-25 **string** Pointer to input buffer

;- Current Font Settings

26 **baselineOffset** Font pixels below line of print

27-28 [[[[[[[[**curSetWidth**](#d_currentMode)](#curDirHead)](#curDirHead)](#curDirHead)](#d_curRecord)](#curDirHead)](#curDirHead)](#curDirHead) Size in bytes of bit Font stream

29 **curHeight** Point size of the font

2A-2B [[[[[[[[**curIndexTable**](#d_curIndexTable)](#curDirHead)](#d_curDevice)](#curDirHead)](#d_curIndexTable)](#curDirHead)](#d_curDevice)](#curDirHead) Address of bit stream indices table

2C-2D [**curDataPntr**](#RstrKernal) Address of the first bit stream ;Font 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 **returnAddress** Pointer to click box data table

42-43 **callRouVector** Jump vector used by [**CallRoutine**](#CallRoutine)

44-45 **dlgBoxVector** [**DoDlgBox**](#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**](#d_curDevice)](#curDirHead)](#d_curDevice)](#curDirHead) **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 |
| Common RAM | BANK 2 (bank 0 if 128 is not expanded) | MMU | ROM |

Bank 3

|  |  |  |  |
| --- | --- | --- | --- |
| $0000 | $400 | $FF00 | $FF05 |
| Common RAM | BANK 3 (bank 1 if 128 is not expanded) | MMU | ROM |

Bank 14

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| $0000 | $400 | $4000 | $D000 | $E000 | $FF00 | $FF05 |
| Common RAM | 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

# **Constants**

## Zero Page

-------------

### pseudoregisters:

pseudoregisters are used when calling into the GEOS kernal. Each call will have a list of pseudoregisters to setup. Registers have common uses across the GEOS API but none are exclusively used 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**](#zr0)-**r15** that do not conflict with your current kernal interaction.

.zsect $02

**r0** .block 2 ; Pointer

**r0L** = $02 ; holds result after **DoDlgBox**

**r0H** == $03

**r1** .block 2 ; Used in RAM operations

**r1L** = $04 ; Track Number in Disk I/O

**r1H** == $05 ; Sector Number in Disk I/O, Y Position for PutChar

**r2** .block 2 ; Ptr to diskname , Buffer Size during Disk I/O

**r2L** = $06 ; Top Margin, Pixel Width, Str Length

**r2H** == $07 ; Bottom Margin, Pixel Height

**r3** .block 2 ; Left Margin, Ptr **dataFileName**

**r3L** = $08 ; Top Margin Track for Allocate Block.

**r3H** == $09 ; Bottom Margin Sector for Allocate Block

**r4** .block 2 ; Ptr to Disk Buffers, margins on boxes

**r3L** = $0A ; Sprite Number

**r3H** == $0B ; Dest Bank on Move operations.

**r5** .block 2 ; Ptr to DirEntry

**r5L** = $0C ;

**r5H** == $0D ;

**r6** .block 2 ; Ptr to T/S List for block allocates

**r6L** = $0E

**r6H** == $0F

**r7** .block 2 ; Start address of Read/Write buffer

**r7L** = $10 ; **FileType** to find with [[**FindFTypes**](#FindFTypes)](#FindFile)

**r7H** == $11 ; Number of files to get from [[**FindFTypes**](#FindFTypes)](#FindFile)

**r8** .block 2 ; Internal Kernal use during some kernal calls

**r8L** = $12

**r8H** == $13

**r9** .block 2 ; Pointer to disk structures. DirEntrys/ Info Sector etc.

**r9L** = $14

**r9H** == $15

**r10** .block 2 ; Class Pointer.

**r10L** = $16 ; Desk Top Page number

**r10H** == $17

**r11** .block 2 ; x Position for PutChar

**r11L** = $18 ; row Number in DrawPoint

**r11H** == $19

**r12** .block 2 ; Not Used by Kernel as a parameter

**r12L** = $1A

**r12H** == $1B

**r13** .block 2

**r13L** = $1C

**r13H** == $1D

**r14** .block 2 ; Not Used by Kernel as a parameter

**r14L** = $1E

**r14H** == $1F

**r15** .block 2 ; Not Used by Kernel. Commonly used in GEOS Applications

**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**](#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

# Data

## variables

### **By Name**:

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| **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  llxx xxxx if getting text input.  bit 0-5 - Counter before prompt flashes  bit 6 - Flag indicating prompt is visible  bit 7 - Flag indicating alphanumeric input is on |
| **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](#d_CPU_DDR)**](#d_CPU_DATA)](#d_CPU_DDR)](#d_CPU_DATA)**:** | 01 | 01 | 1 | **RAM\_64K** | No | Address of **6510 data register** that controls the hardware memory map of the C64. |
| [**[CPU\_DDR](#d_CPU_DDR)**](#d_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**](#d_CPU_DATA)](#d_CPU_DDR) register. This may cause unexpected results. |
| [**[curDataPntr](#RstrKernal)**](#RstrKernal): | 2C | 2C | 2 | D2DC  (BSW 9) | Yes | This is a pointer to the actual card graphic data for the current font in use. |
| [[**[curDirHead](#RstrKernal)**](#curDirHead)](#curDirHead)**[:](#RstrKernal)** | 8200 | 8200 | 256 | $00 | No | buffer containing header information for the disk in currently selected drive. |
| [[[**[curDevice](#curDirHead)**](#d_curDevice)](#curDirHead)](#d_curDevice)**:** | BA | BA | 1 | $08 | No | current serial device number. See [[[[**curDrive**](#d_curDrive)](#curDirHead)](#d_curDevice)](#curDirHead)for more information |
| [[[[[[[**[curDrive](#curDirHead)**](#d_curDrive)](#curDirHead)](#d_curDevice)](#curDirHead)](#d_curDrive)](#curDirHead)](#d_curDevice)**:** | 8489 | 8489 | 1 | $08 | No | device number of the currently active disk drive.  For Commodore, allowed values are 8 – 11. |
| [[[[[[[[[[[[[[[**[curEnable](#curDirHead)**](#d_curEnable)](#curDirHead)](#d_curDevice)](#curDirHead)](#d_curDrive)](#curDirHead)](#d_curDevice)](#curDirHead)](#d_curEnable)](#curDirHead)](#d_curDevice)](#curDirHead)](#d_curDrive)](#curDirHead)](#d_curDevice)**:** | - | 1300† | 1 | None | No | This is an image of the C64 mobenble register. |
| [[[[[[[[[[[[[[[**[currentHeight](#curDirHead)**](#d_currentHeight)](#curDirHead)](#d_curDevice)](#curDirHead)](#d_curDrive)](#curDirHead)](#d_curDevice)](#curDirHead)](#d_currentHeight)](#curDirHead)](#d_curDevice)](#curDirHead)](#d_curDrive)](#curDirHead)](#d_curDevice)**:** | 29 | 29 | 1 | $09 | Yes | card height in pixels of the current font in use. |
| [[[[[[[**[curIndexTable](#curDirHead)**](#d_curIndexTable)](#curDirHead)](#d_curDevice)](#curDirHead)](#d_curIndexTable)](#curDirHead)](#d_curDevice)**:** | 2A | 2A | 2 | D218 | Yes | pointer to the table of sizes, in bytes, of each  card in of the current font. |
| [[[[[[[[[[[[[[[**[curmobx2](#curDirHead)**](#d_curmobx2)](#curDirHead)](#d_curDevice)](#curDirHead)](#d_curDrive)](#curDirHead)](#d_curDevice)](#curDirHead)](#d_curEnable)](#curDirHead)](#d_curDevice)](#curDirHead)](#d_curDrive)](#curDirHead)](#d_curDevice)**:** | - | 1302† | 1 | None | No | Image of the C64 mobx2 register. Used for C128 soft sprites. Resides in back ram |
| **[curmoby2](#curDirHead):** | - | 1301† | 1 | None | No | Image of C64 moby2 register. Used for C128 soft sprites. Resides in back ram. |
| [**[curPattern](#curDirHead)**](#curDirHead)**:** | 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](#curDirHead)**](#d_curRecord)](#curDirHead)](#curDirHead)](#curDirHead)](#d_curRecord)](#curDirHead)](#curDirHead)**:** | 8496 | 8496 | 1 | 0 | No | Current record number for an open VLIR file.  Note: When a VLIR file is opened, using [**OpenRecordFile**](#OpenRecord). [[[[[[**curRecord**](#d_curRecord)](#curDirHead)](#curDirHead)](#curDirHead)](#curDirHead)](#curDirHead) is set to 0 if there is at least 1 record in the file, or -1 if their are no records. |
| [[[[[[[**[currentMode](#curDirHead)**](#d_currentMode)](#curDirHead)](#curDirHead)](#curDirHead)](#d_curRecord)](#curDirHead)](#curDirHead)**:** | 2E | 2E | 1 | $00 | Yes | current text drawing node. 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](#curDirHead)**](#ZZ_SET_UNDERLINE)](#curDirHead)](#curDirHead)](#curDirHead)](#d_curRecord)](#curDirHead)](#curDirHead)**:** = %10000000  b6: Bold SET\_BOLD = %01000000  b5: Reverse SET\_REVERSE = %00100000  b4: Italics SET\_ITALIC = %00010000  b3: Outline SET\_OUTLINE = %00001000  b2: Superscript SET SUPERSCRIPT = %00000100  bl: Subscript SET\_SUBSCRIPT = %00000010  b0: Unused  Clears all flags (plain text) SET\_PLAINTEXT = %00000000  Any combination of flags can be set or clear. If current node 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](#curDirHead)**](#d_currentMode)](#curDirHead)](#curDirHead)](#curDirHead)](#d_curRecord)](#curDirHead)](#curDirHead)](#curDirHead)](#d_currentMode)](#curDirHead)](#curDirHead)](#curDirHead)](#d_curRecord)](#curDirHead)](#curDirHead)**:** | 3c | 3c | 2 | $00 | Yes | Card width in pixels for the current font |
| [[[[[[[[[[[[[[[**[curType](#curDirHead)**](#d_curSetWidth)](#curDirHead)](#curDirHead)](#curDirHead)](#d_curRecord)](#curDirHead)](#curDirHead)](#curDirHead)](#d_currentMode)](#curDirHead)](#curDirHead)](#curDirHead)](#d_curRecord)](#curDirHead)](#curDirHead)**:** | 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  DRU\_1541 = 1 Drive type Commodore 1541  DRV\_1571 = 2 Drive type Commodore 1571  DRV\_1581 = 3 Drive type Commodore 1581 |
| [[[[[[[[[[[[[[[**[curXpos0](#curDirHead)**](#d_curXpos0)](#curDirHead)](#curDirHead)](#curDirHead)](#d_curRecord)](#curDirHead)](#curDirHead)](#curDirHead)](#d_currentMode)](#curDirHead)](#curDirHead)](#curDirHead)](#d_curRecord)](#curDirHead)](#curDirHead)**:** | - | 1303† | 16 | None | No | The current X positions of the C128 soft sprites. BackRAM |
| [[[[[[[[[[[[[[[**[curYposO](#curDirHead)**](#d_curXpos0)](#curDirHead)](#curDirHead)](#curDirHead)](#d_curRecord)](#curDirHead)](#curDirHead)](#curDirHead)](#d_currentMode)](#curDirHead)](#curDirHead)](#curDirHead)](#d_curRecord)](#curDirHead)](#curDirHead)**:** | - | 1313† | 8 | None | No | The current Y positions of the C128 soft sprites. BackRAM |
| [**[diskBlkBuf](#d_diskBlkBuf)**](#d_diskBlkBuf)**:** | 8000 | 8000 | 256 | $00 | No | General disk block buffer. Initialized to all zeros |
| [[[[[[[[[[[[[[[[**doRestFlag**](#d_doRestFlag)](#curDirHead)](#curDirHead)](#curDirHead)](#d_curRecord)](#curDirHead)](#curDirHead)](#curDirHead)](#d_currentMode)](#curDirHead)](#curDirHead)](#curDirHead)](#d_curRecord)](#curDirHead)](#curDirHead)](#curDirHead)**:** | - | lB54† | 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. |
| [[[[[[[[[[[[[[[**[dir2Head](#curDirHead)**](#d_dir3Head)](#curDirHead)](#d_curDevice)](#curDirHead)](#d_curDrive)](#curDirHead)](#d_curDevice)](#curDirHead)](#d_curEnable)](#curDirHead)](#d_curDevice)](#curDirHead)](#d_curDrive)](#curDirHead)](#d_curDevice)**:** | 8900 | 8900 | 256 | None | No | 1571,1581 Second BAM block |
| [[[[[[[[[[[[[[[**[dir3Head](#curDirHead)**](#d_dir3Head)](#curDirHead)](#d_curDevice)](#curDirHead)](#d_curDrive)](#curDirHead)](#d_curDevice)](#curDirHead)](#d_curEnable)](#curDirHead)](#d_curDevice)](#curDirHead)](#d_curDrive)](#curDirHead)](#d_curDevice)**:** | 9C80 | 9C80 | 256 | None | No | 1581 Third BAM block |
| dlgBoxVector: | 44 | 44 | 2 | None | No |  |

;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.