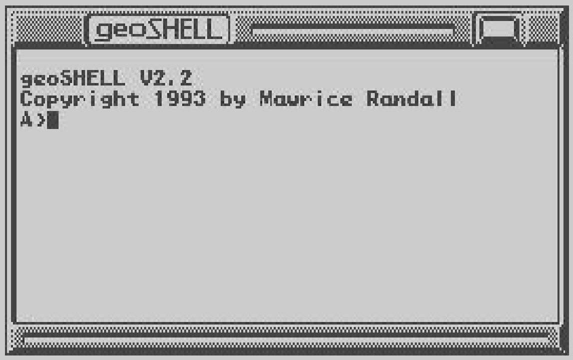
THE geoSHELL PROGRAMMER'S DEVELOPMENT PACKAGE



```
geoSHELL V2.2
Copyright 1993 by Maurice Randall
A>
```

An Aid To Creating Commands For geoSHELL

by Maurice Randall

'READ ME' file for the geoSHELL Programmer's Development Package

Thank you for your interest in geoSHELL and especially in creating commands for geoSHELL, whether it be for yourself or others also. There is a great deal of information contained in this package. If you've obtained it via the postal service, you either received it on one 1581 disk or three 1541 disks. If you downloaded it from a network or BBS, then it is most likely in three arc'd files. You can use CS-DOS, ARC 2.30, or Omega-Q to dearc the files. Each file will require one 1541 disk side.

You will find that referring to this information by viewing it in GeoWrite can be tedious. The best thing to do is to print out the entire manual so that you may have all of the information at your fingertips while working on your source code. Although, in addition to the index, you can use GeoWrite to search for a word that might lead you to the topic you need information on.

Everything included here is in GeoWrite format and was created using the COMMODORE 10 pt font. This will allow you to use the NLQ print mode of your printer for a much faster printout. The entire manual including the cover (GeoPaint file) is 194 printed pages. I've created two new commands for geoSHELL that will help you to print this manual. The command 'ppaint' will print the cover for you, and 'pwrite' will print the rest of the manual. Both of these commands are included in this package along with documentation for them. Ppaint will use your GEOS printer driver, while pwrite uses it's own built-in driver with your printer's built-in font. 64 users, make sure you copy your printer driver to disk one, or the disk that contains the cover file, gsdevpakcover. Be sure to read the documentation on pwrite and edit the file pwconfig before using it. Also, if you wish to use any codes that are in pwconfig, geoSHELL must be able to find it when you use the pwrite command.

There is an 'exec' file that you can use from geoSHELL that will handle the entire printing chore for you. Just enter...

```
exec printdevpak
```

...and follow the onscreen prompts. This exec file will also call one of two other exec files, printserial or printgeocable. If the exec file refers to a particular disk side, just make sure that it is in one of your drives. Also make sure that ppaint and pwrite are available. First the cover will be printed followed by the odd pages. Then you will be prompted to put the paper back into the printer so that the even numbered pages will now be printed on the back side of the odd sides. Once again, make sure that pwconfig can be found during the printing process if you wish to use it's added features. If necessary, copy pwconfig to your path partition, or your ramdisk, or each of the devpak disks along with the pwrite command. Feel free to modify this exec file if you need to print the manual in a slightly different manner. Or you can just print each segment one at a time with GeoWrite or pwrite.

There will be a few more files in the package that won't get printed by the exec file. These contain some routines that you may use in your own source code. There is also the sample source code files that you will use for your starting templates. The geoSHELL equate files are also included. The documentation gives full details on these files. If you wish to print any of these, just use pwrite or GeoWrite to do so.

Here is a listing of each file that is included in this package:

READ ME	What you're looking at now.
printdevpak	Exec file for printing out the manual.
printserial	Used by printdevpak.
printgeocable	Used by printdevpak.
ppaint	Command for printing GeoPaint files.
ppaint.doc	Documentation for ppaint.
pwrite	Command for printing GeoWrite files in NLQ.
pwconfig	Configure file for pwrite.
pwrite.doc	Documentation for pwrite.
gsdevpakcover	The manual cover in GeoPaint format.
gsdevpak1	Section 1 of the manual.
gsdevpak2	Section2, pages 1 through 31.
gsdevpak3	Section2, pages 32 through 61.
gsdevpak4	Section2, pages 62 through 90.
gsdevpak5	Section2, pages 91 through 130.
gsappendix1	Appendix A of the manual.
gsappendix2	Appendix B.
gsappendix3	Appendix C.
gsappendix4	Appendix D.
gsdevindex	The index for the manual.
MiscSource	Miscellaneous source code.
GSVariables	Assemble this to create GSVariables.rel.
GSVars1	geoSHELL variables. (.included in GSVariables)
GSVars2	geoSHELL variables. (.included in GSVariables)
ShellEquates	geoSHELL equates. (.included in GSVariables)
GSVariables.rel	Always put this in your link file.
Sample.lnk	Template for your link files.
SampleHdr	Template for your header files.
Sample	Template for your source code files.
Color80.lnk	Link file for color80 command.
Color80Hdr	Header file for color80 command.
Color80src	Source code for color80 command.

If you obtained this package in the form of three arc'd files, they are as follows:

gs1dvpak.arc	Disk 1, the first 12 files.
gs2dvpak.arc	Disk 2, the next 3 files.
gs3dvpak.arc	Disk 3, the remaining files.

These files will dearc to three 1541 disk sides. Or all three files will dearc to a single 1581 disk. Since you will use GeoWrite for your text editor when creating your source code, you will be able to easily cut and paste any of the included example routines that you will find in this package. If you should encounter any problems, please let me know. Thanks, and good luck.

Maurice Randall
P.O. Box 606
Charlotte MI 48813
PH: (517) 543-5202

I may also be contacted on GEnie as M.RANDALL2

```
clr
echo If you are using a tractor feed printer, set the perforation at
about 1/4" above the printhead^
echo Are you using a geoCable or the serial port?^
echo Hit 'g' for geoCable or 'p' for serial port.^
getkey
{p }
echo Which drive contains Disk 1? a, b, c, or d^
echo Press q to quit^
getkey
{a a: ppaint gsdevpakcover^ }
{b b: ppaint gsdevpakcover^ }
{c c: ppaint gsdevpakcover^ }
{d d: ppaint gsdevpakcover^ }
exec printserial^

{g }
echo Which drive contains Disk 1? a, b, c, or d^
echo Press q to quit^
getkey
{a a: ppaint gsdevpakcover^ }
{b b: ppaint gsdevpakcover^ }
{c c: ppaint gsdevpakcover^ }
{d d: ppaint gsdevpakcover^ }
exec printgeocable^

{q end }
```

```
pwrite op gsdevpak1^
pwrite op gsdevpak2^
clr
echo Place disk two in any drive and hit s to start...^
echo ...or q to quit^
getkey
{s clr pwrite op gsdevpak3^
pwrite op gsdevpak4^
pwrite op gsdevpak5^ }
clr
echo Place disk three in any drive and hit s to start...^
echo ...or q to quit^
getkey
{s clr pwrite op gsappendix1^
pwrite op gsappendix2^
pwrite op gsappendix3^
pwrite op gsappendix4^
pwrite op gsdevindex^ }
clr
echo Re-insert the paper so that the printing will occur on the back side
beginning with the cover...^
echo Place disk one in any drive and hit s to start...^
echo ...or q to quit^
getkey
{s clr pwrite ep gsdevpak1^
pwrite ep gsdevpak2^ }
clr
echo Place disk two in any drive and hit s to start...^
echo ...or q to quit^
getkey
{s clr pwrite ep gsdevpak3^
pwrite ep gsdevpak4^
pwrite ep gsdevpak5^ }
clr
echo Place disk three in any drive and hit s to start...^
echo ...or q to quit^
getkey
{s clr pwrite ep gsappendix1^
pwrite ep gsappendix2^
pwrite ep gsappendix3^
pwrite ep gsappendix4^
pwrite ep gsdevindex^ }
clr
      echo Printing is finished!^
echo You can now assemble your manual into a folder.^
end
{q end }
```

```
pwrite og gsdevpak1^
pwrite og gsdevpak2^
clr
echo Place disk two in any drive and hit s to start...^
echo ...or q to quit^
getkey
{s clr pwrite og gsdevpak3^
pwrite og gsdevpak4^
pwrite og gsdevpak5^ }
clr
echo Place disk three in any drive and hit s to start...^
echo ...or q to quit^
getkey
{s clr pwrite og gsappendix1^
pwrite og gsappendix2^
pwrite og gsappendix3^
pwrite og gsappendix4^
pwrite og gsdevindex^ }
clr
echo Re-insert the paper so that the printing will occur on the back side
beginning with the cover...^
echo Place disk one in any drive and hit s to start...^
echo ...or q to quit^
getkey
{s clr pwrite eg gsdevpak1^
pwrite eg gsdevpak2^ }
clr
echo Place disk two in any drive and hit s to start...^
echo ...or q to quit^
getkey
{s clr pwrite eg gsdevpak3^
pwrite eg gsdevpak4^
pwrite eg gsdevpak5^ }
clr
echo Place disk three in any drive and hit s to start...^
echo ...or q to quit^
getkey
{s clr pwrite eg gsappendix1^
pwrite eg gsappendix2^
pwrite eg gsappendix3^
pwrite eg gsappendix4^
pwrite eg gsdevindex^ }
clr
      echo Printing is finished!^
echo You can now assemble your manual into a folder.^
end
{q end }
```

geoSHELL Command Documentation
Copyright 1993 by Maurice Randall
(requires geoSHELL V2.2 or greater)

ppaint   (external transient)

This command will print a GeoPaint document without having to load up GeoPaint. You can now print your GeoPaint files right from geoSHELL.

Example: ppaint filename

This will print a GeoPaint file from the currently active drive. That's all there is to it.
The standard GEOS printer drivers are used to do the printing.
You can use the STOP key to halt the printing in case the document does not look like you want it to. For some of the multi-pass printer drivers, you may have to hold the key down for several seconds until it responds.

MAKING MULTIPLE COPIES

By using geoSHELL's 'exec' command, you can print a number of GeoPaint files all at once and walk away from the computer while the printing is taking place. Just create a GeoWrite file with the necessary commands in it and execute the file with the 'exec' command. Refer to your geoSHELL manual for more on the exec command.

```
open 27 64^
close 27 64^
boldon 27 71^
boldoff 27 72^
italicon 27 52^
italicoff 27 53^
underlineon 27 45 1^
underlineoff 27 45 0^
subscripton 27 83 1^
subscriptoff 27 84^
superscripton 27 83 0^
superscriptoff 27 84^
```

The codes listed on page one will be used by pwrite when it loads in this configure file. These are the most common ones supported by most Epson mode printers. If these codes do not match the ones supported by your printer, simply change them to the correct ones. If your printer does not support a certain style change, delete that particular one altogether and that particular style change will simply be ignored when printing.
If you are using a 1525 printer, or any other printer with limited capabilities that emulates the 1525, delete the codes from page one and cut and paste the following codes in their place if you want to try these features. Or, you could just use the 'open' string of codes to ensure that the printer is in the correct mode. This will let you do some special things with your printer. Wherever bold print is, you will have double-wide characters useful for headlines or titles. Limit the length of your lines when using bold in this case. Also with these codes, whenever you use italics, you will now have reverse-printing. The open string makes sure that your printer is operating in upper/lower case mode with other settings turned off. The close string does the same thing.

```
open 17 15 146^
close 17 15 146^
boldon 14^
boldoff 15^
italicon 18^
italicoff 146^
```

The following set of codes works on a STAR NX1000 and may also work on other printers. This will set up the printer to use the COURIER font.

```
open 27 64 27 107 0 27 120 49^
close 27 64^
boldon 27 71^
boldoff 27 72^
italicon 27 52^
italicoff 27 53^
underlineon 27 45 1^
underlineoff 27 45 0^
subscripton 27 83 1^
subscriptoff 27 84^
superscripton 27 83 0^
superscriptoff 27 84^
```

pwrite   (external transient)

Once you've used this command a few times, you will love GeoWrite. This gives you the printing capabilites that GeoWrite left out. From the very beginning, GeoWrite was always intended to use your printer's graphic mode. This allows almost any style of font to be used. GeoWrite excels at this type of printing. Some of us still like using our printer's own built-in fonts, however. Yes, GeoWrite has a mode that let's you print in Draft or NLQ. That will use your printer's own font. But here's the catch, all style changes are ignored! Enter 'pwrite'. All bold, italic, underline, subscript and superscript style changes are supported. Only the outline style is ignored since graphic printing must be used in order to support it. So, do not use outline in your document or it will throw the layout off.
This command also supports PAGE, DATE, and TIME in the headers and footers. The title page setting is also supported. Double-spacing is handled as well. But, 1 1/2 spacing is not, since GeoWrite does not format the page properly for this to work on all printers. And not all printers can print at 4 lines per inch. But to do 3 lines per inch only requires an extra linefeed.
Now you can have your cake and eat it too. Use GeoWrite for all your graphic-type printing and use pwrite for all of your NLQ printing. When you create your GeoWrite document, you must use the COMMODORE 10 pt font throughout for proper formatting, since pwrite expects 80 characters per line and 66 lines per page. There will actually be 62 printed lines with 4 lines of nothing in order to skip over the perforation. This is how GeoWrite formats the document, so pwrite does it the same way. In fact, whatever GeoWrite does on the screen, pwrite will do on the printer, as long as the COMMODORE font is used.
Pwrite does not check the font that is in use though. It will ignore any font change. But if you don't use the COMMODORE font, your results may not be what you expect. By using this font, GeoWrite will allocate 12 pixels for each line, giving you 6 lines per inch and so each page on the screen will look just like what comes out on the paper.

THE CONFIGURE FILE

Here is another plus for pwrite. Before printing begins, it will look for a file called pwconfig. If it finds it, it will use the printer codes it finds in the file. Since pwrite does not use your normal GEOS printer driver, it must know how your printer works. This is what the pwconfig file is for. This contains the codes for turning on the style changes and turning them off. If pwconfig is not found, then all style changes will be ignored. You can also leave out any style change codes that you wish to have ignored when printing your document. Delete the ones that your printer does not support, or change them to any setting that you would like activated.
There is also a series of codes that can be sent at the start of a document and another at the end. Any codes following 'open' will be sent at the beginning of your document. This is useful for setting up your

printer into any special configuration you want, such as the desired font selection. Then, when printing is finished, the codes following 'close' will be sent. You can use this to reset your printer back to it's default settings, or whatever you desire. The default codes that are supplied will send the normal reset sequence of 27 64. This is what most Epson mode printers use. You are free to change this any way you want.
All of the other settings should be self-explanatory. For instance 'boldon' will be used whenever the style change for bold is encountered. And likewise, 'boldoff' when non-bold text is once again encountered. The settings that are supplied are used on most Epson mode printers. Each of these settings will accept up to 20 codes. Each string of codes must be terminated with an UP-ARROW character. There must be one space separating each code. The basic rules for any command in geoSHELL is followed here. Just think of these as commands that only pwrite recognizes.

PRINTING YOUR DOCUMENTS

When you are ready to print, you must decide which pages you desire to print, and how your printer is connected. Here is one example:

```
pwrite ap filename
```

If you are familiar with geoSHELL, it's obvious what the filename does. But the parameter before the filename, 'ap', tells pwrite to print 'all' pages and send them to a printer or interface connected to the serial port.
You can see that the 'p' is used just like with any other print command in geoSHELL. Likewise if you are using a geoCable, substitute a 'g' for the 'p'. Instead of using the 'a' to print all the pages, you can use an 'o' or an 'e'. These would select either 'odd' or 'even' pages. This makes it handy for printing double-sided sheets. The only thing you must remember is how your pages are numbered. The first page can start with an even number if you told GeoWrite to do so. Keep this in mind.
For those times when you need to print just one or a few pages out of your document, you can tell pwrite to do so with a slightly different parameter arrangement. You still use the ap, ag, ep, or whatever, but you can also include a range of pages just before this. Here is an example:

```
pwrite 12 24 ap filename
```

This will print all pages from 12 through 24 of your document. If you used an 'o' instead of the 'a', then only the odd pages between those two would get printed. The pages that get printed are determined by how you have them numbered in the document. If your first page starts out as page number 36, then the previous example would print nothing. Let's say you wanted to print just page 10. The following example would do this:

```
pwrite 10 10 ap filename
```

You always select the starting page and the ending page when you use this parameter. Just remember that this is optional and you can leave it out if you wish to print the entire document.
There might be an occasion when you need to print your pages in the other direction. If you were to put a higher number first and a lower number

second in your parameter, then the higher numbered page would get printed first, followed by the next page before it and so on until the second page specified is printed. Here's an example:

```
pwrite 35 30 ap filename
```

This will print the pages from 35 through 30 in reverse order.

HINTS AND TIPS

As with anything, the best way to learn anything is to just start doing it. But, here's some things you might want to remember when you are creating your document.
Unless you are experimenting with something a little different, be sure that the COMMODORE font is selected throughout your document. It is very easy for the BSW font to be mistaken here or there when viewing it on the screen. Watch your headers and footers. Make sure the correct font is chosen here also. It is more critical here. If the wrong font is used in the headers and/or footers, some of the data might not print out because of the space allowed. The BSW font does not require as much vertical space, and so all of the lines contained within might not print out.
Speaking of headers and footers, these can be very useful. GeoWrite does not provide any means of controlling the top and bottom margins unless you use page breaks. But by using a header or footer with some carriage returns, GeoWrite will allocate a blank line for each carriage return. This let's you control the amount of blank space as each page skips over the perforation.
Headers and footers work just a little different here. When using a header, the content of your page will begin at the very next line following your header. But when you use a footer, GeoWrite inserts one blank line before printing the footer. So, this will cause printing to go to the 63rd line instead of stopping at the 62nd line. So, if you want printing to begin two lines down, just hit two carriage returns in a blank header. Or if you want a bigger margin at the bottom of the page, add carriage returns to a footer. Just count one extra space for the blank one that is inserted by GeoWrite. Pwrite will follow the same format that GeoWrite does here.
Don't worry if this sounds confusing, pwrite will format your page properly. Whatever you see on the screen will be what comes out on paper. But use that COMMODORE font or it might not. Even if you put just carriage returns in a header or footer, make sure that those carriage returns have the COMMODORE font assigned to them for proper line spacing. Actually, if you don't like the looks of the COMMODORE font on screen, you can substitute any other font as long as it's dimensions in height and width are exactly the same.
When you set up or modify the pwconfig file, you can put anything you want on any page except page one. Only page one is used by pwrite. This way, you can store different setups on other pages and just cut and paste them to page one when needed. You can also keep notes on the additional pages for your own use. If you use a printer with limited capabilites, such as a 1525, check out page 3 of the supplied pwconfig file. If you are using a CMD device and have assigned a partition with the path command, pwrite will also search the path partition for the pwconfig file. When it searches for your document, it will search all drives

except the path partition. You don't have to have the document on the currently active drive. This allows large works that are spread across multiple files to be more easily printed without having to select the correct drive first. Just make sure that pwrite doesn't find the wrong one if you have more than one copy of the same name visible on your system.
Since pwrite doesn't care what codes you use for each of the style changes, you might be able to do some special stuff. Let's say for example that you don't really need underlining in your document, but you want to use two different fonts. Just change the underlineon and underlineoff to select a different font. Then whenever you want to use this font, underline your text in GeoWrite. You can have up to 20 codes in each string. This should be more than enough to do some pretty fancy stuff. Use your imagination.
In fact, if you are careful in planning, you could have your printer do double-wide and double-high printing. If you do double-wide, limit your line length with carriage returns. If you do double-high or anything else that would print larger than 6 lines per inch, use a page break to limit the length of the page. You might find that you will have to print one page at a time, in some cases when doing special things. Double-wide is handy for headlines.
You can also print columns easily now by using both pcode (or gcode) and pwrite. Start by setting the left margin in your document all the way to the left and the right margin just slightly to the left of the middle on each page. Then begin by printing just the odd pages. When finished, use pcode to send the code to your printer that will move it's left margin over to the center of the page. Put your paper back into the printer and use pwrite to print the even pages. The second column will now print out. Or, if your printer supports returning to the top of form, you can set up an exec file to do it all in one step. The following example works on a STAR NX-1000 and any other printer that supports the CHR$(27)CHR$(12) sequence:

```
pcode 27 64^ {tell the printer this is the top}
pcode 27 67 70^ {make the page length longer}
pcode 27 108 0^ {left margin to zero}
pwrite 1 1 ap filename^ {first column}
pcode 27 12^ {return to top}
pcode 27 108 40^ {left margin to center}
pwrite 2 2 ap filename^ {second column}
{now do the next page}
pcode 27 64^ {tell the printer this is the top}
pcode 27 67 70^ {make the page length longer}
pcode 27 108 0^ {left margin to zero}
pwrite 3 3 ap filename^ {first column}
pcode 27 12^ {return to top}
pcode 27 108 40^ {left margin to center}
pwrite 4 4 ap filename^ {second column}
```

...and so on....

If you have included any photo scraps in your document, these will be ignored by pwrite. However, the correct amount of blank space will be inserted for the graphic. This allows you to put the paper back in the printer and do one of two things. Using a copy of your document, you can delete all of the text lines on the page and enter carriage returns in their place. And then use GeoWrite to print the graphic on your page. Or you can convert the GeoWrite document to a GeoPaint and then erase all the text and let GeoPaint or the 'ppaint' command print the graphic. With a little thought you will actually have more control here. You can place the graphic wherever you want on the page.

If you have a printer that is wide enough to take a page sideways, you could print left and right pages and then fold them into a 5.5 x 8.5 inch booklet. Just use the codes in the 'open' string to control the size of your font and the lines per inch. A condensed elite setting works pretty good here. Also use margin settings in GeoWrite to help with this.

As you use this command, you will discover many tricks that you didn't think were possible.

THE geoSHELL PROGRAMMER'S DEVELOPMENT PACKAGE

TABLE OF CONTENTS

AN INTRODUCTION TO THIS MANUAL

This manual is a part of a new development package for developing commands to be used with geoSHELL, the command line interface for GEOS. This manual isn't going to explain much about geoSHELL from the user's point of view, the geoSHELL User's Manual does that. And I would assume that if you are planning on writing commands for geoSHELL, that you more than likely are already somewhat familiar with the basic operation of geoSHELL.
In order to create a new command for geoSHELL, you should be familiar with machine language programming in GEOS and likewise will need the GEOPROGRAMMER package that was developed by Berkeley Softworks (now known as GEOWORKS) and is now available from CMD Sales, a division of Creative Micro Designs. All of the source code included here and all of the references to macros and variables is compatible only with GeoProgrammer and as such is in GeoWrite format. From this point on, it will be assumed that you do indeed have the GEOPROGRAMMER package and are familiar with using it to write programs for GEOS.
I put a great deal of effort into making geoSHELL easy to use and understand and I hope that throughout this manual and accompanying files, that I have achieved the same results by making it easy for you to become familiar with the inner workings of geoSHELL, and have supplied enough material, so that you might be able to use the information contained here to write new commands for yourself and others to use and enjoy.
Throughout this manual, there are programming examples, and when this document is printed on your printer, some of the characters may not print as they will appear in GeoWrite, depending on how your printer is configured. Since you will be printing this document using your printer's own built-in fonts, here is a list of the characters that are most likely to look different than what they really are:

^	The UP-ARROW
|	The CMDR/UP-ARROW
{	A LEFT CURLY-BRACE
}	A RIGHT CURLY-BRACE
#	The POUND sign
[	A LEFT BRACKET
]	A RIGHT BRACKET
_	An UNDERLINE character
<	A LESS-THAN character
>	A GREATER-THAN character

If none of these characters appear as described, you will have to remember this as you read through the document, or check your printer manual and make the needed changes to your printer. If they look correct, then you are all set. The only other way out is to painstakingly print this document using GeoWrite's 'HIGH' print setting, which uses the graphic mode of your printer. There are a lot of pages, this will take a great deal of time.
While reading through this manual, think about theterminology that is used. Some of my methods may not be the same as yours or anybody else's for that matter, but I think that I have put it in words that might be

fairly easy to follow and understand. There are references to the current drive or currently active drive here and there. Usually, the currently active drive is the one that the user thinks is the current drive, the one represented by the drive letter that is displayed at the left of the command line. A reference to the current drive might be the one that is currently being accessed, however. Hopefully you will make this distinction. The variable PRESDRIVE always contains the device number of the currently active drive that the users sees. Whereas, the current drive would be contained in curDrive.
I have put this development package into the public domain and it may be freely distributed by anyone as long as it is not sold for any amount of money for profit. The only fee that may be charged for it's distribution would be enough to cover the cost of copying and mailing. This allows registered Commodore user groups to put the package into their library and charge their members the normal library disk fee to obtain it. Any individual may privately give out copies of this package as long as there is no charge to do so. Any other person or group or organization must obtain written permission from myself, the holder of the copyright, before being allowed to distribute it in any form. It is also assumed that anyone using this package is also an owner of an original geoSHELL disk and manual and is registered as such. For those who aren't, it is assumed that you are not using this package as an aid in creating commands for geoSHELL, but possibly as an aid in creating program's of other types for GEOS in general. Please feel free to use the information contained here if it helps to keep the GEOS format alive. But you will find that in most part, this package pertains mainly to geoSHELL.
With that out of the way, let's get down to business. Read through this documentation and study some of the sample source code that is included. Look through the section of this manual that covers each of the accessible routines in geoSHELL and try to imagine what you might be able to do with some of those routines. Think of a useful command that you wish geoSHELL had, and then create that command. If you come up with something useful, upload it to some of the major networks and share it with other geoSHELL owners or send it to me so that I can include it with other commands that I will create for geoSHELL owners to use. Anything that you create with this package should be kept as public domain. The only commercial part of geoSHELL is the main program and it's accompanying manual.
I hope this package is helpful, and please write to me with any problems or further questions that this package does not answer for you. If you create something useful, let me know.

Maurice Randall
P.O. Box 606
Charlotte MI 48813

PH: (517) 543-5202 (daytime phone and answering machine)
Using a P.O. Box, I will always be able to receive your mail, since my residence address could possibly change. If you are ever in the Charlotte, Michigan area though, stop in and say hi at my shop during the daytime hours at:

Automotive Performance
426 Sumpter St.

Charlotte MI 48813

There is no shareware fee involved with using this package. You may use it free of charge. Being a registered owner of geoSHELL entitles you to use this package. Just write to me and let me know that you are using it so that if I discover any errors or create any updates, I can keep you informed as such. By the same token, please let me know if you find any errors anywhere. I've done considerable proofreading of this entire document and rechecked it with the original source code for geoSHELL. But, I'm just human also. I think, though, that you will find this to be quite error-free and I hope that you can make good use of it.

A QUICK TUTORIAL

Included with this package are three files that are the very most basic templates that you can have when you start to create a geoSHELL command. They are appropriately named Sample, SampleHdr, and Sample.lnk. Copy these three files along with GSVariables.rel to either a separate work disk or your ram disk. Don't use the original files, use copies. You are about to create your very first geoSHELL command. Be sure that you also have GEOASSEMBLER and GEOLINKER on your disk. You can split these files between drives A and B if you'd like. The other files you will need were included with your GEOPROGRAMMER package. They are geosSym and geosMac. Copy these to drive A or B also. And don't forget GEOWRITE, since that is the text editor that we use.

Start by loading 'Sample' into GeoWrite and study the contents of this file. In it you will see where you would add the code that makes up your command. The very first instruction that will be in this file, where the label 'Sample' is found is where geoSHELL will jump to when it executes your command. From this point on, until you reach the instruction 'jmp ExitCommand', your command will be in control of the machine.

You can have your command do most anything you can think of, within reason, and within the limitations of your machine and the limitations imposed by the memory given to you. We won't worry about any of this for now, because this first command will be very small and will work just fine, hopefully on the first try.

Where the label, 'Sample' is, type in the following code exactly like you see it here:

```
Sample:
      LoadW r0,#nameText     ;point r0 at some text.
      lda   #(IN_ONE|TR_CR)  ;formatting info.
      jsr   OtherMessage     ;display a message.
      jmp   ExitCommand ;quit the command.

nameText:
      .byte "John Doe",0     ;use your name instead.
```

Now load up GEOASSEMBLER. If you're using geoSHELL, just type the hotkeys, 'as', to run it. Assemble the file SampleHdr and then assemble Sample. If everything was done properly, these two files should have assembled without any errors.

Exit out of GEOASSEMBLER and load up GEOLINKER. With geoSHELL, type the hotkeys, 'li', to run it. Now, from GEOLINKER's menu, select the file

'Sample.lnk' and proceed to finish creating your first command. Once again, if everything was done properly, GEOLINKER will finish with no errors and you will have a file on your disk called 'sample'.
If you came up with any errors while assembling or linking, you will have to go back and check the files for errors. When you've corrected the problem, try assembling and linking again.
Now that you have a new command called 'sample', try it out. Just enter the command's name from geoSHELL and see what it does. If it works as expected, you will see your name displayed withinthe geoSHELL window.
What your command did was to access the built-in routine OtherMessage to display the text string that r0 was pointing at. Just before calling the routine, you loaded the accumulator with a value that told OtherMessage how you wanted the text displayed. In this case, we or'd #IN_ONE with #TR_CR. IN_ONE tells OtherMessage to indent one character, and TR_CR tells it to add a trailing carriage return after displaying the message.
When you are ready to go further, take these same sample files and rename them appropriately to reflect the command you are creating. Be sure to go into the SampleHdr file and change the areas that should be altered, such as your name should go where the author's name is. You can also add a message in the Hdr file that will show up in the Desktop's info box or with geoSHELL's info command. Load the .lnk file into GeoWrite and change the names of the files contained in it. Always leave the file GSVariables.rel as the first file to link and you will have access to all of the variables that are accessible in geoSHELL. GSVariables does not add anything to the size of your code, it only gives access to the variables and equates.
Read through this manual and check out all of the features of geoSHELL that are available to you and then think of some ideas for some new commands.

THE geoSHELL STYLE

When you create your commands for geoSHELL, try to stick to the style that everyone is accustomed to. A command's name should be all lower-case letters. It's easier to type when you don't have to hold down the shift key. Keep any required parameters simple and easy to remember. If your command requires a destination of some sort, such as a disk drive or printer, make the requirement one letter, if possible, followed by a space and then a filename if it is required.
You are actually free to do what you want, but people get used to doing things a certain way. Keep that in mind. It is hard for some people to get the hang of using geoSHELL after becoming familiar with the Desktop. But once they have used geoSHELL for awhile, they find out just how easy and powerful it really is. Keep this in mind if you try to change the style that geoSHELL users are now familiar with.
geoSHELL comes with a fairly complete online help file and a help command for accessing it. But since new commands are not included in the help file, it would be a good idea to display a short bit of information on the screen when the user enters the wrong parameter after your command. If no parameters are required, then there would be no need for this. Your command would just do it's thing and exit. You would still want to have error messages displayed when your command detects an error, however. This manual will explain how to display messages on the geoSHELL window. There are also examples in the sample source code that you can cut and

paste into your own source code. That will make it easier to get it right the first time.
Your command, if you decide to share it with others should also include complete documentation. Don't make everyone try to figure out how to use it or exactly what it does. Explain everything, and in a manner that everyone can understand. Even the most expert of people prefer simple instructions. Put your documentation in GeoWrite format. If you create it with a 64, the 128 can still read it on the screen, since a 128 will display the full page width. But if you create the documentation on a 128, think of the 64 users also. Remember that they can only display half of the page width. So, it is acceptable to format the documentation with the right-hand margin set at 4.1 inches. Set the left margin all the way to the left. 64 users should select a 'full page wide' from the menus in order to get the left margin set at .2 inch, thereby making the document a V2.1 file. If a 128 user loads a V2.0 file into GeoWrite, the first thing he is asked is if he wishes to convert it to 2.1. This can be alarming sometimes or annoying if reading the original doc file on a write-protected disk. By keeping the right margin at 4.1 inches, a 64 user won't have to scroll to the right as he is reading the text. If the user wishes to print out the documentation, he is free to reformat the text to any margin settings he desires, but for just reading on the screen, it is very disturbing to have to scroll back and forth.
Once you have a full working command with absolutely no bugs and wish to share it with others, use one of the convert programs (2.5 compatible) or the new convert command for geoSHELL toconvert the command and the documentation to Commodore format and then use one of the arc utilities to arc the files together. This way the command and it's docs stay together when uploading and downloading. Anyone that does any amount of downloading from a BBS or major network should have the utilities to unarc a file. Of course, an easier way is to create a self-dissolving arc file. Then the user merely has to load and run the file to dissolve and then use convert to get the files back into GEOS format.
Always think of the person that is going to use your command. Keep in mind also the many different combinations of hardware that is possible with GEOS. There are many different drive setups that can be used and many different printers. Try to be compatible with everybody. If you are creating a command for yourself, then just do what only you need. But if you are going to share with someone else, keep this in mind. If you stick with the proven routines in geoSHELL, then your command should be compatible with any of the drives that are available.
Also, try to decide if your command will work with both the 64 and 128. If it is specific to the 128's 80 column mode, be sure to check the system that is being used. If you put the correct identification byte in the command's header block, then geoSHELL will do the checking for you before your command is loaded. But there is one catch to this. When a command is used once, it will stay in memory until another transient command overwrites it. If the machine is a 128 and your command only works in 80 column mode, for instance, geoSHELL will, of course, allow the command to execute if the user is currently in 80 column mode. But, once the command is in memory, it is possible that the user could switch to 40 columns and enter the command again. And if your command does nasty stuff in 40 column mode or vice-versa, that wouldn't be good. So, it is a good idea to test the screen mode before continuing with your command in these cases.

geoSHELL provides a variable to check that will tell you exactly the machine and the mode it is running in. The variable 'screenmode' is used for this. If bit 6 is set, the machine is a 64. If bit 7 is set, it is a 128 in 80 column mode. If screenmode equals 0, then the machine is a 128 in 40 column mode. In addition to this, the variable 'videomode' will tell you if the 128's 80 column screen is in multi-color mode or monochrome mode, in case your command needs to know. If bit 7 of videomode is set, the 128's 80 column screen is using color. This can be tested even while in 40 column mode, but is not effective unless the user switches to 80 columns.
Most commands will not have any reason to erase the geoSHELL window from the screen. It is a good idea to try and keep the window active, if at all possible. When your command exits, the user needs the window to be there in order to continue with something else. Of course, geoSHELL has routines that you can call to restore the window and whatever was in it. But when you erase it and do something different, there will be some routines that you won't want to access. The reason is that some geoSHELL routines will cause your command to cease if certain errors are encountered. If that happens and the window is not there, then the user just gets a blinking cursor without the window. He willstill be able to type in a command, but the effect might scare him. There are only two commands that are supplied with geoSHELL that operate without the geoSHELL window. One is 'dcopy' when used with a 40 column screen, and the other is 'color80' with the 80 column screen of the 128. dcopy uses the computer's screen memory area for a buffer when transferring data from one disk to another. While it is doing this, it will blank all but the lower portion of the screen where it displays, 'Please Wait - Disk Copy In Progress'. It still lets the user know what is going on. If it didn't display this, the user might think that the computer has locked up. As soon as the copying is finished, the screen is restored and the window looks just as it did before the copying began. Some of the sample source code contains the same routines that dcopy uses so that you can do the same thing in your code if you need some extra room while your command is running. If you have a 128, you've no doubt seen the color80 command and obviously this command doesn't need to have the geoSHELL window running. The color80 command shows that even graphic type commands may be used with geoSHELL. The source code for color80 is included in this package so that you can have an idea of how to do something similar. The only thing to keep in mind is to make sure that your command can get the screen back into shape without using any of the geoSHELL routines that will stop your command. There aren't very many that will do this, but be aware of the ones that do. The description of each routine will tell you if this can happen. The routine that geoSHELL jumps through to halt your command is called 'NoMoreCmds'. This not only stops your command, but any others that the user might have typed in following yours will be ignored also. You can also jump through NoMoreCmds to halt your command in case of an error that you feel should return control to the user. Keep this in mind. In some cases, the user might type in two commands that relate to each other. If the first command detects a problem, it might not be a good idea to allow the second command to execute. This is something that you have to control in your command. Decide how serious of an error it is and then take appropriate action. You should try to display an error message and then jump to NoMoreCmds if the error is serious enough. If it is not something that would cause a

problem, then just end your command the clean way with a jump through 'ExitCommand'. This routine will return control to geoSHELL in the same state as it was when your command first started. geoSHELL will then process further commands if there are any.

USE geoSHELL TO YOUR ADVANTAGE

geoSHELL contains 101 routines that can be accessed from it's jump table. Use these routines to your advantage. Also, use the GEOS kernal routines. You can make your commands short if you can make use of what is already available to you. Some of geoSHELL's routines might never be useful to you, and some of them will be used in each and every command that you create.

Most simple commands can be written in just one or two pages of source code. The smaller they are, the faster they will load from disk. This is important to the user. Obviously, many commands that have already been written for geoSHELL could be translated into an application. One disadvantage to this would be that each application would end up being much larger and take up more room on a disk. Another disadvantage to this is that when the application is finished, geoSHELL or the Desktop must be reloaded from disk. geoSHELL commands do not remove geoSHELL from memory. It stays there. In fact, your command will also stay in memory until the user loads in another transient command from disk. This allows the user to do other things with geoSHELL and still be able to use your command again.

Another advantage to doing functions in GEOS with geoSHELL commands will be evident with those that use CMD devices. geoSHELL and all of it's commands can be placed in just one partition instead of having to have a copy in each partition like you would need if using the Desktop. Then, as long as the user identifies that partition with the 'path' command, geoSHELL will always have access to any of those commands, no matter which device or partition the user is working from. It is sort of like a big extention of the computer's memory. If your command were to perform some sort of printing function, the user merely types in the command and any required parameters, if needed, and the command is executed. The user doesn't even realize that geoSHELL changed partitions to get access to the command.

When geoSHELL executes your command, it turns control over to your command. Your command can now do what it is designed to do and then exit. The most simplest of commands would act as though they were just another routine in geoSHELL that is being jsr'd to. But you do not want to end your command with an 'rts'. This will not return you to geoSHELL's main processor. This returns you to the GEOS Kernal's mainloop. The reason geoSHELL was written to operate this way, was so that your command could make use of custom input routines and use the GEOS mainloop to catch user input. The command 'color80' works this way. At one point it will do an rts and wait for the user to click on an icon. It sets up an icon table for GEOS to look at when the user clicks the mouse. If you still want user input, and need a blinking cursor, geoSHELL has routines that will turn on the cursor for you, or turn it off, also. You will have to process the characters that the user enters. There are some routines that will do a certain amount of this for you. Plus, the GEOS kernal has routines for this also.

GETTING PARAMETERS FROM THE USER

When the user types in your command and enters a required parameter, you will have access to that parameter in several ways. There is a table that geoSHELL creates at 'ParamTable'. This table is 43 bytes long. The 43rd byte will always be a null. If there is any data following your command, it will be placed in this table. If no parameter was entered, this table will be all zeros. If one or more parameters were entered, they will be placed here at least up to the point where geoSHELL either finds a carriage return or a terminator, which could be an up-arrow character or a null byte. The remainder of the bytes in the table will be nulls.
There is also a pointer into the actual buffer where the commands are coming from. The command pointer, a4, points to this buffer. This buffer can be in one of two places. If the commands are coming from the user at the keyboard, a4 will be pointing somewhere within the command buffer. If the commands are being executed from a startup file or an exec file, then a4 will be pointing somewhere within the startup buffer. geoSHELL looks where a4 is pointing when it is ready to execute the next command.
When you use ParamName to get the filename, a2 and r6 will both be pointing at it. They will also be null-terminated. As you know, the user is allowed to use wildcards when entering a filename. ParamName will identify this fact for you. If a wildcard (? or *) is used, bit 7 of WILDCARD will be set. If your command works like the 'rename' command where the user needs to type in two filenames with an equals (=) sign separating them, then ParamName will have bit 5 set if the equals sign was used properly. And in this case, a4, the command pointer, will automatically be updated to point at the second filename as well as a2, while r0 will be pointing at the first filename. Both will be null-terminated. If bit 7 is set in this case, it will still reflect wildcards in the filename that a2 is pointing at. You are not notified if wildcards exist in the filename that r0 is pointing at. In the case of the rename command, the first filename should be entered without wildcards anyway.
Even though ParamName points a2 at a filename and terminates it with a null byte, the bytes contained in ParamTable will not be affected. You can still get access to the exact characters that the user typed in.
If you need to use the filename that r0 points at in the case of the user entering two filenames, you might want to save the value contained in r0, since that is a fairly popular zero page location. a2 will not get changed unless you call a geoSHELL routine that purposely changes it.
Another variable that might be useful is 'paramsize'. When geoSHELL loads your command and fills ParamTable, it also sets the variable paramsize with the number of bytes that were copied to ParamTable.

THE DIFFERENT COMMAND MODES

If your command needs more than what can be typed in at the keyboard, it would have to be a command that would only execute from a 'startup' or 'exec' file. The user can enter as much as will fit in the exec file at least until it gets too large for the startup buffer. The startup buffer, which is also used for exec files, is 1536 bytes in size. This can accomodate all but the larger GeoWrite pages. When you need to access a parameter that is larger than ParamTable will hold, just read the parameters beginning with the byte that a4 is pointing at.

There is a way to tell if your command is being executed from the command line or a startup/exec file. The variable 'STUPMODE' will identify this for you. If bit 7 of STUPMODE is set, then the command is being executed from a startup or exec file. If bit 7 is cleared, the user typed in the command from the keyboard. If the command came from a function key press, then the variable 'FKEYMODE' will have it's bit 7 set, otherwise it will be cleared. So, in some cases, your command may or may not be compatible with these different modes. As an example, geoSHELL's 'echo' command is not allowed to run from the keyboard. There is no reason for it, even though it would do no harm.
When you are creating your command, test it to be sure that it will operate in these different modes. Try to make it compatible with any of the different ways that the user might want to use it. In most cases, you won't have any problem with your command in any of the different modes, unless it does something out of the ordinary.

EXITING YOUR COMMAND

The proper way to exit a command is through the routine 'ExitCommand'. This way, geoSHELL knows you are finished and it will then proceed to check if any more commands exist that need to be executed. If not, then it will return control to the user. If you simply did an 'rts' instead of 'jmp ExitCommand', you would return to the GEOS mainloop and the user would be stuck, unless your command itself was looking for user input through mainloop. If not, then the only way the user might get out of this situation would be to click on geoSHELL's close icon, and return to the Desktop. So, when testing your command, should you forget this, just click on the close icon, return to the Desktop and then you can get back into geoSHELL from there. If the Desktop isn't found by geoSHELL, then geoSHELL will grab control of the computer and give control back to the user. Just be sure to end your command with the jump through ExitCommand and the user won't run into this problem.
If you had any parameters that your command used, you are required to update the command pointer before you exit your command. Otherwise, geoSHELL will try to execute your parameter and that obviously would not work. If your parameter were a filename, geoSHELL would attempt to load and run that file. This may not be what you had intended, but you are in charge until your command exits, so do as you please. You can put the command pointer wherever you want, but in most cases, you would place it just past your parameter, so that geoSHELL can check for another command following it and do as the user expects.
There is a routine that helps you to adjust the command pointer. It is called 'IncCmdPointer'. If you did not use any parameters, then you would not need to worry about updating the pointer, because it would already be pointing at the next command. But if you had a parameter, you need to put the pointer past it and point at the next command if there is any. Remember that from the user's standpoint, you are required to put at least one space after a command's parameter, so the command pointer needs to be incremented by the number of bytes that make up the parameter plus one for the space that follows it. Assuming your command had a one-byte parameter, the following code would increment the command pointer properly:

```
lda #2
```

```
jsr IncCmdPointer
```

You must load the accumulator with the amount to increment and then call the routine. If your parameter was a filename, and it contained 10 characters, you would have to increment the pointer by 11. geoSHELL will count the length of your filename for you if you use the routine 'ParamName' to fetch the filename that the user typed in. The length of the filename will be contained in 'NUM_IN_NAME'. Just load the accumulator with the value in NUM_IN_NAME and add 1 to it for at least one trailing space, and then call IncCmdPointer. Refer to the information about these routines for more sample source code on how to use them properly.If you were grabbing parameters directly from the command buffer or startup buffer if the command was executed from a startup or exec file (a4 points to the first parameter either way), then you might just manually update the command pointer (a4) as you go. The following code would do this:

```
      ldy #0
10$
      lda (a4),y
      ...
      ...   ;do what you need with the byte.
      ...   ;somewhere in here you might
      ...   ;branch out to 20$ if done.
      ...
      iny   ;or branch to here.
      bne 10$
      inc a4H    ;keep a4 pointing properly.
      bne 10$    ;branch always.
20$
      iny   ;one past our parameter.
      bne 30$
      inc a4H    ;in case y incremented to zero.
30$
      tya   ;increment the last little bit.
      jsr IncCmdPointer
      ...
      ...   ;continue on
      ...
```

WORKING WITH FILENAMES AND DRIVES

You've learned how to get a2 to point to the filename that the user has typed in. Now let's see how we can find a filename on a disk. There are several routines for doing this. You no doubt know of some of the routines that are built into the GEOS kernal for finding a file or accessing a file in some manner. But if you merely take the filename that a2 is pointing at and do the following routine, it might not work.

```
      MoveW a2,r6 ;point r6 to filename
      jsr   FindFile   ;and use GEOS kernal
                 ;to find it.
```

You are probably wondering why this wouldn't work. When using the DeskTop, the user clicks on an icon. The DeskTop knows the exact spelling of the file and the drive it is located on (the currently open drive). But with geoSHELL, the user can type in anything. What if the user included some wildcards in the filename? GEOS doesn't work with wildcards. Those have to be handled by the application. geoSHELL will do this for you. In fact, it is all handled automatically. And there are several routines you can call depending on the search you would like to perform. Here's one example:

```
      jsr   CkPresdrive ;find the file on the
                  ;currently active drive.
      bit   goodflag    ;Is it there?
      bmi   50$   ;branch if so.
```

This example would search the currently active drive for the filename that the user typed in. This would be the drive that the user sees as being open, according to the letter that is displayed at the left of the line. You can tell which drive it is by checking the value of PRESDRIVE. This variable always holds the device number of the user's currently active drive, or the 'present' drive. Using geoSHELL routines to search for filenames will properly handle the wildcards for your command. If the file is found, the variable 'goodflag' will have it's bit 7 set and your command will be able to then process the file as needed. Another thing that these routines do for you is that now a2 is pointing at the actual name of the file as it exists on the disk. The wildcards are converted into the characters that they are supposed to represent. Now the standard GEOS routines can process the filename because you know the exact name as it appears on the disk. If you wanted to keep a copy of the filename that the user typed in, then you might want to save a copy of where a2 is pointing before calling the routine. The routine will actually point a2 at a different location, where geoSHELL assembled the correct filename for you. This location is usually at 'filetoload'. Most of the time, you won't have a need for the original name, just the correct one. If geoSHELL doesn't find the file, then a2 won't change.geoSHELL has many ways to search for a file. Here are the different methods you can select from, depending on what you intend to accomplish:

CkPresdrive will search the current drive.
CkAllDrives will search all of the drives except the path partition, unless that particular partition is currently open on one of the drives.
CkThisDrive will search the drive that you specify in the accumulator. Just load the accumulator with a value from 8-11.
CkOtherDrives   will search any drive that you haven't yet searched with CkThisDrive. This allows you to use CkThisDrive to pick and choose the exact order you want, and then use CkOtherDrives to search all remaining drives. The path partition is ignored.
CkPath   will search the path partition.
CkPathOrder will search a specific order with certain priorities. First all ramdisks are searched, then the path partition, and finally any remaining drives.
FindParName will search the currently active drive just like CkPresdrive does, except you don't have to supply the filename for the search. This

routine will get it for you, since it calls ParamName to get the filename. The only drawback to this routine is that if the file is not found, it will exit through NoMoreCmds and your command will end. Be sure that the geoSHELL window is active if you use this routine.

Each of these routines will deal with any wildcards if contained in the filename that a2 points at. You can always test goodflag for a successful search. If goodflag is set, indicating that the file was found, then the variable DRVFOUNDON will contain the device number where the file was found. Plus the drive that contains the file will be open so that you do not have to call SetDevice or OpenDisk.
If you called a routine that searches the path partition, and if the file is found on that partition, then 'pathload' will have it's bit 7 set. In this case, you will be left with the path partition open. It is your responsibility to restore the partition that was previously open once you are through with that file and before you exit your command. Otherwise, the user might not appreciate it. There is a simple routine in geoSHELL to do this for you. It is called ResetFromPath. Call this when you are finished with the file and are ready to close the partition.
ResetFromPath will reopen the original partition on the path device and will also reselect the drive specified in PRESDRIVE. Also be sure to call ResetFromPath before using any routine that could cause your command to cease. You wouldn't want to leave the user in the wrong partition when control returns to him. You can always reopen the path partition if need be. It all works fairly quick, so it would not slow your command down to any noticable degree.

WORKING WITH PRINTERS

There are routines in geoSHELL for accessing a printer or interface connected to the serial port. Unfortunately, they were mistakenly left out of the jump table. The '@p' command causes geoSHELL to send output to the printer through these routines. In the sample source code files that are included with this package, you will find some source code that you can put in your command for accessing the serial printer or interface.
On a brighter note, the routines that access a printer connected to the user port with a geoCable 'is' in the jump table. There are two routines, Open_GC_Channel and Send_GC_Byte. You don't have to call Open_GC_Channel. However, it is a good idea to do so, because it will check to make sure there is a printer connected and responding before you attempt to send any bytes to it. This routine will return with x=0 if a printer was found and is able to accept data. The universal 'goodflag' is not used with this routine. Once you are sure that a printer is there, use Send_GC_Byte to transmit the bytes. Just load the accumulator with the byte to send and call the routine.
One of the things you might want to try when testing your routine is to see how it works with the @p or @g command, even if your command has nothing to do with the printer itself. Make sure that when geoSHELL is sending screen output to the printer, that your command will work with it. If not, then you would want to either inform the user with an error message, or turn the printer output off yourself. You can check this with the variable PRNTMODE. Any non-zero value here will send output to a printer. Bit 7 set goes to the serial port and bit 6 set goes to the user port. You can turn these off if the printer doesn't get along with your

command for some reason. Just be sure to restore it before you exit your command.

DISPLAYING MESSAGES

geoSHELL is full of messages. A good deal of the program is involved with sending messages to the screen in order to inform the user what is taking place. But it seems as though there were times when it should have even more messages.
As geoSHELL evolved, it grew larger and larger. At one point in time, it did not have the ability to use transient commands. All commands were contained in memory and so every command was considered a resident command. But it got to the point where something had to go. So, some commands were removed from geoSHELL and made into transient commands. Some were moved into VLIR records in the geoSHELL file and others were placed in their own separate files as external transients. This also allowed many of geoSHELL's messages to get moved out of memory and into the transient commands. In this way, more messages could be added to the commands themselves, since they would not require any of geoSHELL's main memory. But, I guess you just can't have enough messages. Not everybody will read a manual, but they will usually notice a message that is displayed in front of them.
So, you will want to keep the user informed of any error that occurs, or, if your command is doing something that might require some processing time, let the user know about it. Don't make him think that his computer has just locked up. But, keep your messages short and to the point.
Another thing to remember is to keep your messages limited in length. Remember, a 128 user might be able to have an 80 character message on his screen, but a 64 user can't. So, for the longer messages, you will have to break them up into two lines. Remember that a 128 user might also be in 40 column mode. You have 40 columns to work with, so keep the messages to a maximum of 40 characters including any spaces.
There are two routines that will help you to display a message to the screen, 'Message' and 'OtherMessage'. The routine Message will allow you to have access to the messages that are built-in to geoSHELL. If you can use any of these, it will make your own command that much smaller.
OtherMessage lets you use messages that are contained in your command.
Here is a couple of examples:

```
;Example #1
      ...
      ldx   #15
      lda   #(IN_TWO|TR_CR)
      jsr   Message
      ...

;Example #2
      ...
      LoadW r0,#wrongText
      lda   #(IN_TWO|TR_CR)
      jsr   OtherMessage
      ...

wrongText:
      .byte "Wrong Filetype!",0
```

In the first example, you will be displaying the internal message #15 because you loaded x with 15 and called Message. Message #15 will display 'Insufficient Room On Disk!'. Notice that you also had to load the accumulator with a value. This value tells the message routine how to display your message. There are some constants for this. The ones we used here are IN_TWO and TR_CR. The first one, IN_TWO, stands for 'initial two spaces', and will make the message appear two spaces over from where the cursor is and the second one, TR_CR, stands for 'trailing carriage return', and will produce a carriage return after your message is displayed.
The second example above uses OtherMessage. Here, you have to point r0 to a null-terminated string, load the accumulator with the formatting information and then call the routine.
If a message causes printing to exceed the limit of the 40 column screen, it will simply be chopped off at the right side. No harm will occur, other than the user won't see the whole message.
In the Appendix, you will find a list of the built-in messages that you can access along with the different constants that can be loaded into the accumulator before calling these routines.
DISPLAYING LARGE AMOUNTS OF TEXT

If you need to display large amounts of text, there are routines in geoSHELL for doing this, complete with word-wrap. The 'type' and 'echo' commands make use of these routines.
The echo command can display any text following it until it encounters either a null byte or an up-arrow terminator. Here is a method you can use that is similar to the one that the echo command uses:

```
      LoadB endecho,#128     ;initialize a flag.
      LoadB typeset,#0  ;ignore tabs and carriage
            ;returns.
      MoveW a4,r0 ;point r0 to the start
            ;of your parameter or the text
            ;that follows your command.
 10$
      jsr   DispText   ;display up to 80 columns
            ;of text.
      jsr   AdjTxtPointer    ;advance a pointer to
            ;some more of the text.
      bit   endecho    ;have we reached the end?
      bmi   10$   ;branch if not.
      MoveW r0,a4 ;adjust the command pointer.
      rts
```

In the above example, using DispText and AdjTxtPointer together, you can display any text that r0 points at. In this example, we are displaying text that follows our command since we are pointing r0 at the same spot that a4 is pointing at. DispText will give you word wrap, because it will only display complete words to the screen. Calling AdjTxtPointer will point r0 at the first word that DispText didn't display. We know that we have reached the end of the text when bit 7 of endecho is cleared. The routine starts out by setting endecho's bit 7, otherwise only one line of text will get displayed. Then as long as bit 7 remains set, there is still some text left and so we loop back to display some more. The above routine will terminate if a null byte or an up-arrow terminator is reached. It will work this way if typeset is set to zero. If bit 7 of typeset is set, then an up-arrow will not terminate the routine, only a null byte will. Also, this would cause carriage returns and tabs to be recognized. A tab will not be treated as a true tab, but will be expanded to equal 5 spaces. That is how the 'type' command works with DispText. Of course, though, since the type command is capable of displaying almost any kind of text file, it also does some other processing of the text it reads from disk before passing it on to DispText. DispText only understands true ascii text. DispText will also call routines that will scroll the text when it reaches the bottom of the geoSHELL window. CkKeyboard is also called to allow the user to pause or stop the text. If the STOP key is pressed, your command will terminate and control will return to the user. So you should only use these routines if the geoSHELL window is active. The routines are designed to display text in the window,anyways. If you wish to defeat this, you can turn the keyboard off by setting bit 7 in the variable 'kboardoff'. If you set bit 6, then only the STOP key will be ignored. Reset kboardoff to zero when you are finished.

COLORING THE 80 COLUMN SCREEN

geoSHELL has a command called 'rgb' that allows a user of the 128 with a color monitor to display geoSHELL in multiple colors, instead of just the two colors that GEOS uses for most applications. Since geoSHELL has been given this ability, the routine to put the color on the screen has been made available in the jump table so that any command can have access to the same routine and add color if it operates outside of the geoSHELL window. This is what the command 'color80' does. The routine that puts color on the screen is called 'ColorScreen'.
Before attempting to color the screen, you should be sure that the user has put the computer in color mode. First, your command should check to make sure that the machine is a 128 in 80 column mode. Test bit 7 of screenmode to check for 80 column mode. If it is set, then the machine is a 128 running in 80 columns. Following this, test bit 7 of videomode. If it is set, then the user has selected color mode with the 'rgb' command. If this all checks out OK, then you can proceed with your command. If the machine is not a 128 in 80 column color mode, then you should inform the user as such, and then exit the command appropriately.
ColorScreen expects two things, a table of compressed bytes pointed at by r0, which from here on we will call a 'Color Scrap', and a table of color combinations pointed at by r1. In the table of color combinations, there can be as many as 255 different combinations of colors. The reason that a separate table is used for the colors is to make it easy to alter the colors in any Color Scrap just by modifying the different color combinations in this table instead of having to go through the Scrap itself and changing each color byte. The colors can be changed very easily while your command is running, or while you are developing the source code. Each combination is a one byte value consisting of a foreground color and a background color. The foreground color (0-15) is stored in the upper nybble of the byte, while the background color is

stored in the lower nybble. So, a byte that defines a black foreground with a white background would be (BLACK80<<4)|WHITE80. Using GeoAssembler, this will shift the value of BLACK80 left 4 bits and 'or' it with WHITE80. In hexidecimal form it would be $0f, since BLACK80 equals 0 and WHITE80 equals 15. In the description for the Color Scrap, you will see how ColorScreen uses this table of color combinations. Here are all of the equates that you can use instead of having to remember which color number corresponds to which color. These equates are all assembled in GSVariables.rel.

```
BLACK80     = 0   ;black
DK80GREY    = 1   ;dark grey
DK80BLUE    = 2   ;dark blue
LT80BLUE    = 3   ;light blue
DK80GREEN   = 4   ;dark green
LT80GREEN   = 5   ;light green
DK80CYAN    = 6   ;dark cyan
LT80CYAN    = 7   ;light cyan
DK80RED     = 8   ;dark redLT80RED = 9   ;light red
DK80PURPLE  = 10  ;dark purple
LT80PURPLE  = 11  ;light purple
DK80YELLOW  = 12  ;dark yellow
LT80YELLOW  = 13  ;light yellow
LT80GREY    = 14  ;light grey
WHITE80     = 15  ;white
```

The Color Scrap defines the areas of the screen that you would like to have color applied or changed from what is currently there. You can color anything from one 8 x 8 pixel area to the full screen. The screen is made up of 1760 individual areas that we can add color to. Normally, there is 2000 areas, but since geoSHELL doesn't use the lower 3 rows when in rgb mode, we are limited to 1760 color areas. These areas correspond to the same character locations if the screen were in text mode such as when you first turn the computer on. We will refer to each area as a card. There are 22 lines of 80 cards on the screen that we can add color to. Each card or line is eight pixels high and each card is eight pixels wide. In the VDC chip's own memory, there are 1760 bytes that hold the color information for each card on the screen. The first byte corresponds to the card at the upper left corner of the screen. Bytes 0-79 represent the cards in the first row. Bytes 80-159 represent the cards in the second row, and so on. So, you can see that each row is stored sequentially in memory. When designing a Color Scrap, you have to keep this arrangement in mind.
The lower three card rows that are blanked out are always in the same color as the border. This way, the blanked out rows just make the lower border area look thicker. geoSHELL provides you with a routine to change the border color. It is called SetBrdrColor. To use it you must first load brdrcolor with the desired color 0-15. Here again, make sure that the machine is in color mode.

Let's design a simple Color Scrap.

Here is how a Color Scrap is laid out:

```
byte 1      row (0-21) to begin at.
byte 2      column (0-79) to begin at.
byte 3      number of times (1-255) to repeat the next command.
byte 4      number of 2 byte commands (1-255) that follow.
byte 5      number of sequential cards (1-255) to color.
byte 6      offset (0-254) into color table of color to use.
      If 255, then no coloring will take place, the routine will simply
increment the number of cards defined by byte 5.

      Bytes 5 and 6 are a pair of bytes that is the first command set in
the number of command sets that is defined by byte 4. If byte 4 was a 2,
then byte 7 and 8 will also be a command set. If byte 3 is a 2, then the
group of bytes from byte 4 through byte 8 will be performed twice. In
this case, then byte 9 will be thestart of the next group.
byte 9      If 0, then end of table.
      If 255, then the next two bytes define a new row and column to
begin coloring from.

      and so on until end of table...

;This example will color a 4 x 4 card box where the upper left ;corner of
the box is at row 2, column 10.

ColorBox:
      LoadW r0,#boxScrap
      LoadW r1,#boxColors
      jmp   ColorScreen

boxScrap:
      .byte 2,10  ;start at row 2, column 10.
      .byte 4     ;repeat the next number of
                  ;sets 4 times so that 4
                  ;rows get colored.
      .byte 2     ;number of command sets.
      .byte 4,0   ;color 4 cards with
                  ;color 0 (1st color combo)
                  ;from the color table.
      .byte 76,255      ;skip 76 cards without
                  ;coloring.
      .byte 0     ;end of table.

boxColors:
      .byte (DK80RED<<4)|LT80GREY ;use one color combo
                  ;with this example.


;this same color scrap could also be written like this to
;accomplish the same result.

boxScrap:
      .byte 2,10  ;start at row 2, column 10.
      .byte 1     ;repeat once.
      .byte 1     ;number of command sets.
      .byte 4,0   ;color 4 cards with color 0
                  ;from the color table.
      .byte 255   ;get a new starting value.
      .byte 3,10  ;start at row 3, column 10.
      .byte 1
      .byte 1
      .byte 4,0
      .byte 255   ;get a new starting value.
      .byte 4,10  ;start at row 4, column 10.
      .byte 1
      .byte 1
      .byte 4,0
      .byte 255   ;get a new starting value.
      .byte 5,10  ;start at row 5, column 10.
      .byte 1     .byte 1
      .byte 4,0
      .byte 0     ;end of table.
```

Even though this would accomplish the same result, it requires a bigger table and will also take about twice as long to put the color on the screen, although the difference in speed can't be seen with the eye. So, when you are creating your Color Scrap, think if you can make it more compact. Here is an example that will put color on the whole screen.

```
      .byte 0,0   ;start at row 1, column 1.
      .byte 22    ;repeat 22 times.
      .byte 1     ;one command set.
      .byte 80,0  ;color 80 cards with 1st
                  ;color from table.
      .byte 0     ;end of table.
```

Once you get the hang of it, you will find it rather easy to create quite complex Color Scraps. Putting color on the 80 column screen really adds to it's appeal. There is no reason not to do it with any new GEOS application, not just geoSHELL. The only drawback is that not everybody has 64K of video ram in their 128s. This is why geoSHELL limits the coloring of the screen to the first 22 card rows due so that the users with only 16K of video ram will still be able to have color.
When you are using the 80 column color screen, remember that when you use geoSHELL to clear the screen for you with ClearScreen, it will clear it with the background pattern that is stored in clr80pattern. Depending on what you are doing with the screen, the background pattern that the user has chosen for the geoSHELL screen may not look good with your command. The trick here is to store the pattern byte that you want at clr80pattern and restore the original one before your command exits. In fact, you can do that with any of the color or pattern bytes that geoSHELL stores beginning at coltable. Here is a list of those variables, and you can see that all of the pattern bytes are stored just after the color variables.

```
coltable:         ;80 column color table.
backcolor:  .block 1   ;screen background color.
shellcolor: .block 1   ;shell color.
textcolor:  .block 1   ;text color.
padcolor:   .block 1   ;pad background color.
```

```
brdrcolor:  .block 1    ;screen border color.
back40pattern:   .block 1   ;backgrnd pattern for 40 column.
back80pattern:   .block 1   ;backgrnd pattern for 80 column.
clr80pattern:    .block 1   ;backgrnd pattern for RGB mode.
```

Feel free to experiment with different settings if your command needs to, but don't forget to restore them. If you change any of these variables, you won't see the effect until you force geoSHELL to clear the screen or redraw the geoSHELL window. Otherwise you will have to do your own coloring with ColorScreen.
-- NOTES --

SECTION 2

This section of the manual contains information about each of the routines that are accessible in geoSHELL from it's jump table. It is laid out in a manner that should be familiar to those of you who have either the Official GEOS Programmer's Reference Manaul by Bantam Books or The Hitchhiker's Guide To GEOS by Berkeley Softworks.

In some of the source code examples that are given, you might see references to a jump to another routine. Some of these might refer to other routines or error routines that you would create somewhere in your own source code. Hopefully, it is understandable which routines are geoSHELL routines and which are not.

Whenever you access any of the geoSHELL routines, be sure that you are doing it from a mode that is allowed. For instance, if you have called the GEOS kernal routine InitForIO, you can't call a geoSHELL routine that also does so. Likewise, some geoSHELL routines require that you call InitForIO first. If you don't, this could be the source of a serious crash or lockup. Read the description about the routine thoroughly before using it. You might also want to refer to the index to find other parts of this manual that might have information about a particular routine.
AdjTxtPointer

Function: Used with DispText to advance r0 within a text buffer.

Pass: textpointer - any value 0-255 (normally set by DispText).

Return: r0 - holding new value.

Destroys: a

Description: When using DispText to display text to a line of the geoSHELL window, r0 will still be pointing at the text and the variable textpointer will hold a value that if added to r0 will then point r0 at the next segment of text that should be displayed. Calling this routine will add that value to r0.

Example: See the entry for DispText for an example on how to use this routine.

See Also: DispText, Message, OtherMessage, DsplyLine, DsplyString, CarrReturn, ClearLine, ClearBoth, ClearRemainder, StripRulers, set_XY_pos, save_XY_pos, MskCtrlCodes, ConvToAscii, CkESC_RULER

Asc_3_Byte

Function: Convert an ascii number (0-255) to a single byte.

Pass: r0 - pointer to a 1,2 or 3 digit ascii number.

Return: r5L - contains the byte.

a - contains the byte.

goodflag - set if value is allowed.

r0 - unchanged.

Destroys: a, x, y, r5L, r8L

Description: Point r0 at an ascii representation of a byte from 0-255 and call this routine to convert it to a single byte that will be placed in r5L. If r0 was pointing to something that did not represent a number between 0 and 255, then the routine will exit through nogood. Always check goodflag before proceeding any further. The number should also end with either a space or some sort of terminator such as an up-arrow, or a null byte. When the user types in a number such as a parameter after your command, ParamTable will be loaded with this number. You could then point r0 at ParamTable and call this routine.

Example:

```
;this example will get the number the user types in and then
;locate a requested font file that was typed after the
;number on the current drive.

      ...
      ...        ;some of your code.
      ...
      LoadW r0,#ParamTable   ;point at the first parameter.
      jsr   Asc_3_Byte ;convert it to a single byte.
      bit   goodflag   ;was it a good value?
      bmi   10$   ;branch if so.
 5$
      jmp   DoError    ;jump to an error routine.
 10$
      MoveB r5L,pointSize    ;save the value for now.
      ldx   #1    ;there was at least one digit.
 20$
      lda   ParamTable,x
      beq   5$    ;branch if no more data.
      cmp   #' '  ;look for the first space.
      beq   30$
      inx
      bne   20$   ;branch always.
 30$
      inx        ;point one past the space.   txa
      jsr   IncCmdPointer    ;aim the command pointer.
      jsr   ParamName  ;and get a possible filename.
      bit   goodflag
      bpl   5$    ;branch if not valid.
      jsr   CkPresdrive ;look for the file on the
                 ;current drive.
      bit   goodflag
      bpl   5$    ;branch if not found.
      ...
      ...        ;at this point we can
      ...         ;continue to check if it
      ...         ;is a font file.
      ...

pointSize:
      .block     1
```

See Also: Asc_Byte, Asc_BCD, ByteNZ_Ascii, ByteWZ_Ascii, ConvertK

Asc_BCD

Function: Convert a two digit ascii number to one byte in binary coded decimal (BCD) format.

Pass: r5L - first ascii digit of number '0'-'9'
      r8L - second ascii digit of number '0'-'9'

Return: r5L - BCD byte $00 - $99

goodflag - set if a valid number and within the allowable range.

Destroys: a

Description: This routine will take a two-character ascii decimal number from '00' - '99' and convert it to a one-byte binary coded decimal number from $00 - $99 and return it in r5L.

Example:

```
;this example takes a one or two character number from the
;user (0-99) and converts it to a one-byte BCD number.
      ...
      ...
      lda   ParamTable+1     ;did the user enter just
      jsr   CkTerminators    ;one digit? (0-9)
      bit   goodflag
      bmi   10$   ;branch if so.
      lda   ParamTable+2     ;how about two
      jsr   CkTerminators    ;digits? (00-99)
      bit   goodflag
      bmi   20$   ;branch if so.
 5$
      jmp   NotValid   ;do an error message.
 10$
                  ;a single digit number is
                  ;handled here.
      MoveB ParamTable+0,r8L ;r8L gets the one (2nd) digit.
      LoadB r5L,#'0'   ;we create the first digit.
      beq   30$   ;branch always.
 20$
      MoveB ParamTable+0,r5L ;load r5L with first digit.
      MoveB ParamTable+1,r8L ;load r8L with second digit.
 30$
      jsr   Asc_BCD    ;convert to BCD byte.
      bit   goodflag   ;was it a valid number?
      bpl   5$    ;branch if not.
      ...         ;continue on...
      ...
```

See Also: Asc_Byte, Asc_3_Byte, ByteNZ_Ascii, ByteWZ_Ascii, ConvertKAsc_Byte

Function: Convert a two digit ascii number to a one byte value.

Pass: r5L - first ascii digit of number '0'-'9'
      r8L - second ascii digit of number '0'-'9'

Return: r5L - value of 0-99.

      goodflag - set if a valid number and within the allowable range.

Destroys: a

Description: This routine will take a two-character ascii decimal number from '00' - '99' and convert it to a one-byte value and return it in r5L.

Example:

```
      ...
      ...
      MoveB asciiNum+0,r5L   ;load r5L with first digit.
      MoveB asciiNum+1,r8L   ;load r8L with second digit.
      jsr   Asc_BCD    ;convert to one byte.
      bit   goodflag   ;was it a valid number?
      bmi   10$   ;branch if so.
      jmp   DoError    ;display some sort of error.
 10$
      ...         ;continue on...
      ...

asciiNum:
      .byte "58"
```

See Also: Asc_3_Byte, Asc_BCD, ByteNZ_Ascii, ByteWZ_Ascii, ConvertK

ByteNZ_Ascii

Function: Convert a 16 bit number to an ascii representation of the number with no leading zeros.

Pass: r8 - number to convert.

Return: Asc_String - ascii characters '0' to '65536'.

Destroys: a, x, y, r4, r8

Description: This routine will take the 16 bit value in r8 and convert it to it's ascii decimal representation and put the result of the conversion at Asc_String. Asc_String will also contain a null byte in the sixth character position. If the result is less than five digits, then any leading zeros are replaced by ascii spaces.

Example:

```
;This example will take the value in r0 and display it to the
;screen at the current cursor position.

ShowValue:
      MoveW r0,r8 ;copy r0 to r8.
      jsr   ByteNZ_Ascii     ;convert to a string.
      LoadW r0,#Asc_String   ;point r0 to the string.
      ldy   #0    ;point to the left-most digit.
 10$
      lda   (r0),y      ;check a digit to see if
      cmp   #' '  ;it is a space character.
      bne   20$   ;branch on the first
                  ;non-space character.
      iny         ;point to the next one
      bne   10$   ;and branch always.
 20$
      clc         ;add the value in y
      tya         ;to r0.
      adc   r0L
      sta   r0L
      lda   r0H
      adc   #0
      sta   r0H
                  ;r0 now points to the
                  ;first non-space character.
      lda   #TR_CR      ;do a carriage return
      jmp   OtherMessage     ;after displaying the string.
```

See Also: Asc_3_Byte, Asc_Byte, Asc_BCD, ByteWZ_Ascii, ConvertK

ByteWZ_Ascii

Function: Convert a 16 bit number to a 5 digit ascii representation of the number with leading zeros.

Pass: r8 - number to convert.

Return: Asc_String - five ascii characters containing 00000 to 65536.

Destroys: a, x, y, r4, r8

Description: This routine will take the 16 bit value in r8 and convert it to it's ascii decimal representation and put the result of the conversion at Asc_String. Asc_String will also contain a null byte in the sixth character position. The result will always have five digits, since leading zeros are left in.

Example:

```
;This example will take the value in r0 and display it to the
;screen at the current cursor position with leading zeros
;remaining.

ShowValue:
      MoveW r0,r8 ;copy r0 to r8.
      jsr   ByteWZ_Ascii      ;convert to a string.
      LoadW r0,#Asc_String    ;point r0 to the string.
      lda   #TR_CR      ;do a carriage return
      jmp   OtherMessage      ;after displaying the string.
```

See Also: Asc_3_Byte, Asc_Byte, Asc_BCD, ByteNZ_Ascii, ConvertK

Byte_Time

Function: Convert the time as contained in directory entries to ascii format. (this routine should not be used)

Pass: not applicable

Return: not applicable

Destroys: not applicable

Description: This routine should not be used. The variables that need to be set are only accessible by internal routines. The routine was invariably left in the jump table and should have been removed, or the required variables should have been moved to where they could be accessed by outside routines. For these reasons, only internal geoSHELL routines have proper access to this routine.

Example: no example since this routine is not recommended.

See Also:

CarrReturn

Function: Perform a carriage return.

Pass: nothing

Return: cur_line - line that cursor is now on.

cur_X_pos - screen X coordinate of cursor.

cur_Y_pos - screen Y coordinate of cursor.

Destroys: no guarantees

Description: Simply perform a carriage return in the geoSHELL window and the cursor will then be on the next line down. If the cursor was already on the last line, then all the lines will scroll up and the cursor will still be on the last line.

Example:

```
      ...
      jsr   CarrReturn ;move the cursor down
      jsr   CarrReturn ;two lines.
      ...
```

See Also: AdjTxtPointer, Message, OtherMessage, DsplyLine, DsplyString, DispText, ClearLine, ClearBoth, ClearRemainder, StripRulers, set_XY_pos, save_XY_pos, MskCtrlCodes, ConvToAscii, CkESC_RULER

CkAllDrives

Function: Check all drives for a file.

Pass: a2 - null terminated filename.

Return: goodflag - set if file is found.

DRVFOUNDON - device number of the drive where file is found.

curDrive - the drive where the file is located is now open.

PRESDRIVE - unchanged

a2 - points to the filename converted to the actual name, any wildcards are converted to the characters they represent.

dirEntryBuf - directory entry for this file.

Destroys: no guarantees

Description: This routine will search all available drives A, B, C, and D for the filename that a2 points at. Upon return, the first thing to test is goodflag. If goodflag is cleared, then the file was not found for whatever reason. If goodflag is set, then the file exists and the drive it is on will be the open drive. PRESDRIVE will still reflect the drive that the user thinks is currently active.

Example:

```
      ...
      ...
      jsr   ParamName   ;point a2 at a filename the
                  ;user typed in.
      bit   goodflag    ;is there a filename?
      bmi   10$   ;branch if so.
```

```
 5$
     jmp   DoError    ;display an error.
 10$
     jsr   CkAllDrives ;look for the file.
     bit   goodflag   ;does it exist somewhere?
     bpl   5$    ;branch if not.
     ...
     ...        ;continue on...
```

See Also: CkPresdrive, CkThisDrive, CkOtherDrives, CkPathOrder, CkPath, CkForDisk, FindParName

CkDrvLetter

Function: Tests if the accumulator is holding a value from 97 to 100 ('a', 'b', 'c', 'd') and converts it to a device number.

Pass: a - with value to test.

Return: goodflag - set if value falls within allowable range.

a - converted to device number 8-11

Destroys: nothing (a is altered however)

Description: This routine can be used by any routine that might work with the drives and gets an input from the user. This will test if a value falls within a range that would equal the ascii value that represents a, b, c, or d. If the accumulator holds a value less than 97 or greater than 100, then the routine will exit through nogood. Otherwise, the routine exits through yesgood and the accumulator will contain a new value from 8-11.

Example:

```
     ...
     ...
     lda   ParamTable+0     ;get the first parameter.
     jsr   CkDrvLetter ;is this a, b, c, or d?
     bit   goodflag   ;branch if so.
     bmi   10$
     jmp   NotDriveLetter   ;go do an error.
 10$
     ...        ;accumulator now
     ...        ;holds 8, 9, 10, or 11.
     ...
     ...        ;continue processing.
     ...
```

See Also: CkForDisk, ParamName

CkESC_RULER

Function: Used for advancing a pointer past bytes used with ruler escapes in a GeoWrite document.

Pass: a - value that r0 is pointing at.

Return: * If byte is a ruler escape:

r0 - incremented by the number of bytes used in the ruler.

goodflag - cleared if byte was a ruler escape.

* If byte is not a ruler escape:

a - unchanged.

r0 - unchanged.

goodflag - set if byte was not a ruler escape.

Destroys: If byte is a ruler escape:
a, x

If byte is not a ruler escape:
x

Description: Here is a routine that geoSHELL uses when processing a startup or exec file. Each byte of the GeoWrite page is sent through this routine to jump past any rulers that might be contained within the file. geoSHELL only needs the actual text that is contained in the file. The text is what makes up the commands. The user is allowed to use different fonts and graphics in a startup file, but geoSHELL doesn't need them. By loading the accumulator with each byte and accessing this routine, the rulers can be stripped from the buffer that the file is loaded into. The 'type' command also makes use of this routine when displaying a GeoWrite file to the screen.

Example:

```
;this example will strip the rulers from a buffer that a
;GeoWrite page was loaded into and then display it
;within the geoSHELL window. The example AdjustBuffer will work
;because the size of the bytes in the buffer will always
;decrease in size. Refer to DispText for an example on how
;to display this converted text to the screen.

AdjustBuffer:
      lda   #[ourBuffer;point r0 and r1
      sta   r0L   ;to the buffer that a
      sta   r1L   ;GeoWrite page is
      lda   #]ourBuffer;loaded into.
      sta   r0H   sta   r1H
      ldy   #0    ;start y at zero.
 10$
```

```
      lda   (r0),y      ;get a byte from the buffer.
      beq   90$   ;branch if end of text.
      jsr   CkESC_RULER;process this byte.
      bit   goodflag    ;do we have a good byte?
      bpl   10$   ;branch if not.
      sta   (r1),y      ;store this byte.
      iny         ;increment to point at the
                  ;next byte.
      bne   10$   ;branch if y hasn't not zero.
      inc   r0H   ;otherwise increment the
      inc   r1H   ;high bytes of our pointers.
      bne   10$   ;branch always.
 90$
      rts         ;done with this routine.
```

See Also: AdjTxtPointer, Message, OtherMessage, DsplyLine, DsplyString, CarrReturn, ClearLine, ClearBoth, ClearRemainder, StripRulers, set_XY_pos, save_XY_pos, MskCtrlCodes, ConvToAscii, DispText

CkForDisk

Function: Check for a disk in any desired drive.

Pass: accumulator - device number from 8-11.

Return: goodflag set if the drive contains a formatted disk.

Refer to OpenDisk for other values that are set.

Destroys: no guarantees

Description: This routine performs a simple check to see if a particular drive contains a valid disk that can be read. You might call this routine before processing your command if it requires disk access. This would allow you to inform the user of a 'missing disk' if necessary. The routine jumps through yesgood if a valid disk is found.

Example:

```
      ...
      ...
      lda   ParamTable+0      ;get a parameter from the user.
      jsr   CkDrvLetter;convert it to 8-11.
      bit   goodflag    ;was it a, b, c, or d?
      bmi   10$   ;branch if so.
      jmp   DoError     ;display an error.
 10$
      jsr   CkForDisk   ;does the desired drive have
      bit   goodflag    ;a disk in it?
      bmi   20$   ;branch if so.
      jsr   MissDisk    ;do a geoSHELL error routine.
      jmp   NoMoreCmds  ;and quit the command.
 20$
      ...         ;otherwise,
      ...         ;continue on...   ...
```

See Also: CkPresdrive, CkThisDrive, CkOtherDrives, CkPathOrder, CkPath, CkAllDrives, MissDisk, FindParName

CkKeyboard

Function: Check for CONTROL, STOP and other keys and process accordingly.

Pass: kboardoff - (optional) ignore the keyboard if bit 7 is set, or if clear, continue with CkKeyboard. Set bit 6 to ignore only the STOP key.

Return: y - keycode of the key that is being pressed.

Destroys: a, x, y

Description: This routine is usually used to check for the user pressing the STOP key or the CONTROL key. If the STOP key is pressed when CkKeyboard is called, your command will end and CkKeyboard will exit through NoMoreCmds. If you call CkKeyboard, be sure that the geoSHELL window is still displayed on the screen, otherwise the user will have whatever your command has put on the screen with a blinking cursor somewhere in the middle. Also be sure that InitForIO is not in effect while calling this routine.

If the user presses the CONTROL key during this routine, the routine will simply pause and wait for the user to let go of the CONTROL key and then it will return. If any other key is pressed, the keycode of the key will be returned in the y register.

CkKeyboard should be called whenever your command does something such as displaying a series of scrolling text that the user might want to halt. Just put CkKeyboard somewhere in the loop of your routine. Think about how your command is working and try to imagine if the user might want a way out or a way to pause the output. In that case, use this routine. If your routine is using certain geoSHELL routines, CkKeyboard may already be in use. Test your command to see if the STOP key or CONTROL key is already functioning.

Example:
```
     ...        ;the bulk of your command.
     ...
 10$
     jsr   CkKeyboard ;allow the user a way out.
     ...
     ...        ;continue with something here.
     ...
     bit   goodflag   ;this might be another way out.
     bmi   10$   ;loop back and do it again.
 90$
     rts
```

See Also: Wait, YesKeypress, NoKeypress, NoMoreCmdsCkModeFlag

Function: Test if a file is compatible with the current mode.

Pass: Load fileHeader with the file's header block.

Return: goodflag - set if file is compatible.

x - pointer to error message if file is not compatible.

Destroys: a, x

Description: geoSHELL uses this routine before it allows a file to be loaded and run. It will check the mode flag in the file's header block which is loaded in at fileHeader. It checks to make sure that the file is configured as being compatible with the mode that the computer is currently running in, whether it be 64, 128, 40 or 80 columns, etc. If the current mode is not compatible with the file, then the routine will exit through nogood and the value in x will be a pointer to a geoSHELL error message that you can display if desired. Otherwise, if goodflag is set, then the file is compatible. See the appendix for a listing of the correct byte to use in the header block for a particular mode.

Example:

```
;this example will get a filename from the user and check the
;currently active drive for the file and if it is compatible,
;it will continue with whatever it is intending to do.

     ...
     ...
     jsr  ParamName  ;get a filename.
     bit  goodflag   ;is it valid?
     bmi  10$   ;branch if so.
 5$
     jmp  DoError    ;display an error.
 10$
     jsr  CkPresdrive ;this will load dirEntryBuf.
     bit  goodflag   ;does the file exist?
     bpl  5$    ;branch if not.
     jsr  GetHeader  ;get the file's header block.
                ;GetHeader doesn't return if
               ;any error.
     jsr  CkModeFlag ;check the file's compatibility.
     bit  goodflag   ;well?
     bmi  20$   ;branch if OK.
     lda  #(IN_TWO|TR_CR)  ;x points to the error message.
     jsr  Message    ;display the message.
     jmp  NoMoreCmds ;and quit.
 20$
     ...        ;otherwise, continue on...
```

See Also: GetHeader, Only128, Message, OtherMessage, DoRun

CkOtherDrives

Function: Search all drives for a file that have not yet been searched.

Pass: a2 - pointer to a null-terminated filename.

drivesChecked - (4 bytes) identify the drives that should be searched. If zero, then search the drive, if bit 7 is set, then ignore the drive.

Return: goodflag - set if file found.

a2 - now points to the exact name of the file in case the user included wildcards in the filename.

DRVFOUNDON - set to device number of the drive the file was found on.

curDevice and curDrive - the drive the file was found on is now open.

PRESDRIVE - unchanged.

dirEntryBuf - directory entry for file if found.

Destroy: no guarantees

Description: When searching for a file, you can choose the order that you wish to search the drives in. Just load the accumulator with a device number and call CkThisDrive. You can check another drive if desired also. When you've checked the ones that you feel should have priority, and wish to check the remaining ones, just call CkOtherDrives and geoSHELL will proceed to do just that. CkOtherDrives will check four locations at drivesChecked before accessing a drive. If a corresponding byte at drivesChecked is zero, that drive will be searched. The ramdisks get priority here, and then the real drives are tested. By manipulating the bytes at drivesChecked, you can force CkOtherDrives to either check a drive or to ignore it. So, you might want to mark the drives that you have already checked. Otherwise, those drives will get searched again. If all the drives have already been checked, then the routine will return as though the file was not found anyway and goodflag will be cleared.

Example:

```
;this example will search only drives A and B for a desired file.
      ...
      jsr   ParamName  ;get a filename from the user.
      bit   goodflag   ;is it valid?
      bmi   10$   ;branch if so.
      jmp   DoError    ;display an error message. 10$
      lda   #128
      sta   drivesChecked+2  ;ignore drive C.
      sta   drivesChecked+3  ignore drive D.
      lda   #0
      sta   drivesChecked+0  ;allow drive A.
      sta   drivesChecked+1  ;allow drive B.
      jsr   CkOtherDrives    ;and search them.
      bit   goodflag
      bmi   20$
      jmp   FileNotAvail     ;do a geoSHELL error routine.
 20$
      ...
      ...         ;file is found,
      ...         ;so continue on...
      ...
```

See Also: CkPresdrive, CkThisDrive, CkAllDrives, CkPathOrder, CkPath, CkForDisk, FindParNameCkPath

Function: Check to see if a file is on the path partition.

Pass: a2 - pointed to a null-terminated filename

Return: goodflag - set if successful.

DRVFOUNDON - contains the device number (8-11) of the path device if successful.

pathload - bit 7 set if successful.

a2 - will point to the null-terminated filename.

path partition - will be left open.

dirEntryBuf - will contain the directory entry for the file.

Destroys: no guarantees.

Description: Call this routine to check the path partition for a desired filename. Check goodflag first after calling this routine. If the file was found on the path partition, then goodflag will be set. Once you have finished with the file, be sure to call ResetFromPath to put the device back to the partition it was in prior to calling this routine. DRVFOUNDON will contain the device number (8-11) of the path device. If a path has not been defined with the path command, or if the defined path does not exist, the routine will exit through nogood.

Example:

```
      ...
      jsr   CkPath
      bit   goodflag
      bmi   10$
      rts         ;goodflag will still be zero.
 10$
      ...         ;do whatever you need to do
      ...         ;with the file here.
      ...
      jsr   ResetFromPath
      jmp   yesgood    ;tell the calling routine
                  ;that this routine was
                  ;was successful.
```

CkPathOrder

Function: Search a path partition for a file, but first check all available ramdisks, then the path partition, and then all remaining drives.

Pass: a2 - pointer to a null-terminated filename.

Return: goodflag - set if file is found.

DRVFOUNDON - contains the device number (8-11) of the device the file is found on.

curDevice and curDrive - drive file is found on will be open.

PRESDRIVE - unchanged.

pathload - bit 7 set if file is found on the path partition.

a2 - will point to the null-terminated filename with any wildcards converted to the characters they represent.

dirEntryBuf - will contain the directory entry for the file.

Destroys: no guarantees.

Description: This routine will search all available drives for a file, including the path partition. The ramdisks get checked first, then the path partition gets the next priority, and then all remaining drives will be searched. As soon as the file is found, the search will end through yesgood and the drive the file is found on will be open. If the file is not found, then the routine will exit through nogood and the currently active drive as the user sees it will remain the open drive.

Example:

```
      ...        ;a2 already points to a filename.
      jsr   CkPathOrder;search for the file, including
                 ;the path partition.
      bit   goodflag   ;did we find the file?
      bmi   10$   ;branch if so.
      jmp   FileNotAvail     ;do a geoSHELL error routine.
 10$
      ...        ;file is found,
      ...              ;so continue on...
      ...
      ...        ;before finishing, be sure to
      ...        ;close the path partition.
      jsr   ResetFromPath    ;in case the file was found             ;on
the path partition.
      ldx   NUM_IN_NAME;how big was the filename?
      inx        ;add one for a space.
      txa        ;put in in a.
      jsr   IncCmdPointer    ;and point the command pointer
                  ;past our command.
      jmp   ExitCommand ;exit cleanly.
```

See Also: CkPresdrive, CkThisDrive, CkOtherDrives, CkAllDrives, CkPath, CkForDisk, FindParName

CkPresdrive

Function: Check the currently active drive for a file.

Pass: a2 - null terminated filename.

Return: goodflag - set if file is found.

DRVFOUNDON - device number of the current drive where file is found.

curDrive - the current drive where the file is located is now open.

PRESDRIVE - unchanged (the current drive)

a2 - points to the filename converted to the actual name, any wildcards are converted to the characters they represent.

dirEntryBuf - directory entry for this file.

Destroys: no guarantees

Description: This routine will search the currently active drive for the filename that a2 points at. Upon return, the first thing to test is goodflag. If goodflag is cleared, then the file was not found for whatever reason. If goodflag is set, then the file exists on the drive.

Example:

```
      ...
      ...
      jsr   ParamName   ;point a2 at a filename the
                  ;user typed in.
      bit   goodflag    ;is there a filename?
      bmi   10$   ;branch if so.
 5$
      jmp   DoError     ;display an error.
 10$
      jsr   CkPredrive  ;look for the file.
      bit   goodflag    ;does the file exist?
      bpl   5$    ;branch if not.
      ...
      ...         ;continue on...
      ...
```

See Also: CkAllDrives, CkThisDrive, CkOtherDrives, CkPathOrder, CkPath, CkForDisk, FindParName

CkTerminators

Function: Check if the value is one of the geoSHELL text terminators.

Pass: a - value to test.

Return: a - unchanged.

goodflag - set if a terminator

Destroys: nothing

Description: This will test the value in the accumulator and return through nogood or yesgood depending on the value. If the value is either a space, an up-arrow, a CMDR up-arrow or a null byte, then goodflag will be set because it will exit through yesgood. The catch here is that a space character will also be considered a valid terminator by this routine. geoSHELL uses this because a command with no parameters may be terminated by a space. If you wish to test for a terminator and ignore a space, then use CkTermNoSpace.

Example:

```
     ...
     LoadW r0,#textBuffer   ;point at our buffer.
     ldy   #0
 10$
     lda   (a4),y      ;get a byte from input.
     jsr   CkTerminators    ;find the first space
     bit   goodflag   ;or end of parameter.
     bmi   20$   ;branch if terminator.
     sta   (r0),y      ;otherwise store byte.
     iny         ;point y at next byte.
     cpy   #80   ;let's limit to 80 bytes.
     bne   10$   ;loop back for more.
     iny         ;point at next byte
     tya         ;transfer value to a.
     jsr   IncCmdPointer    ;set for the next command
                 ;in line after ours.
     ...
     ...         ;now do what we want with
     ...         ;the text in our buffer.
     ...
```

See Also: CkTermNoSpace

CkTermNoSpace

Function: Check if the value is one of the geoSHELL text terminators except for a space.

Pass: a - value to test.

Return: a - unchanged.

goodflag - set if a terminator.

Destroys: nothing

Description: This will test the value in the accumulator and return through nogood or yesgood depending on the value. If the value is either an up-arrow, a CMDR up-arrow or a null byte, then goodflag will be set because it will exit through yesgood.

Example:

```
     ...
     ldx   #0
 10$
     lda   ParamTable,x     ;get a byte from ParamTable.
     jsr   CkTermNoSpace    ;check if a terminator?
     bit   goodflag   ;well?
     bmi   20$   ;branch if end of parameter.
     inx         ;point at the next byte.
     cpx   #16   ;let's only allow 16 bytes.
     bne   10$   ;loop back for more, maybe.
     jmp   DoError    ;do an error of some sort.
 20$
     inx         ;point just past our
     txa         ;parameter,
     jsr   IncCmdPointer    ;at the next command.
     ...
     ...         ;and continue on...
     ...
```

See Also: CkTerminators

CkThisDrive

Function: Check a drive for a file.

Pass: a2 - null terminated filename.

a - device number (8-11) of drive to search.

Return: goodflag - set if file is found.

DRVFOUNDON - device number of the drive if file is found.

curDrive - the drive if file is found, is now open.

PRESDRIVE - unchanged

a2 - points to the filename converted to the actual name, any wildcards are converted to the characters they represent.

dirEntryBuf - directory entry for this file.

Destroys: no guarantees

Description: This routine will search a specific drive for the filename that a2 points at. Upon return, the first thing to test is goodflag. If goodflag is cleared, then the file was not found for whatever reason. If goodflag is set, then the file exists and the desired drive will be the open drive. PRESDRIVE will still reflect the drive that the user thinks is currently active.

Example:

```
;this example will use a drive specifier that the user types in,
;plus a filename also entered by the user and then check the
;desired drive for that file.
;the command might look like:  command b filename

      ...
      ...
      lda   ParamTable+0     ;get the first parameter
            ;from the user.
      jsr   CkDriveLetter    ;convert it to a device number.
      bit   goodflag   ;is it a device number?
      bpl   5$   ;branch if not.
      sta   devNum     ;save it for now.
      lda   #2   ;point past the drive
      jsr   IncCmdPointer    ;specifier.
      jsr   ParamName  ;and get the filename the
            ;user typed in.
      bit   goodflag   ;is there a filename?
      bmi   10$   ;branch if so.
 5$
      jmp   DoError    ;display an error. 10$
      lda   devNum
      jsr   CkThisDrive ;look for the file.
      bit   goodflag   ;does it exist?
      bpl   5$   ;branch if not.
      ...
      ...        ;continue on...

devNum:
      .block     1
```

See Also: CkPresdrive, CkAllDrives, CkOtherDrives, CkPathOrder, CkPath, CkForDisk, FindParName

ClearBoth

Function: Clear the current line on the screen and in the memory buffer.

Pass: nothing

Return: nothing

Destroys: a, x, r0, r1, r2L, r11

Description: geoSHELL maintains an internal buffer that represents the characters that the user sees in the geoSHELL window. This is how geoSHELL is able to restore a window when switching screens, for instance, or returning from a Desk Accessory. If your command intends to work directly with the data that is on the screen as well as the data in the buffer, you can clear both the current line that the cursor is on as well as the internal memory buffer that corresponds to it by calling this routine. Normally, there is no need to work directly like this. geoSHELL does a pretty good job of maintaining the text and buffers while your command can do other things.

Example:

```
      ...
      jsr   ClearBoth   ;clear the data on the line
                  ;the cursor is on.
      ...
```

See Also: ResetScreen, ClearScreen, ClearWindow, EraseWindow, ReDoWindow, DispLetter, DriveLetter, DsplyLine, DsplyString, CarrReturn, set_XY_pos, save_XY_pos, ClearLine, ClearRemainder, ReDisplayWindow, DispText, unsetPrompt

ClearLine

Function: Clear the current line in the geoSHELL window.

Pass: r1H - vertical pixel location set by calling set_XY_pos.

Return: nothing

Destroys: nothing

Description: This is mostly an internal routine for clearing the current line in the geoSHELL window. It does not clear the corresponding memory buffer.

Example:

```
      ...
      jsr   set_XY_pos
      jsr   ClearLine   ;clear the line
                  ;the cursor is on.
      ...
```

See Also: ResetScreen, ClearScreen, ClearWindow, EraseWindow, ReDoWindow, DispLetter, DriveLetter, DsplyLine, DsplyString, CarrReturn, set_XY_pos, save_XY_pos, ClearBoth, ClearRemainder, ReDisplayWindow, DispText, unsetPrompt

ClearRemainder

Function: Clear the window below the current cursor position.

Pass: nothing

Return: nothing

Destroys: a, r1H, r11

Description: If your command displays a series of text to the window, you may desire to clear all text below the current cursor position for a cleaner display. The 'type' command does this as well as some other commands. This makes it easier for the user to read what it being put on the screen. The internal buffers are not cleared, only the visible text on the screen. As new text is added (through the normal geoSHELL text displaying routines), it will show up on the screen and also replace what is in the memory buffers.

Example:

```
      ...
      jsr   ClearRemainder
      ...
```

See Also: ResetScreen, ClearScreen, ClearWindow, EraseWindow, ReDoWindow, DispLetter, DriveLetter, DsplyLine, DsplyString, CarrReturn, set_XY_pos, save_XY_pos, ClearBoth, ClearLine, ReDisplayWindow, DispText, unsetPrompt

ClearScreen

Function: Clear the entire screen.

Pass: nothing

Return: nothing

Destroys: no guarantees

Description: This routine will clear the screen with the default background pattern as established by the command 'backpatt'. By using this routine, geoSHELL will take care of whatever screenmode is being used. This also takes care of the proper screen coloring if the user is in 80 column color mode. By adjusting the above mentioned variables (back40pattern, etc) you can change the pattern that is used.

Example:

```
      ...
      jsr   ClearScreen
      ...
```

See Also: ResetScreen, ClearRemainder, ClearWindow, EraseWindow, ReDoWindow, DispLetter, DriveLetter, DsplyLine, DsplyString, CarrReturn, set_XY_pos, save_XY_pos, ClearBoth, ClearLine, ReDisplayWindow, DispText, unsetPrompt

ClearWindow

Function: Clear the geoSHELL window.

Pass: nothing

Return: nothing

Destroys: no guarantees

Description: Calling this routine will clear any text that is displayed in the geoeSHELL window. The buffer that is holding the text internally is also cleared and the cursor position is reset to be at the top left of the window. This is what takes place when the user hits SHIFT CLR/HOME.

Example:

```
      ...
      jsr   ClearWindow
      ...
```

See Also: ResetScreen, ClearScreen, ClearRemainder, EraseWindow, ReDoWindow, DispLetter, DriveLetter, DsplyLine, DsplyString, CarrReturn, set_XY_pos, save_XY_pos, ClearBoth, ClearLine, ReDisplayWindow, DispText, unsetPrompt

ClrKeyData

Function: Eliminate any detected keypresses.

Pass: nothing

Return: keyData gets reset to zero.

goodflag is always set by this routine.

Destroys: a, x, y

Description: Call this routine to clear out any keypresses that GEOS has detected. This might be needed when the user has pressed a key before your routine actually gets handled. After executing this routine, you can then get a valid keypress. This routine will also wait for the user to let go of the keyboard.

Example:

```
      ...
      ...        ;the bulk of your command.
      ...
      jsr   ClrKeyData  ;clear any existing keypresses
                 ;and wait for the user to let
                 ;go of the keyboard
      ...
      ...        ;no chance for the next
      ...        ;routine to get a wrong
      ...        ;keypress.
```

See Also: NoKeypress, YesKeypress, Wait

ClrTrnsName

Function: Clear a transient command's name from memory.

Pass: nothing

Return: curTransName - set to null.

Destroys: a, x

Description: When a transient command is loaded into memory and executed, it will remain in memory until either another command is loaded or if a resident command needs to use the area reserved for transient commands. In the latter case, the resident command will call ClrTrnsName to inform geoSHELL that no transient command is in memory. Then if the user enters that particular command again, geoSHELL will be forced to reload it from disk.

In most cases, a transient command will not need to call this routine, because it is advantageous to just let your command remain in memory should the user need to use it several times in a row. However, if your command is loaded from a VLIR file or does anything that would change itself to a state that can not be restarted properly, then call ClrTrnsName at any time before exiting your command. This way, your command will be loaded in fresh each time it is used.

Example:

```
      ...
      ...        ;some of the code in your
      ...        ;command might trash the
      ...        ;basic state of your command.
      ...
      jsr   ClrTrnsName ;remove your command's name
                 ;from memory.
      jmp   ExitCommand ;and finish.
```

ColorScreen

Function: Color an area of the 80 column screen.

Pass: r0 - table of compressed locations.

r1 - table of various color combinations to use.

Return: goodflag - always zero (not an error).

Destroys: a, x, y, r0, r1

Description: This routine will put colors on the 80 column screen of the 128. Your routine must test the current screenmode and videomode to be sure that a 64 is not being used or that a 128 is not in monochrome mode before calling this routine. ColorScreen does not check it for you.

Point r0 to a table of compressed bytes and r1 to a table of color combinations. This color table may contain as many as 256 different combinations of foreground and background colors. Each character area can contain it's own foreground and background color. You might say that the table that this routine uses could be called a 'Color Scrap' since it can color a portion of the screen.

It is not a good idea to put colors below the 176th scanline. geoSHELL does not normally use this lower area of the screen. This is so that 128s with only 16K of video memory may also use the color mode.

In the appendix, you will find an explanation of how to create the table of compressed bytes and the color tables for this routine.

Example: Refer to the appendix for a full explanation and example for using this routine. You will also find the complete source code for the command 'color80' included with this package. Naturally, this routine is used in that command, since it was originally developed specifically for color80.

See Also: FixColors

ConvertK

Function: Convert a 16-bit number to an ascii string representing kilobytes.

Pass: r4 - 16 bit number to convert.

Return: Asc_String - a five-character ascii decimal string with leading zeros converted to spaces and null-terminated.

Destroys: a, x, y, r4, r5, r8, r9

Description: This routine is used by the 'dir' command when it displays the 'K Bytes Free' message after displaying a directory. The routine will take the value in r4 and convert it to a decimal string after dividing it by 4 and rounding it down to the nearest whole number (4 blocks equals 1 Kbyte). If the value originally passed in r4 is less than four, then the decimal string will be rounded up to 1. In the case of the 'dir' command, it loads r4 with the number of blocks free on a disk and calls this routine to create the number of kilobytes free. The resulting null-terminated decimal string can be found at Asc_String.

Example:

```
;this example will display the kbytes free on the current drive.

ShowFree:
      LoadW r5,#curDirHead   ;get the number of blocks
      jsr   CalcBlksFree     ;free on the current drive.
      jsr   ConvertK   ;convert it to the nearest K.
      LoadW r0,#Asc_String   ;point r0 to the string.
      lda   #(IN_TWO|TR_ONE) ;indent two + trailing space.
      jsr   OtherMessage     ;put the string on the screen.
      LoadW r0,#kbyteText    ;along with 'KBytes Free'.
      lda   #TR_CR     ;and a carriage return.
      jmp   OtherMessage     ;do it.

kbyteText:
      .byte "KBytes Free",0
```

See Also: Asc_3_Byte, Asc_Byte, Asc_BCD, ByteNZ_Ascii, ByteWZ_Ascii

ConvToAscii

Function: Convert PetASCII text to ASCII text.

Pass: r0 - start of text to convert.

endoftext - pointer to one byte past the last byte of text.

Return: r0 - (unchanged) points to converted text.

endoftext - adjusted to point to a null byte at the new end of text plus one.

r2 - pointing at same location as endoftext.

Destroys: a, x, y, r2

Description: This routine will take any length of PetASCII text and convert it to ASCII text. Just point r0 to the start of the buffer holding the text and point endoftext at the byte following the end of the text. Call this routine and that text will be converted to ASCII. Unrecognized bytes are removed from the buffer. Upon return, r0 will still be pointing at the same starting point, but endoftext might point to a different spot making the buffer smaller in size. The buffer will never grow, since characters will never be added, only removed. The bytes that may get removed are usually control codes such as might be created by various wordprocessors. The text after conversion will also have any null bytes removed and a null terminator will be added at the end.

Example:

```
      ...
      LoadW r0,#textBuffer
      LoadW endoftext,#(bufEnd+1)
      jsr   ConvToAscii
      ...
```

See Also: AdjTxtPointer, Message, OtherMessage, DsplyLine, DsplyString, DispText, ClearLine, ClearBoth, ClearRemainder, StripRulers, set_XY_pos, save_XY_pos, MskCtrlCodes, CarrReturn, CkESC_RULER

DateDisplay

Function: Display the current date.

Pass: nothing

Return: nothing

Destroys: no guarantees

Description: This routine merely displays the current date to the geoSHELL window just as if the user had typed it in without any parameters.

Example:

```
      ...
      jsr   DateDisplay ;display the date.
      ...
```

See Also:

Dir

Function: Display the directory of the current drive.

Pass: ParamTable - parameters just as the user might enter.

kboardoff - (optional) set bit 7 to ignore the keyboard. Set bit 6 to ignore just the STOP key.

Return: nothing

Destroys: no guarantees

Description: This will display a directory of the currently active drive just as if the user typed in the 'dir' command. The format command accesses this routine after it finishes formatting a disk so that the user can see that the disk formatting is complete. You would want to load ParamTable with any parameters you wish Dir to work with just as if the user typed them in at the keyboard. If you place an ascii '6' at ParamTable+0 and a null byte at ParamTable+1 and call this routine, a directory of all the applications on the current drive will be displayed. If ParamTable+0 contains a null byte, then all of the files on the disk will be displayed just as if the user typed 'dir' without a parameter. If the user presses the STOP key while the directory is scrolling, your command will terminate and control will return to the user. You can turn off the keyboard by setting bit 7 of kboardoff. However, this will not allow the user to pause the display. So, instead, you can set bit 6 to just ignore the STOP key. This way, Dir will always return to your command. Just remember to reset kboardoff to zero when finished.

Example:

```
;this example will display all files ending with '.src'.

ShowFiles:
      ldx   #6
 10$
      lda   fileText,x
      sta   ParamTable,x
      dex
      bpl   10$
      jmp   Dir

fileText:
      .byte "*.src",0
```

See Also: CkForDisk, MissDisk, Status

DispLetter

Function: Display the drive letter of the currently active drive.

Pass: nothing

Return: nothing

Destroys: a, x, y, r0-r15

Description: This is primarily an internal geoSHELL routine. It will simply display the letter (A,B,C or D) of the currently active drive at the left of the current line the cursor is on. The cursor will now be in the third character position on the current line.

Example:

See Also: DriveLetter

DispText

Function: Display ascii text to the geoSHELL window up to a maximum of the width of the window.

Pass: r0 - point to the start of ascii text.

endecho - zero if only one line of text is needed, or set bit 7 if you need to test for end of text.

typeset - bit 7 set will handle carriage returns and tabs and ignore up-arrow terminators, while if cleared, then carriage returns and tabs are ignored, and an up-arrow will clear endecho.

kboardoff - normally zero. Set to 128 to turn the keyboard off while displaying text. (reset it to zero when finished)

Return: goodflag - meaningless

r0 - unchanged

endecho - bit 7 cleared if end of text reached

textpointer - offset pointer to next word to display

Destroys: a, x, y, r1-r15

Description: Display any amount of text in the geoSHELL window. Both the echo and type commands use this routine. Just point r0 at the start of the text you wish to display and call this routine. If you wish to have carriage returns and tabs recognized, then set typeset to 128. Also, before calling this routine, set endecho to 128, because DispText will clear it to zero if the end of text is reached. End of text is a null byte. This allows you to repeatedly call DispText to put text in the window. Of course, you would have to keep updating r0 between calls to DispText since it does not change. There is a routine that will do this for you. Call AdjTxtPointer and r0 will point to the next word to display on the next line. DispText will automatically do a carriage return when it reaches the right side of the window, so that the next call will display text on the next line down. If the bottom line is reached, the text will be scrolled. While this is taking place, the keyboard is active so that the user may pause or halt the text displaying with the CONTROL or STOP keys.

Example:

```
;this example will display ascii text that is contained in
```

```
;a buffer of your choice. In this example we will call the
;buffer 'textStart'.
DoText:
     LoadW r0,#textStart    ;set r0 to the start
                ;of your text.
     LoadB endecho,#128     ;initialize a flag.
     LoadB typeset,#128     ;do tabs and carriage
                ;returns.
 10$
     jsr   DispText   ;display up to 80 columns
                ;of text.
     jsr   AdjTxtPointer    ;advance a pointer to
                ;some more of the text.
     bit   endecho    ;have we reached the end?
     bmi   10$   ;branch if not.
     rts
```

See Also: AdjTxtPointer, Message, OtherMessage, DsplyLine, DsplyString, CarrReturn, ClearLine, ClearBoth, ClearRemainder, StripRulers, set_XY_pos, save_XY_pos, MskCtrlCodes, ConvToAscii, CkESC_RULERDoRun

Function: Load and run a file.

Pass: a2 - pointer to null terminated filename.

Return: not applicable.

Destroys: not applicable.

Description: You can use this routine if you desire to load and run any file that is capable of doing so. All you need to do is point a2 to a null terminated filename. You don't even have to know where the file is located, DoRun will find it for you. DoRun will take over and cause your command to cease it's function. If any errors are encoutered, control will return to the user. For this reason you might only want to call this routine if the geoSHELL window is displayed.

Example:

```
;this routine takes the filename that the user typed in and
;then passes control to DoRun. If the file is found on one of the
;available drives, it will be loaded and run.

     ...
     jsr   ParamName  ;get the filename.
     bit   goodflag   ;is it a valid filename?
     bmi   10$   ;branch if so.
     jmp   DoError    ;otherwise display an error.
 10$
     jmp   DoRun ;find it and run it.
```

See Also:

DriveLetter

Function: Display the drive letter of the active drive and turn the cursor on.

Pass: nothing

Return: nothing

Destroys: a, x, y, r0-r15

Description: Most commands have no use for this routine. It is mostly an internal routine that is accessed just before control is returned to the user. However, since geoSHELL gives your command control of the system, you could use this to also return control to the user and interact with the user instead of geoSHELL's own main processor doing so. You would need some sort of routine to call for user input. This routine also gets the cursor blinking.

Example:

See Also: DispLetter

DsplyLine

Function: Display a line of text to the geoSHELL window beginning at the left side of the current line.

Pass: a0 set to the null terminated string to be displayed.

Return: cur_line - updated to reflect the line the cursor is on. A carriage return is performed after the line is displayed.

Destroys: no guarantees.

Description: This routine should only be used by a command that might need a more direct approach for displaying text to the geoSHELL window. The 'type' command uses it and so does geoSHELL's main display routines. For general message displaying, use Message and OtherMessage. a0 is normally used by internal routines for keeping the cursor in sync with the internal memory buffers. A carriage return is always performed after the line of text is displayed and if the cursor is on the last line, the text will scroll up one line.

This routine also calls CkKeyboard to check for the CONTROL or STOP key. So, if you wish this feature turned off, set bit 7 of kboardoff. If you wish to allow the user to still be able to use the CONTROL key to pause text, then set bit 6 of kboardoff and the STOP key will be ignored.

Example:

```
      ...
      lda   PRESDRIVE  ;get the real device number
      jsr   GetRealDriveNum  ;of the currently active drive.
      bit   goodflag   ;is it a real drive?
      bmi   10$   ;branch if so.
      LoadW r0,#notReal ;inform the user that this
      lda   #(IN_TWO|TR_CR)  ;won't work on this drive.
      jsr   OtherMessage
      jmp   NoMoreCmds ;and exit.
 10$
      jsr   PushStuff  ;save a bunch of stuff.
      jsr   InitForIO  ;pop out of GEOS.
      LoadW r0,#iText  ;send an init command
      LoadB cmdlength,#2     ;to the drive.
      lda   REALDRIVE  ;get the device number.
      jsr   SendCmd    ;and send the command.
      lda   REALDRIVE  ;read the error channel.
      jsr   ReadChannel
      jsr   DoneWithIO ;pop back into GEOS.
      LoadW a0,#EnOfShell    ;and display the text
      jsr   DsplyLine  ;from the error channel.
      jsr   PopStuff   ;restore a bunch of stuff.
      ...   ...

notReal:
      .byte "This Is Not A Real Drive!",0
iText:
      .byte "I0"  ;initialize command (be sure
                  ;to use an upper-case i here)
```

See Also: AdjTxtPointer, Message, OtherMessage, DispText, DsplyString, CarrReturn, ClearLine, ClearBoth, ClearRemainder, StripRulers, set_XY_pos, save_XY_pos, MskCtrlCodes, ConvToAscii, CkESC_RULER

DsplyString

Function: Display a string at the current cursor position.

Pass: a0 - pointing to the null-terminated string to be displayed.

Return: cur_X_pos holds the new horizontal pixel location of the cursor.

cur_Y_pos holds the new vertical pixel location of the cursor.

Destroys: no guarantees.

Description: This routine is mostly an internal routine and normally need not be used. It will display a null-terminated string pointed to by a0 to the screen beginning at the current cursor position. The cursor will be left at the end of the string on the screen. This routine will check the bytes before they are printed and any bytes that cannot be printed on the screen by GEOS will be converted to '?'.

Example:

See Also: AdjTxtPointer, Message, OtherMessage, DispText, DsplyLine, CarrReturn, ClearLine, ClearBoth, ClearRemainder, StripRulers, set_XY_pos, save_XY_pos, MskCtrlCodes, ConvToAscii, CkESC_RULER
EraseWindow

Function: Clear only the geoSHELL window

Pass: nothing

Return: nothing

Destroys: no guarantees

Description: This is called by ClearWindow and is usually not needed by most commands unless your command is not dealing entirely with text input and/or display. This routine does not clear the buffer that is holding the text that was displayed in the window. So, in some cases this routine could be used without losing what was in the buffer, and a subsequent call to ReDoWindow will therefore restore the text to the screen.

Example:
```
      ...
      jsr   EraseWindow
      ...
```

See Also: ResetScreen, ClearScreen, ClearWindow, ClearBoth, ReDoWindow, DispLetter, DriveLetter, DsplyLine, DsplyString, CarrReturn, set_XY_pos, save_XY_pos, ClearLine, ClearRemainder, ReDisplayWindow, DispText, unsetPrompt
execStartup

Function: Execute a string of commands located beginning at StUpBuffer.

Pass: nothing (other than to load StUpBuffer with the commands).

Return: not applicable

Destroys: not applicable

Description: Fill the buffer at StUpBuffer with a group of commands and call this routine to execute them. This does not return to your command. When all commands in the buffer have been executed, control will return to the user.

Example:

See Also:

ExitCommand

Function: Exit a transient command cleanly.

Pass: nothing

Return: not applicable

Destroys: not applicable

Description: When everything in your command goes as expected and you are ready to terminate it and return control to geoSHELL, call this routine. If you simply perform an 'rts', then your command will put the GEOS mainloop in control without the user being able to have access to geoSHELL's keyboard input. ExitCommand puts things back like they were before your command was called.

In some cases, you may want to do an rts somewhere within your command if you wish to have it working with the GEOS main loop instead of geoSHELL doing so. In this case, your command really isn't finished yet, but perhaps you have set up your own user input routines for mainloop to handle. geoSHELL contains certain routines that can do many things for you, but some special situations may need new routines to do different things. But when your routine is finally finished, exit through ExitCommand and geoSHELL will process any further commands that the user may have entered.

Example:
```
      ...
      ...        ;the bulk of your command.
      ...
      jmp   ExitCommand ;exit cleanly.
```

See Also: NoMoreCmds

FileNotAvail

Function: Display a 'Filename Not Available!' message and return control to the user.

Pass: a2 - pointing to a null terminated filename.

Return: not applicable

Destroys: not applicable

Description: If your command fails to find a file that the user requests, you could end your command by jumping through this routine. If a2 is still pointing at the null-terminated filename, this routine will display the filename on the screen, followed by 'Not Available!'. Control is then returned to the user instead of your command.

Example:

```
      ...
      jsr   ParamName   ;get the filename.
      bit   goodflag    ;is it valid?
      bmi   10$   ;branch if so.
      jmp   MissgFilename    ;display a geoSHELL error
                  ;and exit.
 10$
      jsr   CkAllDrives ;find the file.
      bit   goodflag    ;does it exist?
      bmi   20$
      jmp   FileNotAvail     ;display error and exit.
 20$
      ...
      ...         ;continue on...
      ...
```

See Also: Message, NoMoreCmds, LoadProblem, OtherMessage, FileNotAvail, MissDisk, MissgFilename, NotFound

FindCMDType

Function: Find the location of a CMD device.

Pass: a - an upper-case letter corresponding to the desired device to locate: F, H, or R.

```
        F = FD Series
        H = HD Series
        R = RamLink or RamDrive
```

Return: goodflag - bit 7 set if the requested CMD device was found.

curDrive - desired CMD device is opened

REALDRIVE - real device number of the CMD device

Destroys: no guarantees

Description: To find out if a particular CMD device is being used, load the accumulator with an F, H, or R and call this routine. If goodflag is set upon return, then the device was found. This routine also calls GetCMDType, and so the variables that are set with that routine will also be in effect.

Example:

```
;this example will locate an FD Series drive if one is being used.
;this same routine can be modified to locate any CMD drive.
;If a RamLink or RamDrive is being searched for and found, then
```

```
;the variables are set by GetCMDType can be checked:
;cmdtype+0 will contain an 'R'
;and cmdtype+1 will have either an 'L' or a 'D'
;while cmdtype+2 will contain either a 3 or a 4.

FindFD:
      lda   #'F'  ;let's find the FD Drive.
      jsr   FindCMDType
      bit   goodflag
      bmi   10$
      LoadB FDdrive,#0  ;set variable to zero to
                  ;indicate no FD.
      jmp   NoFD  ;do whatever, if not found.
 10$
      MoveB REALDRIVE,FDdrive
      rts         ;goodflag is still set
                  ;and FDdrive now contains
                  ;the device number of the FD.
FDdrive:
      .block     1
```

See Also: GetCMDTypeFindParName

Function: Search the current drive for a desired file from a parameter the user entered.

Pass: nothing

Return: If file not found, routine does not return.

If routine returns then file is found on the current drive.

a2 - now points to a null-terminated filename.

dirEntryBuf - contains the directory entry for the file.

Destroys: no guarantees

Description: You could use this routine when you wish to search the current drive for a file that the user desires your command to act on. This routine will take care of calling ParamName for you and call the necessary routines for searching the drive. If the file is not found, however, the routine will display an error that the file is not available and jump through NoMoreCmds, which will return control to the user. If the user did not enter a parameter that could possibly be a filename, this routine will also handle that error for you without returning. If the routine returns, then obviously the file is found, and your command may continue.

Example:

```
      ...
      jsr   FindParName ;find the desired file.
      ...         ;file is found, so continue...
      ...
```

See Also: CkAllDrives, CkThisDrive, CkOtherDrives, CkPathOrder, CkPath, CkForDisk, CkPresdrive

FindRamLink

Function: Find a RamLink or RamDrive if one is currently being used.

Pass: InitForIO must be called first!

Return: goodflag - set if an RL/RD is found on the system.

RAMLINK - holds the real device number of the RL or RD.

cmdtype - (two bytes) holds the characters 'RL' or 'RD'

cmdtype+2 - holds a 3 for RamDrive or a 4 for RamLink.

Destroys: no guarantees

Description: This routine will check to see if a RamLink or RamDrive is connected and enabled, even if it is not being used by GEOS. The only way this routine will not find the device is if the user has given it a device number of 14. geoSHELL always ignores device number 14 in order to prevent a possible crash should the user have a Xetec Super Graphics Gold printer interface connected. Don't forget to call InitForIO before calling this routine.

Example:

```
;this example routine will find a RamDrive. If a RamDrive is found
;then the variable RAMLINK will contain the device number of the
;RamDrive.

FindRD:
      jsr   InitForIO   ;pop out of GEOS.
      jsr   FindRamLink ;look for an RL or RD.
      jsr   DoneWithIO  ;pop back in.
      bit   goodflag    ;did we find one or the other?
      bpl   20$   ;branch if not.
      lda   cmdtype+2   ;check the device type.
      cmp   #3    ;is it an RD?
      bne   10$   ;branch if not.(if it was an RL)
      rts         ;goodflag already set.
 10$
      LoadB RAMLINK,#0  ;in case a RamLink was found,
            ;because we were looking for
            ;a RamDrive instead.
 20$
      jmp   nogood      ;signal to calling routine
            ;that no RamDrive is found.
```

See Also: GetCMDType, FindCMDType, GetRealDriveNumFixColors

Function: Restore the default colors to the 40 column screen.

Pass: nothing

Return: Colors on screen restored.

Destroys: a, y, r0, r1, r2L

Description: geoSHELL uses this routine to restore the colors on the 40 column screen whenever a desk accessory exits back to geoSHELL. Some DA's alter the screen colors without restoring them upon exit. This is actually allowed and is left up to the application to restore the colors as well as any other data displayed on the screen. If your command alters the colors for it's own purpose, use this routine to restore them, unless the intention is to alter the colors permanently. In this case, you would also want to change the GEOS variable, screencolors. The high nybble of screencolors is the foreground color (0-15) while the low nybble is the background color (0-15).

Example:

```
      ...
      jsr   FixColors   ;restore 40 column color
      ...
```

See Also: ColorScreen

GetCMDType

Function: Check if a device is a CMD device and if so, what type.

Pass: a - real device number of device to check. (remember that a RamLink or RamDrive can be 12 or higher if it is being used as a RAM_1581)

InitForIO must be called before calling this routine.

Return: goodflag - set if a CMD device.

cmdtype - (two bytes) contains 'FD', 'HD', 'RD', or 'RL'.

cmdtype+2 - contains 1, 2, 3, or 4.
1 = FD Series drive.
2 = HD Series hard drive.
3 = PP RamDrive.
4 = RamLink.

Destroys: a, x, y, r0

Description: After calling InitForIO, you can call this routine to find out if a particular drive is a CMD device and exactly what type it is. Upon return, if goodflag is set, then the device is a CMD device.

Then you can proceed to check the bytes at cmdtype to see which CMD device it is.

Example:

```
;this particular example will check if drive A is a CMD HD.

CheckHD:
     jsr  InitForIO  ;first, pop out of GEOS.
     lda  #8   ;drive A.
     jsr  GetCMDType ;is it a CMD device?
     jsr  DoneWithIO ;pop back into GEOS.
     bit  goodflag   ;so was it a CMD?
     bpl  90$  ;branch if not.
     lda  cmdtype+2  ;check the type byte.
     cmp  #2   ;is it an HD?
     bne  90$  ;branch if not.
     rts       ;or exit with goodflag
               ;still set.
 90$
     jmp  nogood     ;tell the calling routine
               ;that this is not an HD.
```

See Also: FindRamLink, FindCMDType, GetRealDriveNum

GetHeader

Function: Load the header block of a file into fileHeader.

Pass: dirEntryBuf with a file's directory entry.

Return: fileHeader contains the file's header block.

r7 - start address of the file.

r1 - track and sector of first data block, or if VLIR file, then T/S of index block.

Description: This routine will not return if any errors are encountered. If an error such as file not found occurs, then an error message will be displayed and control will return to the user. If a header block is successfully loaded, then control returns to your command and you may continue.

Example:

```
     ...
     jsr  ParamName  ;get the filename.
     jsr  CkAllDrives ;find the file.
     bit  goodflag   ;did we find it?
     bmi  10$  ;branch if so.
     jmp  FileNotAvail    ;tell the user, no file.
 10$
```

```
      jsr   GetHeader   ;no need to check for errors,
                  ;routine returns if OK.
      ...
      ...         ;so we can continue...
      ...
```

See Also:

GetMess

Function: Set a pointer to a message.

Pass: r0 - pointing to the start of a group of null-terminated message strings.

x - desired message number (1-255)

Return: r0 - pointing to the desired message.

Destroys: a, x, y

Description: This is part of the routine that 'Message' uses to locate a particular message within a whole group of messages. You can set up a table of messages, with each message being null-terminated. Set x to the message you need and r0 to the start of the group of messages and then call this routine. Upon return, r0 will be pointing at the desired message. A maximum of 255 messages can be contained in a table.

Example:

```
;this example will display the error message corresponding to the
;value in x.

DoError:
      LoadW r0,#errorMessages;point to the start of
                  ;message table.
                  ;x is set before calling this.
      jsr   GetMess     ;point to the desired message.
      lda   #(IN_TWO|TR_CR)  ;indent two and trailing CR.
      jmp   OtherMessage     ;display the message.

errorMessages:
      .byte "This Would Be Error Number 1!",0
      .byte "And Error Number 2!",0
      .byte "Or Number 3!",0
      .byte "This Can Be Error Number 4!",0
      .byte "Number 5 Might Be Here!",0
```

See Also: GetMessage, Message, OtherMessage
GetMessage

Function: Point at one of geoSHELL's built-in messages.

Pass: x - desired message number.

Return: r0 - pointing to the message

Destroys: a, x, y

Description: This routine will simply point r0 to the desired message contained in the standard geoSHELL message table, depending on the value in x. There really isn't much use for this command, although it could be used to locate a particular message and then alter the message. The only problem with this is that the message must contain the exact same number of characters, since if any null bytes are added, then they are treated as blank messages and any following messages will be out of sync when geoSHELL tries to display one of them. In other words, the wrong message might get displayed, thereby creating confusion for the user.

Example:

See Also: GetMess, Message, OtherMessage

GetRealDriveNum

Function: Get the 'real' device number of a drive.

Pass: a - device number 8-11 of drive.

Return: goodflag - set if device is a real drive or a device that uses some form of CBM DOS such as a RamLink.

REALDRIVE - device number of the device.

curdrvtype - drive type of device as GEOS sees it.

RAMLINK - 0 if device is not a RamLink or RamDrive or if so it will contain a copy of REALDRIVE.

Destroys: a, x, y, r0-r15

Description: If working with the built-in DOS in the drive is desired, you need to know the device number before using standard kernal routines to access the device. Normally this is not a problem. However, it is possible that a RamLink or RamDrive might be used with 1581 partitions being used to emulate a RAM_1581. In these cases, the GEOS disk driver is controlling the device as a RAM device and not a DOS device. So, the device number of the device in question might be 12 or higher instead of the usual 8-11 that GEOS uses. This routine will test the device and return the correct device number for you in REALDRIVE.

This routine will also call PurgeAllTurbos. So, if you call another routine after this one that needs the turbo code purged in the drives, it will not be necessary to call PurgeAllTurbos.

Example:

```
      ...
      lda   PRESDRIVE   ;let's get the device number
                  ;of the currently active
                  ;drive.
      jsr   GetRealDriveNum
      bit   goodflag
      bmi   10$
      jmp   NotRealDrive     ;go do some sort of error.
 10$
      ...         ;now we have the real
                  ;drive number for sure
                  ;just in case a RamLink
                  ;is being used.
```

See Also:
ParamName

Function: Point r6 and a2 to the filename that the user entered.

Pass: nozero - set bit 7 of this variable if you do not want a zero placed at the end of the filename, otherwise simply ignore this variable (it is automatically reset to zero when the routine is finished), it is not necessary for your command to set it to zero.

Return: goodflag - set if what appears to be a valid filename was found.

r6 - points to a null terminated filename.

a2 - points to a null terminated filename.

WILDCARD - bit 7 set if any wildcard characters are found (* and/or ?).

r0 - if bit 5 of WILDCARD is set, this means that the user included an equals (=) sign somewhere in the parameter and r0 will be pointing at the first null-terminated filename while a2 and r6 will be pointing at the second one.

NUM_IN_NAME - contains the number of characters 1-16 that are in the filename.

Description: Call this routine when your command needs to have a filename follow as a parameter. If ParamName finds what appears to be a valid filename (up to a maximum of 16 characters), it will exit through yesgood. If you find that goodflag does not have bit 7 set, then the routine failed and you do not have any valid filename to process. Handle this accordingly.

When a command needs two filenames entered in a manner such as what the 'rename' command requires, then r0 will return pointing at the first filename and a2 will be pointing at the second.

Example: ren newname=oldname

In this case, both filenames will become null-terminated. You can tell that the user entered a parameter this way by checking bit 5 of WILDCARD. If it is set, then an equals sign was found.

If bit 7 of WILDCARD is set, then the user entered at least one wildcard in the filename. In the case of bit 5 being set, wildcards are only checked in the second filename, the one that a2 is pointing at.

Example:

;this example requires that the user enters two filenames similar;to the way that the rename command works.

```
     ...
     jsr   ParamName
     bit   goodflag
     bmi   10$
 5$
     jsr   MissgFilename
 10$
     bbrf  5,WILDCARD,5$    ;use a macro to test bit 5.
                ;did the user enter two names?
                ;it will branch if not.
     MoveW r0,firstName     ;save r0 for now.
     jsr   CkPresdrive ;check the current drive
                ;for the second filename.
     bit   goodflag   ;does the file exist?
     bmi   30$   ;branch if so.
 20$
     jsr   NotFound   ;tell the user, not found.
     jmp   NoMoreCmds ;and exit.
 30$
     MoveW a2,secondName    ;save this name for now.
     MoveW firstName,a2     ;now let's
     jsr   CkPresdrive ;look for the first filename.
     bit   goodflag   ;did we find it?
     bpl   20$   ;branch if not.
     ...
     ...        ;at this point we have
     ...        ;determined that both
     ...        ;filenames exist on the
     ...        ;currently active drive
     ...        ;and we may continue on...
     ...
     inc   NUM_IN_NAME ;get the length of the a2
     lda   NUM_IN_NAME ;filename plus on for a space.
     jsr   IncCmdPointer    ;point at the next command.
```

```
      jmp   ExitCommand;and exit cleanly.


firstName:
      .block     2
secondName:
      .block     2
```

See Also: FindParName

PopStuff

Function: Restore certain variables after a call to PushStuff.

Pass: nothing

Return: r0-r15, a0-a4 are restored to the values they held when PushStuff was called.

Destroys: nothing

Description: Refer to PushStuff for the basic purpose of these two routines. Do not call this routine except after you have already called PushStuff, but be sure to call it if you have called PushStuff. PopStuff restores various zero page locations from a special built-in stack that is maintained by geoSHELL. Treat it just like you would if you were to push a byte onto the processor's stack. One difference here though is that if you exit your command through NoMoreCmds, then this built-in stack is reset anyway and you would not need to call PopStuff.

Example:

```
      ...
      jsr   PushStuff   ;save some zero page registers.
      ...
      ...         ;you might have some code
      ...         ;in here that trashes the
      ...         ;contents of r0-r6 or whatever.
      ...
      jsr   PopStuff    ;restore the registers.
      ...
```

See Also: PushStuff, NoMoreCmds

PurgeAllTurbos

Function: Purge the turbo code from all the drives that are being used in order to perform low-level DOS routines.

Pass: nothing

Return: goodflag - could be either set or cleared, but in the case of this routine, it is meaningless.

Destroys: a, x, y, r0-r3

Description: Before performing any low level functions with the drives such as direct memory accessing, it is necessary to purge the turbo code in the drive so that the disk driver will restore it completely when EnterTurbo is called. If not, then your routines could possibly corrupt the turbo code that is in the drive and the driver would think that it is still intact.

This routine purges the turbo code from 'all' the drives so that you might be able to do low level functions with more than one drive at a time as you wish. geoSHELL's normal disk accessing will tell the disk drivers to restore the needed turbo code before any GEOS disk accessing. For that reason, it is not necessary that your command call EnterTurbo unless it will also need some GEOS disk accesses.

If you are dealing with the DOS in just one drive, then you could just call PurgeTurbo for that one device. However some of the routines that you call in geoSHELL, especially the ones that deal with the CMD devices, might deal with more than one drive and therefore needs the turbo code in all the drives to be purged. This routine will do that for you.

Example:

```
      ...
      jsr   PurgeAllTurbos   ;kill all the turbo code.
      jsr   InitForIO  ;pop out of GEOS.
      ...
      ...        ;do some low-level stuff.
      ...
      jsr   DoneWithIO ;pop back into GEOS.
      jmp   ExitCommand
```

See Also:
PushStuff

Function: Save certain zero page locations to a built-in stack.

Pass: nothing

Return: nothing

Destroys: nothing

Description: In some cases you might want to preserve the locations from r0-r15. Just call this routine and geoSHELL will save them on a special stack contained in memory. The locations from a0-a4 are also saved with this routine. Nothing gets trashed when calling this routine. a,x and y are not trashed. Use PopStuff to restore all of these locations. If you call PushStuff, be sure to call PopStuff, for the same reason you would do a pla after you do a pha instruction. If you exit your command through NoMoreCmds, this will be done for you.

Since the stack this routine uses is 256 bytes long, and 42 bytes are saved to it each time, you can safely call this routine 6 times before having to call PopStuff. The last set in is the first set out, just like pushing and pulling one byte on the processor's stack.

Example:

```
;this example looks for a filename on the current drive, and
;whether the file is found or not, it will continue with whatever
;it is designed to do. But while it is displaying a message to the
;user, it will save some stuff and restore afterwards and then
;continue on.
      ...
      jsr   ParamName   ;get the filename.
      jsr   CkPresdrive ;is it on the current drive?
      bit   goodflag    ;well?
      bmi   10$   ;branch if so.
      jsr   PushStuff   ;save everything.
      jsr   NotFound    ;tell the user 'not found'.
      jsr   PopStuff    ;restore everything.
 10$
      ...
      ...         ;and continue...
      ...
```

See Also: PopStuff

RdFnKeyDefs

Function: Load the default function key definitions from the current drive.

Pass: geoSHELL must be an open VLIR file.

Return: goodflag - set if geoSHELL was found on the current drive and the function key definitions loaded successfully.

Destroys: a, y, r1-r4, r7

Description: This routine will load in the function key definitions from the currently open drive if there is a copy of geoSHELL on the drive. The user will now have access to those function key definitions. The 'defkey' command uses this routine. You must first open geoSHELL as a VLIR file before calling this routine. Close the file afterwards.

Example:

```
;here is an example of a complete geoSHELL command that will
;load in the function key definitions from the drive that the
;user chooses. Let's call this command 'loadkeys'. The user is
;supposed to enter the command followed by a parameter that
;specifies the desired drive to load the keys from.
```

```
;Example:  loadkeys b
;This would load the function key definitions from drive B.
;You'll notice in this source code, that the idea is to keep
;checking for errors to keep the user from crashing his machine.
;Here's the complete source code for the command.

LoadKey:
      lda   ParamTable+1     ;check that only one
      jsr   CkTerminators    ;character is entered.
      bit   goodflag
      bmi   10$   ;branch if so.
 5$
      jmp   MissgFilename    ;(bad parameter)
 10$
      lda   ParamTable+0
      jsr   CkDrvLetter;convert this to the
                  ;desired device number (8-11).
      bit   goodflag   ;is it legal?
      bpl   5$    ;branch if not.
      pha         ;save the device number.
      LoadW a2,#ShellText    ;point a2 at the geoSHELL
                  ;name.
      pla
      jsr   CkThisDrive;see if geoSHELL is on this
      bit   goodflag   ;drive.
      bmi   20$   ;branch if so.   jsr   NotFound   ;tell the user, not
found.
      jmp   NoMoreCmds ;and exit.
 20$
      MoveW a2,r0 ;point r0 at the name.
      jsr   OpenWrFile ;this opens a VLIR file.
      bit   goodflag   ;was it successful?
      bmi   30$   ;branch if so.
 25$
      jmp   LoadProblem;tell about a loading problem.
 30$
      jsr   RdFnKeyDefs;load in the function keys.
      bit   goodflag   ;did it go OK?
      bmi   40$   ;branch if so.
      jsr   CloseRecordFile  ;close the file.
      bra   25$   ;and do an error thing.
 40$
      jsr   CloseRecordFile  ;close the file.
      LoadW r0,#successText  ;and tell the user
      lda   #(IN_TWO|TR_CR)  ;that it went OK.
      jsr   OtherMessage
      lda   #2
      jsr   IncCmdPointer    ;point at the next command.
      jmp   ExitCommand;and exit cleanly.

successText:
      .byte "Function Keys Loaded!",0
ShellText:
      .byte "geoSHELL",0
```

See Also:
ReadChannel

Function: Read a drive's error channel.

Pass: a - device number of desired drive.

InitForIO must be called first.

Return: EnOfShell - up to 256 bytes of information from the drive's error channel.

goodflag - set if the read was successful.

Destroys: a, x, y

Description: This command will read a drive's error channel and return the information to a buffer at EnOfShell. This buffer is 256 bytes long in order to accomodate the number of bytes that may come from a CMD device's error channel as opposed to a CBM device. This routine handles the communication between the computer and the drive for you. If goodflag is set, then you will find some valid bytes in the buffer at EnOfShell, otherwise it may just contain null bytes. So check goodflag first upon return from this routine. Also, before calling this routine, be sure to call PurgeAllTubos and InitForIO.

Example:

```
      ...
      lda   PRESDRIVE   ;do this in case it's an
      jsr   GetRealDriveNum  ;RD or RL as a RAM_1581.
                ;this also calls PurgeAllTurbos.
      jsr   InitForIO   ;pop out of GEOS.
      ...
      ...         ;through here, you might
      ...         ;have some code that
      ...         ;accesses the drive's DOS
      ...         ;for whatever reason.
      ...
      lda   REALDRIVE   ;now let's check the
      jsr   ReadChannel ;error channel.
      jsr   DoneWithIO  ;pop back into GEOS.
      bit   goodflag    ;was the read successful?
      bmi   20$   ;branch if so.
      jmp   DoError     ;display some sort of error.
 20$
      ...
      ...         ;now we can read the data
      ...         ;located at EnOfShell.
      ...
```

See Also: PurgeAllTurbos, SendCmd, SendListen, SendTalk, GetRealDriveNumReDisplayWindow

Function: Clear each line in the geoSHELL window and redisplay the text contained in it from the internal buffers.

Pass: nothing

Return: nothing

Destroys: no guarantees

Description: If your command does anything that might alter the state of what is being displayed in the geoSHELL window, you can use this routine to restore the text as it should be.

Example:

```
      ...
      ...         ;let's say that your command
      ...         ;puts up a couple of icons
      ...         ;in the middle of the geoSHELL
      ...         ;window in this code here.
      ...         ;that would wipe out the text
      ...         ;that was there.
      bit   goodflag   ;did the user do as he should?
      bmi   50$   ;branch if so.
      jsr   ReDisplayWindow  ;restore the text in the window.
      jmp   NoMoreCmds ;and exit.
 50$
      ...
      ...         ;continue on...
      ...
```

See Also: ResetScreen, ClearRemainder, ClearWindow, EraseWindow, ReDoWindow, DispLetter, DriveLetter, DsplyLine, DsplyString, CarrReturn, set_XY_pos, save_XY_pos, ClearBoth, ClearLine, ClearScreen, DispText, unsetPrompt
ReDoWindow

Function: Redraw the geoSHELL window and any text that was already contained in it.

Pass: nothing

Return: nothing

Destroys: no guarantees

Description: When your command performs some sort of action that might disrupt the geoSHELL window, call this routine to restore it. The text that is contained in the memory buffer will be restored to the window after the window is redrawn.

Example:

```
      ...
      jsr   ClearScreen
      ...
      ...         ;do something on the screen.
      ...
      jsr   ReDoWindow ;redraw the window as it was.
      jmp   ExitCommand;and exit cleanly.
```

See Also: ResetScreen, ClearRemainder, ClearWindow, EraseWindow, ReDisplayWindow, DispLetter, DriveLetter, DsplyLine, DsplyString, CarrReturn, set_XY_pos, save_XY_pos, ClearBoth, ClearLine, ClearScreen, DispText, unsetPrompt

ResetFromPath

Function: Reset a CMD device back from the path partition to the current partition the user is operating in.

Pass: nothing

Return: nothing

Destroys: no guarantees

Description: After calling any routine that does a file search on a path partition, call this routine to put the drive back to it's proper partition. If you do not, then the drive will remain in the partition that is defined as the path partition if the desired file was found on the path partition. It is up to your command to do this. This way, the path partition can remain open until you are finished with it.

Example:

```
      ...
      jsr   ParamName  ;get the filename.
      bit   goodflag   ;is it valid?
      bmi   10$   ;branch if so.
      jmp   MissgFilename    ;do an error message and exit.
 10$
      jsr   CkPath     ;check the path partition
      bit   goodflag   ;for the file.
      bmi   20$   ;branch if found.
      jmp   FileNotAvail     ;otherwise do a file not
                  ;available error and exit.
 20$
      ...
      ...         ;do what your command intends
      ...         ;at this point.
      ...
      jsr   ResetFromPath    ;restore the original partition.
      inc   NUM_IN_NAME;ParamName sets this variable.
      lda   NUM_IN_NAME
      jsr   IncCmdPointer    ;point to the next command.
      jmp   ExitCommand ;and exit cleanly.
```

See Also: CkPresdrive, CkThisDrive, CkOtherDrives, CkPathOrder, CkPath, CkForDisk, FindParName

ResetScreen

Function: Display a fresh geoSHELL screen in the correct mode.

Pass: nothing

Return: nothing

Destroys: a, x, y, r0-r15

Description: This will completely clear the screen and redraw it fresh in the current mode (40 or 80 columns). The geoSHELL window will be redrawn and the cursor will be positioned to the left side of line one.

Example:

```
      ...
      jsr   ResetScreen ;wipe the screen and start over.
      ...
```

See Also: ResetScreen, ClearRemainder, ClearWindow, EraseWindow, ReDisplayWindow, DispLetter, DriveLetter, DsplyLine, DsplyString, CarrReturn, set_XY_pos, save_XY_pos, ClearBoth, ClearLine, ClearScreen, DispText, unsetPrompt

R_Icons

Function: Restore the default geoSHELL icon.

Pass: nothing

Return: otherPressVector - set for mouse positioning of cursor.

Close icon - set to exit to Desktop or the calling interface that geoSHELL was entered from.

Mouse limits - set to default settings.

Destoys: a, x, y, r0-r11

Description: If your command does anything to change the icon status that the GEOS kernal is controlling, or otherPressVector, or the mouse position limits, call this routine to fix the problem. This will inform GEOS that the icon at the upper right of the geoSHELL window is there. Some actions might kill this icon, thereby making it necessary to call this routine before exiting your command. While testing your command, be sure to check the icon by clicking on it to see if it is still active after exiting your command. If not, then add the call to this routine before exiting. If you exit to NoMoreCmds, this will be done for you. It actually does no harm to call this routine before exiting your command even if your command does nothing to change these things.

Example:

```
      ...
      jsr   R_Icons    ;fix the close icon.
      jmp   ExitCommand;all done with command.
```

See Also:

SaveToREU

Function: Copies geoSHELL and Input driver info to REU.

Pass: nothing

Return: nothing

Destroys: a, x, y, r0-r3

Description: When geoSHELL is activated, it installs itself as the default user interface into the GEOS kernal in place of the Desktop. If the user were to exit to BASIC and then use RBOOT to get back into geoSHELL, the Desktop would then be the default interface. This routine fixes that problem. The same thing would happen if the user were to change input drivers. So, for that reason, the 'install' command calls this routine also. Only the portion of the kernal that contains the needed info for the user interface and the input driver is copied to the GEOS kernal area of the REU. There probably is no reason to use this routine unless your command should affect the input driver in some way, such as a patch would. Then you would want to call this routine to insure the patch would be used in an RBOOT situation.

Should there be no REU, no harm is done since the routine will simply rts if no REU is present.

Example:

See Also:
save_XY_pos

Function: Save the current values of r1H and r11 into two variables.

Pass: r1H - the current desired pixel location of the cursor.

r11 - the desired horizontal pixel location of the cursor.

Return: cur_X_pos will hold the value that was in r11.

cur_Y_pos will hold the value that was in r1H.

Destroys:	a

Description:	This routine allows you to bypass geoSHELL's management routines and physically relocate the cursor on the screen, if only temporary. Once your command exits, geoSHELL might again take control and properly place it.

There is really no need to use this routine since most of your needs can be handled through other routines that let geoSHELL handle the text displaying and cursor positioning.

Example:

```
;this example serves no real purpose other than to show
;how this routine could be used. First r1H and r11 are loaded
;with values that correspond to the upper area of the screen.
;This location is 10 pixels down from the top and 48 pixels
;over from the left side. Then save_XY_pos is used to save
;these values to their proper locations and OtherMessage will
;then print a message at the top of the screen instead of
;in the geoSHELL window.

      ...
      LoadB r1H,#10
      LoadW r11,#48
      jsr   save_XY_pos
      LoadW r0,#upperText
      lda   #0
      jsr   OtherMessage
      ...
```

See Also:	set_XY_pos

Searching

Function:	Display the 'Searching For Filename' message.

Pass: nosearchmessage - cleared to zero. (it is normally zero and also is always cleared by this routine)

a2 - null terminated filename to display in the message.

Return:	message displayed on screen.

Destroys:	no guarantees

Description:	This routine merely needs a2 to be pointing at a null-terminated filename, and that filename will be included in a message that will be displayed to the screen, such as, 'Searching For Filename'. This is basically just another internal geoSHELL message that can also be called with the routine, Message. 'Searching For' is message number 2. Usually, the entire line the cursor is currently on will be cleared first before displaying this message. The message will always start on the third character location and perform a carriage return afterwards.

Whenever the routine CkAllDrives is being used to search for a file, this routine is automatically called. So, if you wish the 'Searching For ...' message to be displayed you need not do anything when CkAllDrives is being used. For any other searching function, such as CkThisDrive, you would have to call this routine. If you wish to have CkAllDrives 'not' call this routine, then set the variable nosearchmessage to 128. It is not necessary to reset this variable afterwards since it is automatically reset.

Example:

```
      ...
      jsr   ParamName   ;get the filename.
      bit   goodflag    ;is there one?
      bmi   10$   ;branch if so.
      jmp   MissgFilename    ;display error and exit.
 10$
      jsr   Searching   ;display searching message.
      jsr   CkPresdrive ;look on the current drive.
      bit   goodflag    ;did we find the file?
      bmi   20$   ;branch if so.
      jmp   FileNotAvail     ;display an error and exit.
 20$
      ...         ;continue on...
```

See Also:SendCmd

Function: Send a string of bytes to the drive's command channel.

Pass: a - real device number of desired drive.

r0 - points to the bytes to send.

cmdlength - number of bytes (1-255) to send. A zero here will send 256 bytes.

Be sure to purge the turbo code from the drive that the command is being sent to, and call InitForIO also.

Return: goodflag - set if the drive responded and accepted the bytes.

r0 - unchanged.

Destroys: a, x, y

Description: This routine may be used to send any string of bytes to a drive that is connected to the computer. In fact, it does not have to be one of the drives that GEOS is working with. Just point r0 at the string of bytes, and set cmdlength to the number of bytes you wish to send, load the accumulator with the (real) device number of the desired drive and call this routine.

Don't forget to purge the turbo code in the drive and call InitForIO before calling this routine.

Example:

```
;this example will send a string to the drive's command channel
;that will cause a drive to initialize the disk.

      ...
      jsr   PurgeAllTurbos   ;let's kill all the turbos.
      jsr   InitForIO  ;pop out of GEOS.
      LoadW r0,#iText  ;point r0 to bytes.
      LoadB cmdlength,#2     ;load cmdlength with # of bytes.
      lda   driveNum   ;get the device number.
      jsr   SendCmd    ;send the command.
      bit   goodflag   ;did the drive take it?
      bmi   20$   ;branch if so.
      jsr   DoneWithIO ;pop back into GEOS.
      jmp   DoError    ;do an error of some sort.
 20$
      ...
      ...        ;continue on...
      ...


iText:
      .byte "I0"  ;initialize command.driveNum:
      .byte 8     ;drive number stored here.
```

See Also: PurgeAllTurbos, ReadChannel, SendListen, SendTalk, GetRealDriveNum

SendListen

Function: Get a drive's attention and send it a command.

Pass: a - real device number of desired drive.

cmdtosend - the secondary command byte to send to the drive.

You must purge the turbo code in the drive and call InitForIO before calling this routine.

Return: goodflag - set if device is responding.

the device - still in the 'Listen' mode (if goodflag is set) and will accept more bytes on the serial bus.

Destroys: a

Description: This routine will prepare a drive to receive bytes on the serial bus. After setting up and calling this routine, if goodflag is set, then you may continue to send bytes on the serial bus using the kernal routine, Ciout, and the drive will continue receiving them. Just load cmdtosend with a byte to send as a secondary command and load the accumulator with the device number of the desired drive and call this routine. Then just keep sending bytes through Ciout. When finished, just call the kernal routine Unlsn, and it will tell the drive we are finished sending bytes.

Example:

```
;this particular example will open a sequential file on the current
;drive so that it's contents may be read into the computer.

      ...
      jsr   ParamName  ;get a filename.
      bit   goodflag   ;is it valid?
      bmi   10$   ;branch if so.
      jmp   MissgFilename    ;display bad parameter.
 10$
      lda   PRESDRIVE
      jsr   GetRealDriveNum  ;get the real device number.
                  ;this will also purge all turbos.
      bit   goodflag   ;is this a real drive?
      bmi   20$   ;branch if so.
      LoadW r0,#notRealText  ;display an error message,
      lda   #(IN_TWO|TR_CR)
      jsr   OtherMessage
      jmp   NoMoreCmds ;and exit.
 20$
      jsr   InitForIO  ;pop out of GEOS.
      LoadB cmdtosend,#(3|$f0)     ;open channel 3,
      lda   REALDRIVE  ;in this particular device,
      jsr   SendListen ;attempt to open it.   bit   goodflag   ;was it
successful?
      bmi   30$   ;branch if so.
 25$
      jsr   DoneWithIO ;otherwise,
      LoadW r0,#notRespText  ;do an error message.
      lda   #(IN_TWO|TR_CR)
      jsr   OtherMessage
      jmp   NoMoreCmds ;and exit.
 30$
      ldy   #0
 40$
      lda   (a2),y     ;open the channel with
      jsr   Ciout ;our filename.
      iny
      cpy   NUM_IN_NAME ;have we sent the whole
                  ;filename?
      bne   40$   ;branch if not.
      lda   #CR   ;send a carriage return
      jsr   Ciout ;to the drive.
      jsr   Unlsn ;make it do it's thing.
      lda   REALDRIVE
      jsr   ReadChannel ;see if it did it's thing.
      bit   goodflag   ;check goodflag first.
      bpl   25$
      lda   EnOfShell+0;did the error channel
      cmp   #'0'  ;return '0' in the first byte?
      bne   25$   ;branch if not.
      cmp   EnOfShell+1;how about the second byte?
      bne   25$   ;branch if not.
      LoadB cmdtosend,#(3|$60)     ;get the channel ready
      lda   REALDRIVE  ;for reading.
      jsr   SendTalk
      bit   goodflag   ;is it ready?
      bpl   25$   ;branch if not.
      LoadW r0,#buffer ;point r0 at our buffer.
      LoadB STATUS,#0  ;be sure to clear STATUS.
      ldy   #0
 50$
      jsr   Acptr ;read a byte from the drive.
      sta   (r0),y     ;store it in our buffer.
      iny         ;increment for the next byte.
      bne   60$
      inc   r0H
      lda   r0H   ;have we reached the end
      cmp   #$7f  ;of our buffer?
      beq   70$   ;branch if so.
 60$
      lda   STATUS     ;is this the end of file
            ;or any other error?
      beq   50$   ;branch if not.
 70$
      jsr   Untalk     ;stop the drive.
      LoadB cmdtosend,#(3|$e0)     ;and close the channel.
      lda   REALDRIVE
      jsr   SendListen
      jsr   Unlsn jsr   DoneWithIO ;pop back into GEOS.
      ...
      ...         ;at this point, we have
      ...         ;a buffer filled with the
      ...         ;bytes from the file that
      ...         ;the user desired and our
      ...         ;command may continue to do
      ...         ;as it is designed to do.
      ...


notRealText:
      .byte "This Is Not A REAL Drive!",0
notRespText:
      .byte "The Drive Is Not Responding!",0
```

See Also: PurgeAllTurbos, ReadChannel, SendCmd, SendTalk, GetRealDriveNum

SendTalk

Function: Get a drive's attention and send it a command to prepare it for talking.

Pass: a - real device number of desired drive.

cmdtosend - the secondary command byte to send to the drive.

You must purge the turbo code in the drive and call InitForIO before calling this routine.

Return: goodflag - set if device is responding.

the device - is in the 'Talk' mode (if goodflag is set) and will send bytes to the computer by using the kernal routine 'Actpr'.

Destroys: a

Description: This routine will prepare a drive to send bytes on the serial bus. After setting up and calling this routine, if goodflag is set, then you may begin receiving bytes on the serial bus using the kernal routine Acptr. Just load cmdtosend with a byte to send as a secondary command and load the accumulator with the device number of the desired drive and call this routine. Then just keep receiving bytes through Acptr until STATUS is any non-zero value. When finished, use the kernal routine Untalk to tell the drive we are finished receiving bytes.

Example: Refer to the example in the description for the routine 'SendListen'. It includes an example of how to use SendTalk.

See Also: PurgeAllTurbos, ReadChannel, SendCmd, SendTalk, GetRealDriveNum

Send_GC_Byte

Function: Send a byte to a printer via a geoCable connected to the user port.

Pass: a - byte to send.

InitForIO must be called before accessing this routine.

Return: nothing - no error checking is performed

Destroys: a

Description: This routine will send the byte contained in the accumulator to a parallel printer connected to the user port via a geoCable. No error checking is done. If the routine returns, the byte was sent, and you can send another one if desired. It is a good idea to call Open_GC_Channel before this one to verify the presence of a printer before using this routine. Otherwise, if the printer is not accepting the byte, the routine might just wait until it does so. That could be forever. As long as Open_GC_Channel returns with x=0 then it is OK to use this routine to send data to the printer.

Example:

```
;this example will send the bytes contained in a buffer to the
;printer. r0 points to the start of the buffer and r2 points to
;the byte just past the end of the buffer. If this example returns
;with goodflag set, the buffer was sent to the printer.

SendBuffer:
      jsr   InitForIO   ;pop out of GEOS.
      jsr   Open_GC_Channel  ;check for a printer.
      txa         ;check if x = 0.
      beq   10$   ;branch if so.
      jsr   DoneWithIO ;pop back into GEOS.
      jmp   nogood     ;signal calling routine of error.
 10$
      ldy   #0    ;y will stay at zero throughout
                  ;the sending of text.
 20$
      lda   (r0),y     ;get a byte from the buffer.
      jsr   Send_GC_Byte     ;send it to the printer.
      inc   r0L   ;point at the next byte.
      bne   30$
      inc   r0H
 30$
      sec         ;have we reached the end?
      lda   r0L
      sbc   r2L
      lda   r0H
      sbc   r2H
      bcc   20$   ;branch if not.  jsr   DoneWithIO ;pop back into GEOS.
      jmp   yesgood    ;and tell the calling routine
                  ;everything went OK.
```

See Also: Open_GC_Channel

SetBrdrColor

Function: Change the color of the 80 column screen's border.

Pass: brdrcolor - color (0-15).

Return: border color changed.

Destroys: a, x, temp_check

Description: This routine will change the color of the 80 column screen's border to the color that is stored in brdrcolor. Your routine must check to be sure that the machine is a 128 before calling this routine. It should also check the videomode to make sure that the computer is not in monochrome mode.

Example:

```
;this example is a routine that will change the border to blue.

BlueBorder:
      bit   videomode   ;128 in RGB mode?
      bmi   10$   ;branch if so.
      rts
 10$
      LoadB brdrcolor,#DK80BLUE
      jmp   SetBrdrColor     ;change the border
                  ;to blue.
```

See also: ColorScreen, FixColors

SetDrName

Function: Display drive letter, drive type, and disk name.

Pass: a - desired drive number (8-11) representing drive A-D.

Return: goodflag - meaningless with this routine.

Destroys: no guarantees

Description: Load the accumulator with a device number (8-11) representing any of the available drives or ramdisks and call this routine to display the drive letter, the drive type, and the name of the disk that is in the drive. Both the 'status' and the 'dir' commands call this routine. The status command will call it once for each of the drives that are being used. The line the cursor is on will be cleared before the text is displayed.

Example:

```
;suppose you wanted to create a command that would display various
;bits of information about a drive and the disk that it contains.
;You could call SetDrName to start with and then test the drive
;and disk to get other bits of info to display.

      ...
      lda   ParamTable+1     ;make sure only one
      jsr   CkTerminators    ;parameter was used for this.
      bmi   10$   ;branch if so.
 5$
      jmp   MissgFilename    ;display bad parameter.
 10$
      lda   ParamTable+0     ;get the parameter.
      jsr   CkDrvLetter ;convert it to 8-11.
      bit   goodflag   ;did it convert OK?
      bpl   5$    ;branch if not.
      jsr   SetDrName  ;display some stuff.
      ...
      ...         ;now we can continue
      ...        ;testing the drive...
      ...
```

See Also: Dir, MissDisk, Status, CkDrvLetter, CkForDisk

SetPartParams

Function: Set up some internal variables for the current partition.

Pass: curdrvtype - Set this variable by calling GetRealDriveNum first.

a - real device number from REALDRIVE after calling GetRealDriveNum.

Return: goodflag - set if a CMD device with partitions.

certain internal variables will contain information about the current partition on the desired device.

Destroys: a, x, y, r0

Description: The main purpose of this routine is to establish the information that geoSHELL needs for the current partition of a particular drive. If you were to open a different partition such as what happens when geoSHELL accesses commands on the path partition, then with this information, the original partition and drive is known so that it may be restored. Until the partition is restored, the routine should not be used again. Once the partition is restored, either through the required call to ResetFromPath, OpenCurPartition, or InitCurPartition, then SetPartParams may be used again in order to set up for another partition to be opened. Even when your command intends to open a partition through a call to OpenPartition, and leave that partition open with no intentions of returning to the original one, this routine still needs to be called. This is because OpenPartition checks to make sure that the desired partition to open is of the same type as the current one so that it is compatible with the disk driver that is being used with GEOS on that drive.

Example: Refer to OpenPartition for an example of how to use this routine.

See Also: InitCurPartition, OpenCurPartition, ResetFromPath, OpenPathPartition, SetPartParams

SetThisDevice

Function: A substitute for the GEOS routine 'SetDevice'.

Pass: a - drive number 8-11.

Return: goodflag - set if device is now ready.

x - contains zero or error number if any from GEOS.

Destroys: a, x, y

Description: This routine is a recommended substitute for the GEOS kernal routine 'SetDevice'. During it's execution, it still calls SetDevice, but also does some other checking. It also does a jump through yesgood or nogood for setting goodflag. During geoSHELL development, it was discovered that some disk drivers that are used in GEOS would mysteriously lock up under certain circumstances. This lock up occured while calling SetDevice. The lockup required just the right conditions for it to occur. By calling this routine instead of SetDevice, you will still accomplish the same function but with less chance of any problems occuring under these conditions.

Example:

```
;this example expects a drive letter from the user following the
;command. After the usual checking of the parameter, SetThisDevice
;is called and the command proceeds to execute.
      ...
      lda   ParamTable+1     ;only one parameter needed.
      jsr   CkTerminators
      bit   goodflag
      bmi   10$   ;branch if one parameter.
 5$
      jmp   MissgFilename    ;display bad parameter.
 10$
      lda   ParamTable+0     ;get the parameter.
      jsr   CkDrvLetter;set it to 8-11.
      bmi   goodflag   ;is it 8-11?
      bpl   5$   ;branch if not.
      jsr   SetThisDevice    ;Set the Device.
      bit   goodflag   ;everything OK?
      bmi   20$   ;branch if so.
      LoadW r0,#noDriveText  ;do an error message.
      lda   #(IN_TWO|TR_CR)
      jsr   OtherMessage
      jmp   NoMoreCmds ;and exit.
 20$
      ...
      ...        ;continue on...
      ...

noDriveText:     .byte "Drive Not Available!",0
```

See Also: CkForDisk

set_XY_pos

Function: Load r1H and r11 with the pixel values of the current cursor position.

Pass: nothing

Return: r1H holds the vertical pixel location of the cursor.

r11 holds the horizontal pixel location of the cursor.

Destroys: a

Description: Used by internal geoSHELL routines or your routines if you need to do any manual displaying of text to the screen. Text displaying is normally handled nicely by geoSHELL with other routines, so you rarely would have a need to call this routine.

Example: See save_XY_pos for an example of how to use this routine.

See Also: save_XY_pos

Status

Function: Display information for each of the connected drives.

Pass: nothing

Return: information on the screen.

goodflag is meaningless here.

Destroys: no guarantees.

Description: This is the routine that the 'status' command calls. In fact, this is the status command. You can use this if you need to show the drives that are currently online. The swap command calls this to display the arrangement of the drives after it performs the swap.

Example:

```
      ...
      jsr   Status
      ...
```

See Also: Dir, MissDisk, SetDrName, CkDrvLetter, CkForDisk

StripRulers

Function: Strip the GeoWrite rulers, escapes and any other unwanted characters from a GeoWrite page loaded into memory.

Pass: r0 - start of null terminated buffer.

r2 - location to save converted buffer to.

Return: buffer stripped, null-terminated, and located where r2 was pointing when the routine was called.

goodflag - affected, but meaningless with this routine.

r2 - points at the zero byte that ends the buffer.

Destroys: a, x, y, r0, r2

Description: This routine will take a GeoWrite page that is loaded into memory and convert in such a way that you will be left with only ASCII text. Load r0 with start of the buffer and r2 with the start of the area to save the converted buffer to. r2 must be less than or equal to r0, or it can point past the end of the buffer that r0 is pointing at. Normally, you would set r2 equal to r0 so that the buffer will remain in the same location. This will strip the GeoWrite rulers, escapes and any other unwanted characters.

The only hitch to this routine is that it was designed for the startup and exec commands. This means that carriage returns and tabs are converted to a single space. In fact, any byte that is less than a 32 or greater than a 126 is converted to a space. For an exec file, this ends up making it basically one long command line. Remember that geoSHELL allows more than one command to be on a line, so it has no use for any carriage returns.

After this routine is called, the only byte in the buffer that will fall outside of 32-126 will be the last one and it will be a zero byte.

Example:

```
      ...
      lda   #[buffer
      sta   r0L
      sta   r2L
      lda   #]buffer
      sta   r0H
      sta   r2H
      jsr   StripRulers
      ...
```

See Also: AdjTxtPointer, Message, OtherMessage, DsplyLine, DsplyString, CarrReturn, ClearLine, ClearBoth, ClearRemainder, CkESC_RULER, set_XY_pos, save_XY_pos, MskCtrlCodes, ConvToAscii, DispText

Switch

Function: Switch 40/80 column modes on a 128.

Pass: nothing

Return: screenmode - reflects the switched mode.

if 64 mode - does not return.

Destroys: no guarantees

Description: This is the routine that the 'switch' command jumps to. In fact, this is the switch command, and will act just as if the user entered the command. If this routine is accessed in 64 mode, the user will be presented with a message stating '128 Only Command' and will promptly jump through NoMoreCmds, thereby ending your command. So, you would want to check screenmode to make sure that the machine is a 128 before using this routine, unless you intend for this to happen.

Example:

```
;this example is actually the source code for a possible command
;that could be called 'srun'. This will check the screenmode of a file
;before it is loaded, and it will give the user the option of
;switching screens or aborting. The routine CkModeFlag is called in
;this command to check if the file is compatible with the machine.
;The various values that can be returned in x by this routine can
;be found in the appendix.

SRun:
      jsr   ParamName   ;get the filename.
      bit   goodflag    ;was there one?
      bmi   20$   ;branch if so.
      jmp   MissgFilename    ;display bad parameter.
 10$
      jsr   CkAllDrives ;look for the file.
      bit   goodflag    ;did we find it?
      bmi   20$   ;branch if so.
      jmp   FileNotAvail     ;display not available.
 20$
      jsr   GetHeader   ;get the file's header block.
      bit   goodflag    ;is it a GEOS file?
      bpl   80$   ;branch if not and then try
                  ;to run it.
      jsr   CkModeFlag  ;see if it's compatible?
      bit   goodflag    ;well?
      bmi   80$   ;branch if so and run it.
      bit   screenmode  ;check the mode we're in.
      bvs   90$   ;branch if 64 mode (can't run).

;at this point we know that we are running on a 128. All we need
;to check for is if the file is for 64 only. The only other
;possibility would be if we need to switch from 40 to 80 or;from 80 to
40. So, if not 64 mode, then we can switch screens.
;if x = 36 " is for 40 columns!"
;if x = 37 " is for 128-40 columns!"
;if x = 38 " is for 128-80 columns!"
;if x = 39 " is for 128 mode!"
;if x = 40 " is for 64 mode!"
;if x = 41 " is not a GEOS file!"

      cpx   #40   ;is the file for 64 only?
      beq   90$   ;branch if so (can't run).
      txa         ;save x for now.
      pha
      MoveW a2,r0 ;display the filename...
      lda   #(IN_TWO|TR_ONE) ;stay on the same line...
      jsr   OtherMessage
      pla
      txa         ;restore x and...
      lda   #TR_CR
      jsr   Message     ;display the mode the file
                  ;is designed for.
      LoadW r0,#switchText   ;display a requestor
      lda   #(IN_TWO|TR_CR)  ;to switch screens.
      jsr   OtherMessage
      jsr   Wait  ;wait for user input.
      lda   st_KeyData ;get the key that was pressed.
      cmp   #25   ;was it a 'y'.
                  ;(keycode, not ascii code)
      beq   50$   ;branch if so.
      jmp   NoMoreCmds ;any other key will abort.
 50$
      jsr   Switch     ;switch screens.
 80$
      jmp   DoRun ;hand the job over to
                  ;geoSHELL to run the file.
 90$
      jmp   CantRun    ;tell the user, no go.
                  ;(this would be your own
                  ;error routine)


switchText:
      .byte "Would You Like To Switch Screens?  y/n",0
```

See Also: CkModeFlag, DoRun

unsetPrompt

Function: Turn the blinking cursor off.

Pass: nothing

Return: cursor is turned off.

Destroys: nothing

Description: If you wish to turn the cursor on, it is a simple matter to do so. Just set bit 7 of cursorOn. But to turn it off requires more work. This routine will do it for you. It has to make sure that the cursor isn't currently in the visible stage with the character under it in reverse video. If so, it will erase the cursor and reverse the character and then it will make sure that it stays off by clearing bit 7 of cursorOn. Of course, you do not need to access this routine unless you have already turned the cursor on for user input of some sort and then wish it turned back off.

If your routine does an rts to have GEOS mainloop handle your input stuff, then you might want to turn the cursor off if you are not working with any text input. Otherwise, you might have a blinking cursor right in the middle of the graphic that you just displayed on the screen while waiting for the user to click on an icon. This would not look good.

Example:

See Also:

Wait

Function: Wait for the user to press a key.

Pass: nothing

Return: st_keyData contains the keycode (not ascii value) of the key that was pressed.

goodflag - meaningless, it is always set by this routine.

Destroys: a, x, y, r3L

Description: This routine is similar to YesKeypress, except that it turns the mouse off while it is waiting for the user to press a key. This prevents the user from being able to click on geoSHELL's close icon, instead of pressing a key.

Example:

```
      ...
      jsr   Wait  ;wait for a keypress with the
                  ;mouse turned off.
      ...
      ...         ;continue on...
      ...
```

See Also: NoKeypress, ClrKeyData, CkKeyboard, YesKeypress

yesgood

Function: Usually identifies a successful routine.

Pass: nothing

Return: goodflag - bit 7 set.

Destroys: nothing

Description: geoSHELL uses this routine extensively. It and nogood are the universal routines for identifying successes and failures. This routine actually does nothing more than set a flag (goodflag) to give an indication to a calling routine that a success occured. All the calling routine has to do is check goodflag with the bit instruction. It will be minus if yesgood set it. As an example, the routine 'CkForDisk' will jump through yesgood upon exit of the routine if it found a formatted disk in the drive.

It's true that the carry flag or some other form of error checking could have been used, but it's also possible that the x register or the carry flag could change before we are ready to check it. We have more control over how goodflag is affected, and can do some processing before checking it if needed.

Example:

```
;this particular example will test an 80 character buffer for any
;non-space character. If a non-space character is found, then it
;will return with goodflag set by calling yesgood.

CkNonSpace:
     ldx   #0
 10$
     lda   charBuffer,x     ;get a character from the buffer.
     beq   20$   ;branch if null encountered.
     cmp   #' '  ;is it a space?
     bne   40$   ;branch if not.
     inx         ;point at the next byte.
     cpx   #80   ;have we reached the maximum?
     bne   10$   ;branch if not.
 20$
     jmp   nogood     ;signal no non-spaces found.
 40$
     jmp   yesgood    ;signal at least one
                 ;non-space found.
```

See Also: nogood

YesKeypress

Function: Wait for the user to press a key.

Pass: nothing

Return: st_keyData contains the keycode (not ascii value) of the key being pressed.

goodflag - always cleared by this routine.

Destroys: a, x, y

Description: Call this routine when you need a keypress from the user. The Commodore keycode value as listed in the 64 or 128 Programmer's Reference Manual will be returned in st_KeyData. This routine will hold the computer indefinitely until a key is pressed. The value in goodflag is meaningless upon return, however be cautious that you pass the correct value to the next routine in your command because goodflag will always be reset to zero by this routine.

Example:

```
;this example will check for the user to press the spacebar.
;Of course, any other key could also be checked for, or
;all of them.

CkForSpace:
      jsr   YesKeypress ;wait for the user to
                  ;press a key.
      lda   st_keyData ;get the keycode of the key
                  ;the user is pressing.
      cmp   #60   ;is it the spacebar?
      beq   10$   ;branch if so.
      jmp   nogood      ;inform the calling routine
                  ;that the spacebar was not
                  ;pressed.
 10$
      jmp   yesgood     ;inform the calling routine
                  ;that the spacebar was
                  ;pressed.
```

See Also: NoKeypress, ClrKeyData, CkKeyboard, Wait

-- NOTES --

IncCmdPointer

Function: Position the command pointer past your command's parameter.

Pass: accumulator - number of bytes in your parameter plus one for a trailing space which is supposed to separate your command with the next.

Return: a4 - incremented the number of desired bytes.

Destroys: a, y

Description: Since geoSHELL allows more than one command to be typed in on a single line and also since your command could contain a parameter that might not be any specific length, geoSHELL requires that your command manage the command pointer (a4) at some point in time before you exit your command. You must load the accumulator with the number of bytes that you need the pointer advanced and then call this routine. This number should be the number of characters that were in your parameter plus one for the space that the user should have typed in just before the next command. If your command does not use any parameter, then you do not need to call this routine, because geoSHELL's command pointer will already be pointing to the location just past your command's name in the command buffer.

Example:

```
      ...
      ...        ;the bulk of your command.
      ...
      lda   #2    ;assuming your command only
                 ;used one parameter plus a
                 ;space after the parameter.
      jsr   IncCmdPointer    ;point geoSHELL in the
                 ;proper spot.
      jmp   ExitCommand ;and exit your command.
```

Example #2

```
      ...
      ...        ;part of your command.
      ...
      jsr   ParamName  ;point a2 to a filename
                 ;parameter,
                 ;and set NUM_IN_NAME to the
                 ;length of the filename.
      bit   goodflag
      bmi   10$
      ...   ...   ;display an error here.
      jmp   NoMoreCmds ;and then stop any further
                 ;command processing. 10$
      ...
      ...        ;the bulk of your command.
      ...
      inc   NUM_IN_NAME ;add one for a terminator
                 ;following the filename.
      lda   NUM_IN_NAME ;and get the value.
      jsr   IncCmdPointer    ;point geoSHELL in the
                 ;proper spot.
      jmp   ExitCommand ;and exit your command.
```

See Also: ExitCommand

InitCurPartition

Function: Open the currently active partition after changing to a different one.

Pass: Certain internal variables must be set by calling GetRealDriveNum and SetPartParams beforehand.

Return: The partition the user is seeing will be reopened.

Destroys: a, x, y, r0-r15

Description: This routine requires that a certain amount of setup is performed to work properly. Normally you would want to call other routines such as ResetFromPath instead of this one.

In order to make this routine work properly, geoSHELL needs to figure out what the current partition is first as well as the correct device number. You do that by calling other geoSHELL routines to properly set up the internal routines.

Example:

```
;this could be part of the code that you might use for a command
;that would switch among partitions.

     ...
     jsr  SetPartStuff     ;set up the variables for
               ;the current partition
     bit  goodflag   ;does this device use
               ;partitions?
     bmi  10$   ;branch if so.
     jmp  NotPartDevice    ;otherwise display an error.
 10$
     ...
     ...       ;do whatever you had planned.
     ...
     jsr  InitCurPartition ;open the current partition.
     ...
     ...       ;continue with more code
     ...
     rts

SetPartStuff:
     lda  PRESDRIVE
     jsr  GetRealDriveNum  ;is this a DOS type device?
     bit  goodflag
     bpl  10$   ;branch if not.
     jsr  InitForIO
     lda  REALDRIVE  ;this was set by GetRealDriveNum
     jsr  SetPartParams    ;set some internal variables.
               ;this will set goodflag
               ;if the device has partitions.
     jsr  DoneWithIO 10$
     rts
```

See Also: OpenPartition, OpenCurPartition, ResetFromPath, OpenPathPartition, SetPartParams
InitgeoSHELL

Function: Start geoSHELL (Initial jump address).

Pass: nothing

Return: not applicable

Destroys: not applicable

Description: This routine will 'start' geoSHELL. This routine is the main starting point of geoSHELL. When either GEOS or getshell loads geoSHELL into memory, this address is jumped to. It is not recommended to use this at any other time since the state of GEOS must be such as it is when an application is first started due to the way geoSHELL operates. There is actually no reason to ever jump to this routine. In some cases, if you were to try to restart geoSHELL with this routine, then some functions may not work quite right. NoMoreCmds may not work quite right afterwards. When it is necessary to reset the state of geoSHELL, use the routine NoMoreCmds instead of InitgeoSHELL.

Example: no example since this routine is not recommended.

See Also: NoMoreCmds

LoadProblem

Function: Display an error message and return control to the user.

Pass: nothing

Return: not applicable

Destroys: not applicable

Description: If your command works in such a way that it attempts to load in a file and has some sort of problem with loading, you can use this routine to exit your command. This will display to the user, 'Problem Loading File!', and then it will jump to NoMoreCmds. Then the user can check to see why a problem occured, whether it be a bad disk or a missing or corrupted file or whatever and try your command again.

Example:

```
     ...
     jsr   ParamName  ;fetch the filename the
     bit   goodflag   ;user typed in.
     bmi   10$   ;branch if successful.
     jmp   MissgFilename    ;display 'Bad Parameter'.
 10$
     jsr   CkPresdrive;find the file on the
     bit   goodflag   ;current drive.
     bmi   20$   ;branch if found.
     jmp   FileNotAvail     ;tell the user it's not there.
 20$
     MoveW a2,r0 ;point r0 to the now true
                 ;filename without wildcards.
     jsr   OpenRecordFile   ;let GEOS open it.
     txa         ;well?
     beq   30$   ;branch if OK.
     jsr   CloseRecordFile  ;make sure it's closed.
     jmp   LoadProblem ;and exit with an error.
 30$
     ...
     ...         ;desired VLIR file is now
     ...         ;open, so we can
     ...         ;continue...
     ...
```

See Also: MissgFilename, NoMoreCmds, Message, OtherMessage, FileNotAvail, MissDisk, NotFound

Message

Function: Display an internal geoSHELL message.

Pass: x - number of desired message.

a - formatting information.

Return: nothing

Destroys: no guarantees

Description: This routine will display one of geoSHELL's own internal messages. This allows you to make use of whatever is available without the need to include your own messages. Just load x with the number of the desired message, load the accumulator with the formatting information and call this routine. A complete list of available messages and a description of the formatting information is listed in the appendix.

Example:

```
;this particular example is for a command that might only work
;from a startup or exec file. If coming from anything but, then
;message #72, 'This Is A Startup Command!', will be displayed.
;This example could be the first thing your command does.


      bit   STUPMODE   ;are we in startup mode?
      bmi   10$   ;branch if so.
      ldx   #72   ;point at message 72.
      lda   #(IN_TWO|TR_CR)  ;indent two and carriage rtn.
      jsr   Message    ;do the message.
      jsr   ClrTrnsName ;remove the command.
      jmp   NoMoreCmds ;and return to the user.
```

```
 10$
      ...
      ...        ;continue on...
      ...
```

See Also: MissgFilename, NoMoreCmds, LoadProblem, OtherMessage, FileNotAvail, MissDisk, NotFound

MskCtrlCodes

Function: Remove control codes (any non-ASCII characters) from a text buffer.

Pass: r0 - start of buffer.

endoftext - one byte past end of buffer.

txttype - always set this to zero.

Return: the buffer is altered.

endoftext - points to the new end of text + 1. The buffer will decrease in size if any unwanted bytes have been removed from it.

Destroys: a, y, r0, r2

Description: If your command has just loaded a buffer with text, you can call this routine to strip any characters from it that GEOS is unable to display on the screen. You will be left with only ASCII text plus any carriage returns, tabs, or shifted-spaces. You should always set txttype to zero, just in case the 'type' command has recently been used. The type command uses this variable for various purposes. One of the uses is to tell MskCtrlCodes if the buffer it is converting is a Text Scrap or a text album. In this case, MskCtrlCodes will strip the first escape ruler from the buffer. You could also use this routine for this purpose, however geoSHELL has other routines for doing this also. This feature is quite specific to the type command, so you would normally just clear the variable txttype before calling MskCtrlCodes.

ConvToAscii also calls this routine when it is converting from PetASCII to ASCII, so if your command uses ConvToAscii, calling MskCtrlCodes will not change anything in your buffer, since it will already have been done.

Example:

```
      ...
      jsr   ReadFile   ;use GEOS to read in a linked
                 ;chain of blocks.
      txa        ;any problems?
      beq   10$  ;branch if not.
      jmp   LoadProblem ;display an error and exit.
 10$
      LoadW r0,#textBuffer   ;point r0 to our new text.
      MoveW r7,endoftext     ;GEOS left r7 pointing to
                  ;the last byte plus one.
      LoadB txttype,#0  ;always do this. jsr   MskCtrlCodes     ;strip
unwanted characters.
      ...
      ...         ;continue on...
      ...
```

See Also: AdjTxtPointer, Message, OtherMessage, DsplyLine, DsplyString, CarrReturn, ClearLine, ClearBoth, ClearRemainder, StripRulers, set_XY_pos, save_XY_pos, CkESC_RULERS, ConvToAscii, DispText

MissDisk

Function: Display the message 'Missing Disk?!'.

Pass: nothing

Return: nothing

goodflag - meaningless

Destroys: no guarantees

Description: This routine displays a built-in message informing the user of a missing disk. It will return to your command so that you may act on it further if needed.

Example:

```
      ...
      lda   #10   ;check for a disk in
      jsr   CkForDisk  ;drive C.
      bit   goodflag   ;is there one?
      bmi   10$   ;branch if so.
      jsr   MissDisk   ;display a message.
      jmp   NoMoreCmds ;and exit.
 10$
      ...
      ...         ;otherwise,
      ...         ;continue on...
      ...
```

See Also: MissgFilename, NoMoreCmds, LoadProblem, OtherMessage, FileNotAvail, Message, NotFound

MissgFilename

Function: Display the message 'Bad Parameter!' without returning.

Pass: nothing

Return: not applicable

Destroys: not applicable

Description: This routine displays a built-in message informing the user of a bad parameter. It will not return to your command, since it then jumps through NoMoreCmds.

Example:

```
      ...
      jsr   ParamName   ;check if the user typed in
      bit   goodflag    ;a filename.
      bmi   10$   ;branch if so.
      jmp   MissgFilename    ;tell him, bad parameter.
 10$
      ...
      ...         ;otherwise,
      ...         ;continue on...
      ...
```

See Also: Message, NoMoreCmds, LoadProblem, OtherMessage, FileNotAvail, MissDisk, NotFound

nogood

Function: Usually identifies a failure in a routine.

Pass: nothing

Return: goodflag - bit 7 set to zero.

Destroys: nothing

Description: geoSHELL uses this routine extensively. nogood and yesgood are the universal routines for identifying successes and failures. This routine actually does nothing more than clear a flag (bit 7 of goodflag) to give an indication to a calling routine that a failure has occured. All the calling routine has to do is check goodflag with the bit instruction. It will be plus if nogood cleared it. As an example, the routine 'CkForDisk' will jump through nogood upon exit of the routine if it did not find a formatted disk in the drive.

It's true that the carry flag or some other form of error checking could have been used, but it's also possible that the x register or the carry flag could change before we are ready to check it. We have more control over how goodflag is affected, and can do some processing before checking it if needed.

Example:

```
      ...
      jsr   OpenDisk   ;call a GEOS routine.
      txa         ;check the value returned in x.
      beq   10$   ;branch if OpenDisk was OK.
      jmp   nogood     ;signal the calling routine
            ;that this routine failed.
 10$
      ...         ;continue with more.
```

See Also: yesgood
NoKeypress

Function: Wait for the user to release the keyboard.

Pass: nothing

Return: goodflag - meaningless, although the routine will always return with goodflag's bit 7 set.

    the keyboard - completely released.

Destroys: a, x, y

Description: This routine is only used when you want the user to release any keys that he may be pressing. When you desire user input of some sort, you might want to call this routine to first insure that no keys are being pressed and then proceed to look for a keypress with YesKeypress. Refer to the Wait routine, as it will call both of these routines for you.

One problem with using this routine however would be that GEOS will still return the character code of the key if the user was pressing one. If you merely want the user to let go of the keyboard, but do not need keyboard input, then this routine will do just fine. If you want to clear out the keypress that the user was holding, then call ClrKeyData instead. That routine will also wait for the user to release the keyboard.

Example:

```
      ...
      ...         ;some of your command's code.
      ...
      jsr   NoKeypress ;wait for no keys being
            ;pressed.
      ...
      ...         ;continue with further
      ...         ;parts of your command.
```

See Also: YesKeypress, ClrKeyData, Wait, CkKeyboard

NoMoreCmds

Function: Exit from just about anything and return complete control back to geoSHELL.

Pass: nothing

Return: not applicable

Destroys: not applicable

Description: On just about any kind of error that a user might create, or if you just want to get out of a situation, or if you want to end your command and not allow any further commands to be processed, call this routine. Certain situations occur when a command is running, and it is up to the programmer of the command to check for any possible situation that might arise, and when necessary call this routine to end a line of commands. As an example, 'fcopy' will call this routine when it is unable to find the desired file that the user typed. First it displays the necessary message and then jumps to NoMoreCmds so that no problems will arise with any other commands that might follow.

Example:
```
      ...
      ...         ;parts of your command.
      ...
      jsr   DoWhatever ;jsr to another routine.
      bit   goodflag   ;check for an error on return.
      bmi   10$   ;branch if routine was ok.
      jsr   DisplayError     ;you might want to have a
                  ;routine to display an error.
      jmp   NoMoreCmds ;stop any further commands.
 10$
      ...
      ...         ;continue with your command.
      ...
```

See Also: ExitCommand, IncCmdPointer

NotFound

Function: Display a 'FILENAME Not Found!' message.

Pass: a2 - pointer to null-terminated filename.

Return: nothing

Destroys: no guarantees

Description: In a case where your command fails to find a desired filename that perhaps the user is requesting, you can call this routine to display a message that the filename in question is not available. Whatever the filename is that the user typed in will be displayed followed by 'Not Available!'. The routine will return to your command where you can decide how to continue processing. Just be sure that a2 is pointing at the null-terminated filename that you wish to include in the message.

Example:

```
      ...
      jsr   ParamName   ;get the filename.
      bit   goodflag    ;is it valid?
      bmi   10$   ;branch if so.
      jmp   MissgFilename    ;display an error.
10$
      jsr   CkAllDrives ;look for it.
      bit   goodflag    ;did we find it?
      bmi   20$   ;branch if so.
      jsr   NotFound    ;display 'not found'.
      jmp   NoMoreCmds  ;and in this case we
                  ;will just exit.
 20$
      ...
      ...         ;continue on...
      ...
```

See Also: Message, NoMoreCmds, LoadProblem, OtherMessage, FileNotAvail, MissDisk, MissgFilename

Only128

Function: Inform the user when a 128 command is being used on a 64 and exit.
Pass: nothing

Return: not applicable

Destroys: not applicable

Description: This routine will display the message '128 Only' and end by jumping to the routine NoMoreCmds, thereby bringing all commands to a halt. This is a nice way to get out of a command when it will only work on a 128, but an attempt is being made to use it on a 64. Simply test the variable 'screenmode' to determine the machine that is being used.

Example:

```
      bit   screenmode ;check the mode.
      bvc   10$   ;branch if a 128.
      jsr   ClrTrnsName ;clear the command.
      jmp   Only128     ;do an error and exit.
 10$
      ...
      ...         ;this is a 128, so
      ...         ;continue on...
      ...
```

See Also: CkModeFlag

OpenCurPartition

Function: Open the currently active partition after changing to a different one.

Pass: Certain internal variables must be set by calling GetRealDriveNum and SetPartParams beforehand.

InitForIO must be called first.

PurgeAllTurbos must be called first.

Return: The partition the user is seeing will be reopened.

Destroys: a, x, y, r0-r15

Description: This routine is a lower level geoSHELL routine that requires a certain amount of setup to work properly. Normally you would want to call other routines such as ResetFromPath instead of this one. But in the case of a filecopier, for instance, you may wish to bypass the routines that take you back into GEOS while moving data around using the drive's own DOS. In that case, this routine will reopen the current partition, while your command can maintain it's own variables for another partition that you are working with.

In order to make this routine work properly, geoSHELL needs to figure out what the current partition is first as well as the correct device number. You do that by calling other geoSHELL routines, GetRealDriveNum and SetPartParams, to properly set up the internal variables.

Example:
;this could be part of the code that you might use for a command
;that would switch among partitions.

```
      ...
      jsr   SetPartStuff     ;set up the variables for
                  ;the current partition
      bit   goodflag   ;does this device use
                  ;partitions?
      bmi   10$   ;branch if so.
      jmp   NotPartDevice    ;otherwise display an error.
 10$
      jsr   PurgeAllTurbos   ;SetPartParams also does this.
      jsr   InitForIO
      ...
      ...         ;do whatever you had planned.
      ...
      jsr   OpenCurPartition ;open the current partition.
      jsr   DoneWithIO
      ...        ;continue with more code...  ...
      rts
SetPartStuff:
      lda   PRESDRIVE
      jsr   GetRealDriveNum  ;is this a DOS type device?
      bit   goodflag
      bpl   10$   ;branch if not.
      jsr   InitForIO
      lda   REALDRIVE  ;this was set by GetRealDriveNum
      jsr   SetPartParams    ;set some internal variables.
                  ;this will set goodflag
                  ;if the device has partitions.
      jsr   DoneWithIO
 10$
      rts
```

See Also: InitCurPartition, ResetFromPath, GetRealDriveNum, SetPartParams, OpenPartition, GetCMDType, FindCMDType, FindRamLink, PurgeAllTurbos, OpenPathPartition, SetPartParamsOpenGS

Function: Locate and open geoSHELL to gain access to it's VLIR records.

Pass: nothing

Return: goodflag - set if successful.

geoSHELL is now an open VLIR file, and so everything pertaining to a call to OpenRecordFile pertains here.

pathload - set if geoSHELL was found on the path partition.

DRVFOUNDON - device number (8-11) of the drive that geoSHELL was found on.

curDevice and curDrive - the currently open drive.

PRESDRIVE - unchanged.

Also anything that OpenRecordFile returns is returned with this routine, such as the Index Table in fileHeader, and dirEntryBuf.

Destroys: a, x, y, r0-r15, a2

Description: This routine will search for geoSHELL on all available drives as well as a 'path' partition if one is set and jump through yesgood if it is found. If found, a call to OpenRecordFile is made and geoSHELL is left as an open VLIR file. If it was found on the path partition, then 'pathload' is also set. In this case, after closing geoSHELL with a call to CloseRecordFile, a call should also be made to ResetFromPath. Having geoSHELL open like this allows access to make modifications such as what the 'custom' command does. The search for geoSHELL is performed through CkPathOrder.

Example:

```
      ...
      jsr   OpenGS      ;open the geoSHELL file.
      bit   goodflag    ;was it successful?
      bmi   10$   ;branch if so.
      jsr   NotFound    ;display that geoSHELL was
      jmp   NoMoreCmds  ;not found and exit.
 10$
      ...
      ...         ;now we can do what we
      ...         ;want with the geoSHELL
      ...         ;file that is the one
      ...         ;that is currently in control.
      ...
```

See Also: OpenGSIndex

OpenGSIndex

Function: Load geoSHELL's VLIR index block into fileHeader.

Pass: nothing

Return: goodflag - set if routine was successful.

pathload - set if geoSHELL was found on the path partition.

DRVFOUNDON - device number (8-11) of the drive that geoSHELL was found on.

fileHeader - contains geoSHELL's index table if successful (goodflag set)

dirEntryBuf - geoSHELL's directory entry.

curDevice and curDrive - the currently open drive.

PRESDRIVE - unchanged.

Destroys: a, x, y, r0-r15, a2

Description: This routine will find geoSHELL and load it's index sector into memory at fileHeader. If bit 7 of pathload is set, then geoSHELL was found on the path partition. The path partition is still open if so, and a call to ResetFromPath should be made before your command exits, or the user will have to do it himself (not desireable). The geoSHELL file is not left as an open VLIR file with this routine, only the index table is loaded into memory.

Example:

```
GetGSIndex:
      jsr   OpenGSIndex ;get geoSHELL index sector.
      bit   goodflag    ;was it successful?
      bmi   10$   ;branch if so.
      rts         ;goodflag is still cleared.
 10$
                  ;geoSHELL's index sector is
                  ;now in memory at fileHeader.
                  ;We will now restore the drive.
                  ;This could also be done by
                  ;the calling routine if it
                  ;needs to keep geoSHELL
                  ;available while performing
                  ;it's operation on it.
      jsr   ResetFromPath    ;reset the drive to it's previous
                  ;partition in case geoSHELL
                  ;was found on the path
                  ;partition.
      jmp   yesgood    ;tell the calling routine               ;that
geoSHELL's index is
                  ;in memory.
```

See also: OpenGS

OpenPartition

Function: Open a partition on a CMD device.

Pass: r0 - pointing to a string with 'CP' and the ascii characters representing the desired partition to open.

a - the real device number of the desired CMD device.

cmdlength - the length (3-5) of the string that r0 is pointing at.

PurgeAllTurbos, InitForIO, and SetPartParams must be called just before calling this routine.

Return: goodflag - set if the desired partition was opened.

Destroys: no guarantees

Description: This routine is the one you would call when you wish to open a partition on a CMD device. It does not have to be the currently active drive either. Any connected CMD device may be controlled with this routine. Be sure to call PurgeAllTurbos and then InitForIO first. Also, you will need to call SetPartParams before calling OpenPartition. This is so that the type of partition that is currently open on the desired drive can be compared to the one that you are attempting to open. As long as they are of the same type, then the routine may open it. This is to prevent any problems with the disk driver that is being used.

All the low-level handling of the device is taken care of for you with this routine. This makes it a very easy task to open any partition. The only hitch with doing this is that you should only use this on one device at a time. Because geoSHELL maintains some internal variables for you that will be used when you do a subsequent call to OpenCurPartition or InitCurPartition. Those routines will restore the partition that was currently open before calling OpenPartition. That is another reason for calling SetPartParams first. In the case of the 'fcopy' command, if the user copies to a destination by specifying a particular partition instead of a drive, the source drive is always the current partition. If the destination is on the same drive, then fcopy has no problem. If the destination is a different drive, then then OpenCurPartition will consider the current partition to be the one that is currently open on the destination drive. A call to OpenCurPartition or InitCurPartition would then reopen the original partition on the destination drive. This may sound complicated but in reality it is not. As long as the current partition is restored, another call can be made to open a partition on another device or the same one.Just remember to work with one partition at a time or you can work with a currently open partition and one that is not currently open when you need to work with two partitions at the same time.

Example:

```
;this example might look fairly lengthy, but the bulk of it is
;taking the parameter that the user typed in and building a
;string that we will pass on to OpenPartition. This example
;performs some sort of whatever to a desired partition that
;the user chooses. When finished, it will reopen the original
;partition.
;The command that the user types in might look like this:
;commandname 45
;This example command does not expect a fixed length parameter
;and so a space can terminate the 45 in this case. The 45
;represents partition number 45, the one the user wants this
;command to act on. A partition number such as 45 could be
;anything from 1 to 255 or the maximum allowed by the device.

     lda   PRESDRIVE  ;this acts on the current
                ;drive as the user sees it.
     jsr   GetRealDriveNum  ;just in case it's an RL or RD.
                ;This also calls PurgeAllTurbos.
     LoadB paramsize,#2     ;let's keep track of the size
                ;of our parameter. (it's at
                ;least one plus a space)
     ldx   #2   ;first fill string with spaces.
     lda   #' '
 10$
     sta   partNumber,x
     dex
     bpl   10$
     lda   ParamTable ;we need at least one character.
     jsr   CkTerminators    ;is this a valid character?
     bne   20$  ;branch if so.
 15$
     jmp   MissgFilename    ;this will display
                ;'Bad Parameter' and exit.
 20$
```

```
      sta   partNumber+0     ;save this first character.
      lda   ParamTable+1     ;get the next character.
      jsr   CkTerminators    ;is it a character?.
      beq   40$   ;branch if not.
      inc   paramsize  ;the parameter is at least
                  ;two characters now.
      sta   partNumber+1
      lda   ParamTable+2
      jsr   CkTerminators
      beq   40$
      inc   paramsize  ;the parameter is at least
                  ;three characters now.
      sta   partNumber+2
      lda   ParamTable+3     ;there must be a space or
                  ;terminating character after            ;the last
character in the
                  ;parameter.
      jsr   CkTerminators    ;well?
      bne   15$   ;branch if not.
 40$
      jsr   InitForIO  ;pop out of GEOS.
      lda   REALDRIVE  ;get the device number.
      jsr   SetPartParams    ;set some internal variables
                  ;and check if this device has
                  ;partitions.
      bit   goodflag   ;well?
      bpl   50$   ;branch if not.
      LoadW r0,#partString   ;point r0 to the string we
                  ;just built up.
      LoadB cmdlength,#5     ;any spaces in the string are
                  ;ignored by CMD's DOS.
      jsr   OpenPartition    ;open the partition.
 50$
      jsr   DoneWithIO ;pop back into GEOS.
      bit   goodflag   ;did the partition get opened?
      bmi   60$   ;branch if so.
      LoadW r0,#cantOpen     ;otherwise, let's tell the
      lda   #(IN_TWO|TR_CR)  ;user we couldn't get to the
      jsr   OtherMessage     ;partition.
      jmp   NoMoreCmds ;and return control to him.
 60$
      ...
      ...         ;here we can do what the
      ...         ;command is designed to do
      ...         ;with the partition that we
      ...         ;just opened.
      ...
      jsr   InitCurPartition ;restore the original
                  ;partition.
      lda   paramsize  ;we were keeping track of this.
      jsr   IncCmdPointer    ;point past our command and
                  ;parameter.
      jmp   ExitCommand ;and do a clean exit.
```

```
partString:
      .byte "CP"
partNumber:
      .byte "1  " ;this gets changed.
cantOpen:
      .byte "Unable To Open Partition!",0
```

See Also: InitCurPartition, OpenCurPartition, ResetFromPath, OpenPathPartition, SetPartParamsOpenPathPartition

Function: Open the path partition.

Pass: nothing

Return: goodflag - set if successful.

pathcheck - set to the device number of the drive containing the path partition.

REALDRIVE - contains the real device number of the drive containing the path partition.

Destroys: a, x, y, r0-r15

Description: This routine will open the partition of the device that has been defined by the path command to be the path partition. Upon return, if goodflag is set, the routine was successful and the device containing the path partition is now open along with the correct partition. Since this device is now open, curDrive will contain the device number as well as pathcheck. In the case of a RamLink or RamDrive being used as a RAM_1581, REALDRIVE will contain the real device number of the device for DOS purposes. In these cases, GEOS might see the device as 8-11, but the DOS in the RL or RD is seing it as 12 or higher.

When finished with the path partition, be sure to call ResetFromPath to restore the device to the partition that the user is currently seeing.

If no path has been established, or if the path partition has been defined, but cannot be opened for whatever reason, the routine will exit through nogood.

Example:

```
      ...
      jsr   OpenPathPartition
      bit   goodflag
      bmi   10$
      rts
 10$
      ...
      ...        ;do what you need to do.
      ...
      jsr   ResetFromPath
      jmp   yesgood
```

See Also: InitCurPartition, OpenCurPartition, ResetFromPath, OpenPartition

OpenWrFile

Function: Load a file's VLIR index into memory.

Pass: r0 - pointer to a null terminated filename.

Return: goodflag set if successful.

      fileHeader contains the file's index sector.

Destroys: a, y ,r1, r4-r6

Description: In cases where you need a VLIR file's index in memory, simply point r0 to the filename and call this routine. The index will be loaded in fileHeader. If the filename was typed in by the user, call ParamName to point r6 and a2 to the filename and then copy a2 or r6 to r0. You can then call this routine. This routine does not leave any VLIR files open, and only checks the currently active drive.

Example:

```
;here is how you would search all drives for a file and then
;load it's VLIR index into memory.

GetIndex:
      jsr   ParamName  ;what filename did
                  ;the user type in.
      bit   goodflag
      bmi   10$   ;branch if the user entered
                  ;a filename.
 5$
      rts         ;otherwise, exit with
                  ;goodflag reset to zero.
 10$
                  ;a2 now points to the filename
                  ;that the user requests.
      jsr   CkAllDrives ;find the file.
      bit   goodflag
      bpl   5$
                  ;a2 now points to the
                  ;real filename even if the
                  ;user used wildcards.
      MoveW a2,r0
      jmp   OpenWrFile ;this ends the routine.
                  ;goodflag will represent the
                  ;success or failure of loading
                  ;in the file's VLIR index.
                  ;if goodflag has bit 7 set,
                  ;then the index is loaded.
                  ;if not, then maybe the file
                  ;was not a VLIR type file.
```

See Also: OpenGS, OpenGSIndex, GetHeaderOpen_GC_Channel

Function: Check the user port via a geoCable for the presence of a printer.

Pass: nothing

InitForIO must be called before accessing this routine.

Return: x - zero means the printer is ready for input. (this is one of the few routines that doesn't affect goodflag)

Destroys: a, x, y

Description: This routine works with a parallel printer via a geoCable connected to the user port. It will check to make sure the printer is there. It is not necessary to call this routine if you already know that a printer is ready for input. However, if your command is uncertain that the printer is connected and running, call this routine. If x is zero upon return, the printer is connected and ready. Otherwise the printer might be there but not responding, such as if it is still printing from it's buffer or if it has been switched offline by the user. If you don't access this routine before sending any bytes, your command will just hang and the computer will appear locked up until the user realizes that the printer is not available and turns the printer on. In most cases, the user doesn't think about this, he will think your command has caused a crash.

Example:

```
      ...
      jsr   InitForIO   ;pop out of GEOS.
      jsr   Open_GC_Channel   ;check for a printer.
      txa         ;check if x = 0.
      beq   10$   ;branch if so.
      jsr   DoneWithIO  ;pop back into GEOS.
      jmp   nogood      ;signal calling routine of error.
 10$
      ...
      ...         ;continue on...
      ...
```

See Also: Send_GC_Byte

OtherMessage

Function: Display a message.

Pass: r0 - pointer to text string to display.

      a - formatting information.

Return: nothing

Destroys: no guarantees

Description: This routine will display whatever message you wish to display in the geoSHELL window. You are limited to the length of the current screenwidth that is being used, 40 or 80 columns. Just point r0 to the null terminated text that you wish to display and load the accumulator with the formatting information and call this routine.

Example:

```
;this particular example is supposed to be some sort of
;command that acts on a GEOS file. If the file exists and
;it is not a GEOS file, then OtherMessage will be used
;to inform the user that this file is not a GEOS file.

      ...
      jsr   FindParName ;this won't return if error.
      lda   dirEntryBuf+0    ;first check the filetype.
      and   #%00001111
      cmp   #REL  ;is it a REL file?
      beq   20$   ;branch if so.
      lda   dirEntryBuf+22   ;is this a GEOS file?
      bne   30$   ;branch if so.
 20$
      MoveW a2,r0 ;tell the user that
      lda   #(IN_ONE|TR_ONE) ;this
      jsr   OtherMessage     ;file...
      LoadW r0,#notText ;is not
      lda   #TR_CR     ;a
      jsr   OtherMessage     ;GEOS file.
      jmp   NoMoreCmds ;and exit.
 30$
      ...
      ...         ;this is a GEOS file, so
      ...         ;we can continue on...
      ...

notText:
      .byte "Is Not A GEOS File!",0
```

See Also: Message, NoMoreCmds, LoadProblem, NotFound, FileNotAvail, MissDisk, MissgFilename

APPENDIX A

MEMORY USAGE

geoSHELL uses a fair amount of memory in it's operation and does not leave a great deal for you to use with your command. However with some thought and making use of geoSHELL resources, you will find that you do not need much memory to do even complex tasks. The smaller your command, the faster you will create it anyway. And perhaps, the more commands you will be able to create, also.

Even still, there is almost 6K of programming space that you are free to use for your command. This is sufficient for all but the largest of commands. The command 'fcopy' works well with this amount of memory and even includes it's transfer buffer in this space. The 'type' command works just fine in this space also, including the area it uses for it's text buffer. Usually when a command needs more room than this, it will be because it needs a larger area for a buffer, rather than for program code. The 'dcopy' command likes a big buffer. The bigger, the better. For this reason, it will use the area from $a000 to $a000+8000, the area occupied by the 40 column foreground screen. If you've ever used the dcopy command in 40 column mode, you've noticed that it blanks the screen all except for enough room at the bottom to display a message to the user. With both the foreground and background colors being the same, the user is unaware of all the activity going on with the screen memory.

Your command is always loaded in at $6800 and is jumped to at that address. Obviously, if you make use of a GEOS printer driver, you will have to limit yourself to the area from $6800-$78ff, since the printer driver will load in at $7900. Otherwise, your program code may extend on up to $7f3f. You are free to use the area up to $7fff but geoSHELL won't allow your command to load if it is large enough to fill the area between $7f40 and $7fff. This should have been changed as geoSHELL was being developed, but it never was, and this oversight wasn't discovered until this manual was being put together. Normally, this shouldn't be a problem, since most commands need some free space just for variable storage.

In looking through this listing of memory that geoSHELL uses, you might find a byte here and a byte there that you can use. You might even find a whole page of memory here or there. And there will be locations outside of your command that you will be using anyway.

Here is a simplified memory map of geoSHELL:

$0400 - $10ff	Variable Storage Area
$1100 - $16ff	Startup Buffer
$1700 - $1846	Jump Table
$1847 - $66ff	The geoSHELL main code. Do not attempt to alter any of this area.
$6700 - $67ff	EnOfShell - Routines that read a drive's error channel use this area. Free for use at any other time.
$6800 - $7fff	Transient Command Area

It is not really necessary to know the actual location of any variable or routine since everything available in geoSHELL can be accessed with a label. All of the labels are listed in this document with explanations of their uses and are available when you link the file GSVariables.rel with your source code. The variables are also defined for you in this appendix. You might also wish to study the file ShellEquates to discover some of the equates that are available to you when creating your source code. Some of these equates and the available variables are discussed throughout the descriptions of the geoSHELL routines. You should become familiar with these equates and variables and use them to your advantage. On the 128, geoSHELL uses only front ram (bank 1) so that you are free to use whatever back ram is not being used by the GEOS kernal as needed. geoSHELL doesn't care if your command uses any of the zero page locations, r0-r15. You will just have to keep track of what routines in geoSHELL and the GEOS kernal might trash any of these that you are using. If necessary, use PushStuff and PopStuff to preserve these when needed. Also, some of the registers from a0 through a9 are available. Two of them are used heavily by geoSHELL and your command, a2 and a4. Normally, you will let geoSHELL manage a4 for you. a4 is the command pointer. It is always pointing at the current parameter that is being used or the next command that will be used. a2 is the universal filename pointer. Here is the list of the a0-a9 registers:

a0	geoSHELL's user input buffer pointer. (not a good idea to mess with this one)
a1	not used.
a2	universal filename pointer.
a3	not used.
a4	command pointer. (use this properly)
a5	used by time and directory routines.
a6	used by time and directory routines.
a7	not used.
a8	used by directory routines.
a9	not used.

As you can see, there are plenty of free zero page registers that you can use as needed.
One of the geoSHELL variables that you should remember aboutis called 'PRESDRIVE'. This variable holds the device number of the currently active drive that the user thinks he is working from. In other words, if the drive letter on the screen looks like this...
B>
...then PRESDRIVE will contain the value 9. None of the routines that geoSHELL uses for searching the different drives will alter PRESDRIVE. This way, any drive can be accessed by your routine and the user will still keep the same drive as his current one. geoSHELL let's the user control which drive is the currently active one. He can change it by entering a letter followed by a colon. If do have a need to change the currently active drive, then simply change the value in PRESDRIVE. Just make sure that the drive is online before doing so.
This is a listing of all the variables that will always be available, even in future versions of geoSHELL. They begin at StOfRamsect ($0400) and end at EnOfRamsect ($16ff). Actually, the last $0600 bytes of this area is the startup buffer.

```
StOfRamsect:          ;used as a reference point for
```

```
                  ;initializing this area.

screenmode: .block 1   ;current mode of the machine
            ;that is being used.
            ;#%10000000 =80 column 128
            ;#%00000000 =40 column 128
            ;#%01000000 =64
PRESDRIVE:  .block 1   ;device number of the currently
            ;active drive as seen by
            ;the user.
goodflag:   .block 1   ;universal success/error flag.
DRVFOUNDON: .block 1   ;drive a file is found on.
STUPMODE:   .block 1   ;bit7 set =startup or exec mode.
FKEYMODE:   .block 1   ;bit7 set =function key mode.
NUM_IN_NAME:      .block 1   ;number of characters in a
            ;filename.
ParamTable: .block 44  ;table that holds the parameter
            ;for a command.
comchars:   .block 1
cur_Y_pos:  .block 1   ;current vertical pixel position
            ;of cursor.
cur_X_pos:  .block 2   ;current horizontal pixel
            ;position of cursor.
cur_line:   .block 1   ;current line (0-11) of cursor.
cur_char:   .block 1   ;current character (0-39 or 0-79)
            ;position of cursor.
tempFileName:     .block 17  ;temporary storage.
curTransName:     .block 17  ;transient command that is
            ;currently in memory.
scrnwidth:  .block 1   ;40 or 80.
domessages: .block 1         ;a zero here will prevent
            ;messages from being displayed.
            ;bit 7 set will allow messages.
cur_keyData:      .block 1
CMD_POINTER:      .block 1   ;normally zero during execution
            ;of a command. Leave this alone.
tempDrive:  .block 1   ;temporary storage used for
            ;status and directory displays.
typeset:    .block 1   ;used by DispText.
cmpwidth:   .block 1   ;used by DispText.
paramsize:  .block 1   ;holds the size of the parameter
            ;that is loaded in ParamTable.
pathload:   .block 1   ;this is set to 128 when a
            ;file is found on the
            ;specified path partition.
Asc_String: .block 6   ;ascii decimal strings are
            ;built here.
endecho:    .block 1   ;used by DispText.
sourceType: .block 1   ;used by copy commands.destType:   .block 1
      ;used by copy commands.
sourceDrive:      .block 1   ;used by copy commands.
destDrive:  .block 1   ;used by copy commands.
sourcePartition:  .block 1   ;used by copy commands.
destPartition:    .block 1   ;used by copy commands.
```

```
;the custom command saves the following locations from StOfDefs
;to EnOfDefs to the geoSHELL file on disk.

StOfDefs:
videomode:  .block 1    ;used only for 80 columns.
                  ;0 = monochrome
                  ;128 = color
distype:    .block 1    ;used by 'ddate' and 'dtype'.
fastdirectory:    .block 1    ;used by 'fast' and 'slow'.
defStName:  .block 17   ;"Startup",0,0,0,0,0,0,0,0,0

coltable:         ;80 column color table.
backcolor:  .block 1    ;screen background color.
shellcolor: .block 1    ;shell color.
textcolor:  .block 1    ;text color.
padcolor:   .block 1    ;pad background color.
brdrcolor:  .block 1    ;screen border color.
back40pattern:    .block 1    ;default background pattern for
            ;40 column.
back80pattern:    .block 1    ;default background pattern for
            ;80 column.
clr80pattern:     .block 1    ;default background pattern for
            ;80 column RGB mode.

      .block 3    ;these bytes are unused.

curPrinter: .block 1    ;printer device number 4 or 5.
curSecond:  .block 1    ;printer secondary address.
crtextstring:     .block 3
curCharSet: .block 1    ;character set to send to
            ;printer when @p or @g is used.
            ;  0 = Ascii
                  ;128 = PetAscii

      .block 10   ;reserved for future use...
EnOfDefs:

TrRtnLAddr: .block 16   ;do not mess with this!
TrRtnHAddr: .block 16   ;do not mess with this!

      .block 4    ;reserved for future use.

RAMLINK:    .block 1    ;the real device number for
            ;RamLink or RamDrive is set
            ;here after a call to
            ;FindRamlink. This is
            ;normally only needed if there
            ;is a RAM1581 used.
cmdtype:    .block 3    ;contains FD,HD,RD, or RL.
                  ;cmdtype+2 can be from 1-4
                  ;1=FD drive
                  ;2=HD
                  ;3=RD
```

```
                ;4=RL

KeyBuffStart:

Key1Buff:   .block 81   ;buff_size =81
Key2Buff:   .block 81   ;each buffer here holds
Key3Buff:   .block 81   ;a function key definition.
Key4Buff:   .block 81
Key5Buff:   .block 81
Key6Buff:   .block 81
Key7Buff:   .block 81
Key8Buff:   .block 81

restartbyte:     .block 1
CmdNumber:  .block 1
zeroflag:   .block 1
hotkeycommand:   .block 2
top_rectangle:   .block 1
bot_rectangle:   .block 1
left_rectangle:  .block 2
right_rectangle: .block 2
temp_line:  .block 1
temp_check: .block 1   ;used for temporary storage of
                ;a, x, or y registers.
ck_year:    .block 1
ck_month:   .block 1
ck_day:     .block 1
ck_hour:    .block 1
ck_minute:  .block 1
CMD_COUNT:  .block 1
lastblock:  .block 2
filesiz:    .block 2
lstbyte:    .block 2
headlocation:    .block 6
FIRSTDRIVE: .block 1
DRIVECHECK: .block 1
foundtrans: .block 1
inserty:    .block 1
stparsize:  .block 1
scancheck:  .block 1
check_key:  .block 1
nosearchmessage: .block 1
saver0:     .block 2
numfillbytes:    .block 1
numfillcopies:   .block 1
numfillstrings:  .block 1
stringlocation:  .block 2
dublclickspot:   .block 1
drivesChecked:   .block 4
PRNTMODE:   .block 1   ;bit 7 set =send output to the          ;serial
printer in addition to
            ;the screen.
            ;bit 6 set =send output to the
            ;geoCable.
```

```
strtoprint: .block 2    ;temporarily save r0 here until
            ;ready to send the string that
            ;it points at to the printer.
savex:      .block 1    ;temp storage for x.
line_count: .block 1
cursorStatus:     .block 1    ;bit 7 set means cursor is
            ;inverted.
                  ;cleared means cursor is normal.
cursorOn:   .block 1    ;high bit set means cursor is
            ;active.
                  ;clear means cursor is inactive.
st_keyData: .block 1
reg24:      .block 1
gs_stack:   .block 256
stackpointer:     .block 1
stupsequence:     .block 1
ActHWide:   .block 1
ActLeft:    .block 1
svkeyVector:      .block 2
Dir_Entry:  .block 2
Dir_Type:   .block 1
StripZFlag: .block 1
ParentName: .block 17
PermName:   .block 17
DiskName:   .block 17
filetoload: .block 17
WILDCARD:   .block 1
DRV_CKD:    .block 1
FILECOUNTER:      .block 1
NxtBlock:   .block 2
X_PARAM:    .block 2
Y_PARAM:    .block 1
MoveXTable: .block 2
MoveYTable: .block 2
NOSPACE:    .block 1
stackReturn:      .block 2
stripStack: .block 1
NMCreturn:  .block 2
DISPBUFFER: .block 84
WrModeFlag: .block 1
test1string:      .block 17
test2string:      .block 17
clrlineflag:      .block 1
clearcount: .block 1
messflag:   .block 1
msgchars:   .block 1
messlocation:     .block 2
horizcount: .block 1
kboardoff:  .block 1
tempr0:     .block 2
gstructype: .block 1gfiletype:     .block 1
cfiletype:  .block 1
texttrsc:   .block 2
nozero:     .block 1
```

```
gscolcount: .block 1
headcolor:  .block 1
colbyte:    .block 1
staddress:  .block 2
clickspot:  .block 1
tempy:      .block 1         ;temp storage of y register.

      .block 16  ;currently unused.

TrRtnNum:   .block 1
tcomnum:    .block 1
pathcheck:  .block 1
REALDRIVE:  .block 1
cmdlength:  .block 1
curparnum:  .block 1
curpartype: .block 1
curdrvtype: .block 1
errorck:    .block 1
partdir:    .block 1
syscount:   .block 1
cmdtosend:  .block 1
lastck:     .block 1
runorprint: .block 1
textpointer:     .block 1
interpret:  .block 1
endoftext:  .block 2   ;this contains the byte past
            ;the last byte read in.
txttype:    .block 1   ;0=SEQfile  (CBM or GEOS)
                  ;1=SCRAPfile (Text Scrap)
                  ;2=WRITEfile (GeoWrite file)
                  ;3=VLIRfile (but not GeoWrite)
                  ;4=ALBUMfile (text album)

nextbatch:  .block 2
ovrflbyte:  .block 1
curpage:    .block 1
scrcode:    .block 2
asciicode:  .block 2
petcode:    .block 2

cur_table:             ;variables defining the
            ;location and size of the
            ;geoSHELL window.
lft_shell:  .block 2         ;left position word
            ;(0-319 or 0-639).
rht_shell:  .block 2         ;right position word
            ;(0-319 or 0-639).
left_shell: .block 1         ;left position of window on
            ;screen in bytes.
wid_shell:  .block 1         ;width of the bitmap in bytes.
lft_window: .block 2rht_window:    .block 2
col0: .block 2
rht_column: .block 2
en_cur_table:
```

```
;the internal memory buffers holding a copy of the text
;that appears on the screen for each line in the geoSHELL
;window, lines 0-11.

buff_0:     .block 81
buff_1:     .block 81
buff_2:     .block 81
buff_3:     .block 81
buff_4:     .block 81
buff_5:     .block 81
buff_6:     .block 81
buff_7:     .block 81
buff_8:     .block 81
buff_9:     .block 81
buff_10:    .block 81
buff_11:    .block 81

      .block 3   ;extra added here to
            ;protect bufmem10.

;copies of the ten most recent command line strings.
bufmem10:   .block 80
bufmem9:    .block 80
bufmem8:    .block 80
bufmem7:    .block 80
bufmem6:    .block 80
bufmem5:    .block 80
bufmem4:    .block 80
bufmem3:    .block 80
bufmem2:    .block 80
bufmem1:    .block 80

CmdBuffer:  .block 82
      .block 10  ;currently unused.

StUpBuffer: .block $0600     ;buffer for startup
            ;and exec files.
EnOfStUpBuffer:
EnOfRamsect:
```

THE JUMP TABLE

Here is a list of all the available routines that are easily accessible in geoSHELL through it's jump table. The jump table begins at $1700. These routines are listed in the order as they appear in the jump table. For a complete description of each routine, refer to section 2 of this manual.

$1700 InitgeoSHELL   Start geoSHELL (Initial jump address).
$1703 Only128   Inform the user when a 128 command is being used on a 64 and exit.
$1706 OpenGS   Locate and open geoSHELL to gain access to it's VLIR records

$1709	NoMoreCmds	Exit from just about anything and return complete control back to geoSHELL.
$170c	ResetScreen	Display a fresh geoSHELL screen in the correct mode.
$170f	ClearScreen	Clear the entire screen.
$1712	ClearWindow	Clear the geoSHELL window.
$1715	EraseWindow	Clear only the geoSHELL window.
$1718	Asc_3_Byte	Convert an ascii number (0-255) to a single byte.
$171b	ReDoWindow	Redraw the geoSHELL window and any text that was already contained in it.
$171e	R_Icons	Restore the default geoSHELL icon.
$1721	OpenWrFile	Load a file's VLIR index into memory.
$1724	PushStuff	Save certain zero page locations to a built-in stack.
$1727	PopStuff	Restore certain variables after a call to PushStuff.
$172a	ResetFromPath	Reset a CMD device back from the path partition to the current partition the user is operating in.
$172d	CkForDisk	Check for a disk in any desired drive.
$1730	ParamName	Point r6 and a2 to the filename that the user entered.
$1733	DoRun	Load and run a file.
$1736	IncCmdPointer	Position the command pointer past your command's parameter.
$1739	CkPath	Check to see if a file is on the path partition.
$173c	DriveLetter	Display the drive letter of the active drive and turn the cursor on.
$173f	DispLetter	Display the drive letter of the currently active drive.
$1742	OpenPathPartition	Open the path partition.
$1745	Switch	Switch 40/80 column modes on a 128.
$1748	(reserved)	
$174b	ClrKeyData	Eliminate any detected keypresses.
$174e	YesKeypress	Wait for the user to press a key.$1751 NoKeypress Wait for the user to release the keyboard.
$1754	nogood	Usually identifies a failure in a routine.
$1757	yesgood	Usually identifies a successful routine.
$175a	DsplyLine	Display a line of text to the geoSHELL window beginning at the left side of the current line.
$175d	DsplyString	Display a string at the current cursor position.
$1760	CarrReturn	Perform a carriage return.
$1763	set_XY_pos	Load r1H and r11 with the pixel values of the current cursor position.
$1766	save_XY_pos	Save the current values of r1H and r11 into two variables.
$1769	ClearLine	Clear the current line in the geoSHELL window.
$176c	ClearBoth	Clear the current line on the screen and in the memory buffer.
$176f	ClearRemainder	Clear the window below the current cursor position.
$1772	StripRulers	Strip the GeoWrite rulers, escapes and any other unwanted characters from a GeoWrite page loaded into memory.
$1775	AdjTxtPointer	Used with DispText to advance r0 within a text buffer.
$1778	ReDisplayWindow	Clear each line in the geoSHELL window and redisplay the text contained in it from the internal buffers.
$177b	Message	Display an internal geoSHELL message.
$177e	OtherMessage	Display a message.
$1781	GetMessage	Point at one of geoSHELL's built-in messages.

$1784 Asc_Byte	Convert a two digit ascii number to a one byte value.
$1787 ByteWZ_Ascii	Convert a 16 bit number to an ascii representation of the number with no leading zeros.
$178a ByteNZ_Ascii	Convert a 16 bit number to a 5 digit ascii representation of the number with leading zeros.
$178d ConvertK	Convert a 16-bit number to an ascii string rounded off to the nearest kilobyte.
$1790 Asc_BCD	Convert a two digit ascii number to one byte in binary coded decimal (BCD) format.
$1793 CkESC_RULER Used for advancing a pointer past bytes used with ruler escapes in a GeoWrite document.$1796	DispText	Display ascii text to the geoSHELL window up to a maximum of the width of the window.
$1799 GetMess	Set a pointer to a message.
$179c execStartup Execute a string of commands located beginning at StUpBuffer.
$179f CkPathOrder Search a path partition for a file, but first check all available ramdisks, then the path partition, and then all remaining drives.
$17a2 FindParName Search the current drive for a desired file from a parameter the user entered.
$17a5 Wait	Wait for the user to press a key.
$17a8 ExitCommand Exit a transient command cleanly.
$17ab unsetPrompt Turn the blinking cursor off.
$17ae ColorScreen Color an area of the 80 column screen.
$17b1 SetBrdrColor	Change the color of the 80 column screen's border.
$17b4 SetThisDevice	A substitute for the GEOS routine 'SetDevice'.
$17b7 SetDrName	Display drive letter, drive type, and disk name.
$17ba FileNotAvail	Display a 'Filename Not Available!' message and return control to the user.
$17bd Searching	Display the 'Searching For Filename' message.
$17c0 MissgFilename	Display the message 'Bad Parameter!' without returning.
$17c3 NotFound	Display a 'FILENAME Not Found!" message.
$17c6 Dir	Display the directory of the current drive.
$17c9 CkTerminators	Check if the value is one of the geoSHELL text terminators.
$17cc CkTermNoSpace	Check if the value is one of the geoSHELL text terminators except for a space.
$17cf MissDisk	Display the message 'Missing Disk?!'.
$17d2 CkAllDrives Check all drives for a file.
$17d5 (reserved)
$17d8 CkKeyboard	Check for CONTROL, STOP and other keys and process accordingly.
$17db CkPresdrive Check the currently active drive for a file.
$17de CkThisDrive Check a drive for a file.
$17e1 PurgeAllTurbos	Purge the turbo code from all the drives that are being used in order to perform low-level DOS routines.
$17e4 CkOtherDrives	Search all drives for a file that have not yet been searched.
$17e7 DateDisplay Display the current date.$17ea	Byte_Time	Convert the time as contained in directory entries to ascii format. (this routine should not be used)
$17ed (reserved)
$17f0 FixColors	Restore the default colors to the 40 column screen.

$17f3	MskCtrlCodes	Remove control codes (any non-ASCII characters) from a text buffer.
$17f6	ConvToAscii	Convert PetASCII text to ASCII text.
$17f9	GetHeader	Load the header block of a file into fileHeader.
$17fc	LoadProblem	Display an error message and return control to the user.
$17ff	OpenGSIndex	Load geoSHELL's VLIR index block into fileHeader.
$1802	RdFnKeyDefs	Load the default function key definitions from the current drive.
$1805	Open_GC_Channel	Check the user port via a geoCable for the presence of a printer.
$1808	Send_GC_Byte	Send a byte to a printer via a geoCable connected to the user port.
$180b	SaveToREU	Copies geoSHELL and Input driver info to REU.
$180e	ClrTrnsName	Clear a transient command's name from memory.
$1811	CkDrvLetter	Tests if the accumulator is holding a value from 97 to 100 ('a', 'b', 'c', 'd') and converts it to a device number.
$1814	Status	Display information for each of the connected drives.
$1817	SendCmd	Send a string of bytes to the drive's command channel.
$181a	ReadChannel	Read a drive's error channel.
$181d	SendListen	Get a drive's attention and send it a command.
$1820	SendTalk	Get a drive's attention and send it a command to prepare it for talking.
$1823	FindRamLink	Find a RamLink or RamDrive if one is currently being used.
$1826	GetCMDType	Check if a device is a CMD device and if so, what type.
$1829	FindCMDType	Find the location of a CMD device.
$182c	OpenPartition	Open a partition on a CMD device.
$182f	SetPartParams	Set up some internal variables for the current partition.
$1832	InitCurPartition	Open the currently active partition after changing to a different one.
$1835	OpenCurPartition	Open the currently active partition after changing to a different one.
$1838	CkModeFlag	Test if a file is compatible with the current mode.
$183b	GetRealDriveNum	Get the 'real' device number of a drive.

-- NOTES –

APPENDIX B

MESSAGES AND CONSTANTS

As discussed in other parts of this manual, geoSHELL has some built-in text displaying routines. Of these routines, you have access to displaying messages to the user describing such things as errors or how your command might be progressing. It is always good to let the user know what is going on. The main routines for doing this is Message and OtherMessage. You will find yourself using OtherMessage the most, since Message is limited to only the built-in geoSHELL messages. OtherMessage needs to know where the message is that you wish to display. Point r0 at this null-terminated text string. One other thing it needs is a value loaded into the accumulator telling it how to display your text. You can indent from 1-7 spaces and also add trailing spaces and carriage returns. The routine Message also needs this information loaded into the accumulator. The following are the constants for this:

```
IN_CR = %10000000 ;initial carriage return.
                  ;otherwise begin at present
            ;cursor location.
TR_CR = %01000000 ;carriage return after text.
                  ;otherwise leave cursor one
                  ;space past text.

IN_ONE      = %00000001 ;one leading space
IN_TWO      = %00000010 ;two leading spaces...
IN_THREE    = %00000011 ;etc...
IN_FOUR     = %00000100
IN_FIVE     = %00000101
IN_SIX      = %00000110
IN_SEVEN    = %00000111

TR_ONE      = %00001000 ;one trailing space
TR_TWO      = %00010000 ;two trailing spaces...
TR_THREE    = %00011000 ;etc...
TR_FOUR     = %00100000
TR_FIVE     = %00101000
TR_SIX      = %00110000
TR_SEVEN    = %00111000

;A typical section of a routine might look like this:

      ...
      LoadW r0,#textToDisplay;point to the text.
      lda   #(IN_TWO|TR_CR)  ;insert 2 spaces, add a
                  ;carriage return.
      jsr   OtherMessage     ;go display it.
      ...

textToDisplay:
      .byte "This Text Will Be Displayed!",0
```

You can also load the accumulator with a zero if you wish tojust print the text right where the cursor is and leave the cursor at the space following the last character that gets displayed on the screen. You will find yourself using this routine quite often.
geoSHELL has some internal messages also that are accessible with the routine Message. Some of these messages are specific to certain routines and may not be of any benefit to you. However, there are also some that will be used quite frequently. You will notice in this listing the ones that look familiar and will realize the useful ones and the not so useful ones. In order to use these, you simply load the x register with the number of the message and load the accumulator just like you would for OtherMessage and then call Message.

Here is a complete list of the built-in messages:

```
      .byte "Can't Display File!",0      ;#1
      .byte "Searching For",0;#2
      .byte "Loading...",0   ;#3
      .byte "Bad Parameter!",0     ;#4
      .byte "Deleting",0     ;#5

      .byte "On Drive"
      .byte "A",0 ;#6

      .byte "Copying",0 ;#7

      .byte "Drive "
      .byte "A",0 ;#8

      .byte "To Drive "
      .byte "A",0 ;#9

      .byte "Not Found!",0   ;#10
      .byte "Write Protect On!",0  ;#11
      .byte "Missing Disk?!",0     ;#12
      .byte "Too Many Asterisks!",0      ;#13
      .byte "Command is Corrupted!!",0   ;#14
      .byte "Insufficient Room On Disk!",0     ;#15
      .byte "Filename May Not Exceed",0  ;#16
      .byte "8 Char's!",0    ;#17
      .byte "11 Char's!",0   ;#18

      .byte "File Is Locked!!!",0  ;#19
      .byte "Is This A Command?",0 ;#20
      .byte "128 mode only",0;#21
      .byte "Not Available",0;#22
      .byte 0     ;#23
      .byte "Cannot Run",0   ;#24
      .byte "No Matches Found",0   ;#25
      .byte "** Border Files **",0 ;#26
      .byte "** Non-GEOS Disk **",0      ;#27

      .byte "Date:   "
      .byte "01/"
```

```
.byte "02/"te     "91",0      ;#28

.byte "Time:   "
.byte "12:"
.byte "00 "
.byte "pm",0      ;#29

.byte "geoSHELL V2.2",0;#30
.byte "Copyright 1993 by Maurice Randall",0 ;#31
.byte 0     ;#32 unused
.byte 0     ;#33 unused
.byte "Wrong Format",0 ;#34
.byte "Problem Loading File!",0    ;#35

.byte " is for 40 columns!",0      ;#36
.byte " is for 128-40 columns!",0  ;#37
.byte " is for 128-80 columns!",0  ;#38
.byte " is for 128 mode!",0  ;#39
.byte " is for 64 mode!",0   ;#40
.byte " is not a GEOS file!",0     ;#41
.byte " 0=Not Geos",0  ;#42
.byte " 1=Basic   ",0  ;#43
.byte " 2=Assembly",0  ;#44
.byte " 3=Data    ",0  ;#45
.byte " 4=System  ",0  ;#46
.byte " 5=Desk Acc",0  ;#47
.byte " 6=Applic  ",0  ;#48
.byte " 7=App Data",0  ;#49
.byte " 8=Font    ",0  ;#50
.byte " 9=Printer ",0  ;#51
.byte "10=64Input ",0  ;#52
.byte "11=Disk Drv",0  ;#53
.byte "12=Sys Boot",0  ;#54
.byte "13=Temp    ",0  ;#55
.byte "14=AutoExec",0  ;#56
.byte "15=128Input",0  ;#57
.byte "16=???     ",0  ;#58
.byte "17=GW Doc  ",0  ;#59
.byte "   ???     ",0  ;#60

.byte 0     ;#61 reserved
.byte 0     ;#62 reserved.
.byte "21=TransCom",0  ;#63
.byte "  SubDir   ",0  ;#64 reserved.
.byte "  SPLAT!!! ",0  ;#65
.byte 0     ;#66 reserved.
.byte 0     ;#67 reserved.
.byte 0     ;#68 reserved.
.byte 0     ;#69 reserved.
.byte 0     ;#70 reserved.

.byte "    0  KBytes Free",0 ;#71
.byte "This Is A Startup Command!",0     ;#72
.byte "Partition Not Compatible!",0      ;#73
```

```
.byte "Unknown To Device!",0 ;#74
.byte "Unable To Rename File!",0   ;#75
.byte "Drive "
.byte "D Not Available!",0   ;#76

.byte "-- Drive Selected --",0     ;#77
.byte "Printer Unavailable!",0     ;#78
.byte "Startup File Too Large!",0  ;#79
.byte "Parent Application",0 ;#80
.byte "Double-Sided Disk!",0 ;#81
.byte "Bad Directory Header?!",0   ;#82
.byte "Can't Find Main Loop!",0    ;#83
.byte "Sub-Dir Command Failed!!!",0      ;#84
.byte "Exec File Too Large!",0     ;#85
.byte "-- Partition Selected --",0 ;#86
.byte "Printer:",0     ;#87
.byte "Input:",0  ;#88
.byte 0     ;#89
.byte 0     ;#90
.byte 0     ;#91
.byte 0     ;#92
.byte 0     ;#93
.byte 0     ;#94
.byte 0     ;#95
.byte 0     ;#96
.byte 0     ;#97
.byte 0     ;#98
.byte 0     ;#99
.byte 0     ;#100
.byte 0     ;#101
.byte 0     ;#102
.byte 0     ;#103
.byte 0     ;#104
.byte 0     ;#105
.byte 0     ;#106
.byte 0     ;#107
.byte 0     ;#108
.byte 0     ;#109
.byte 0     ;#110
.byte 0     ;#111
.byte 0     ;#112
.byte 0     ;#113

      ;(used for the time display)
.byte "Time:   "
.byte "12:"
.byte "00 :"
.byte "00 "
.byte "pm",0      ;#114

.byte "A:",160,0  ;#115
.byte "B:",160,0  ;#116
.byte "C:",160,0  ;#117
.byte "D:",160,0  ;#118
```

```
      .byte 0     ;#119 reserved
      .byte 0     ;#120 reserved   .byte 0     ;#121 reserved
      .byte 0     ;#122 reserved
      .byte 0     ;#123 reserved

      .byte "1541  ",0  ;#124
      .byte "1571  ",0  ;#125
      .byte "1581  ",0  ;#126
      .byte "RAM 41",0  ;#127
      .byte "RAM 71",0  ;#128
      .byte "RAM 81",0  ;#129
      .byte "SHDW41",0  ;#130
      .byte "SHDW71",0  ;#131
      .byte "SHDW81",0  ;#132
      .byte "RL1581",0  ;#133
      .byte "RL NTV",0  ;#134
      .byte "GW RAM",0  ;#135
      .byte "----  ",0  ;#136
      .byte "???   ",0  ;#137
      .byte "HD1581",0  ;#138
      .byte "HD NTV",0  ;#139

      .byte "HD1541",0  ;#140 unknown?
      .byte "HD1571",0  ;#141 unknown?

      .byte "FD1581",0  ;#142
      .byte "HD1581",0  ;#143
      .byte "RD1581",0  ;#144
      .byte "RL1581",0  ;#145

      .byte "FD NTV",0  ;#146
      .byte "HD NTV",0  ;#147
      .byte "RD NTV",0  ;#148
      .byte "RL NTV",0  ;#149
      .byte 0     ;#150 reserved

      .byte "Exit To Drive A Or B!",0    ;#151

      .byte "SEQUENTIAL",0   ;#152
      .byte "VLIR",0    ;#153
      .byte "???",0     ;#154

-- NOTES --
```

APPENDIX C

MODES AND THEIR VALUES

GEOS is a system that has been developed to be shared as closely as possible between the two Commodore computers, the 64 and 128. In most cases, when an application works with GEOS 64, it will work with GEOS 128 in 40 column mode. But, once in a while, an application will show up that can only be run on a specific machine or in a specific mode, such as a 128 in 80 column mode. There is a byte in a file's header block that will identify the type of computer that it is allowed or was designed to be run on. That byte can be found at fileHeader+96 when the file's header block is loaded in at fileHeader. There is a geoSHELL routine for checking if a file is compatible with the computer or the current mode it is in. That routine is CkModeFlag. Refer to the documentation on this routine for more info on how to use it.
For manual checking of this byte, here is a listing of the possible values and their meaning:

```
$00 = 40 column only - 128 or 64
$20 = 128 - 40 column only
$40 = All modes allowed
$60 = 128 only - 40 or 80 columns
$80 = 64 only
$c0 = 128 - 80 column only
```

If the current mode of the computer does not match or is not compatible with this byte, then geoSHELL will not execute a file that the user is trying to run. For the same token, if your command is being written specifically for a particular machine or mode, then be sure to include the correct byte in the header file of your command.
While your command is executing, if you need to check the type of computer or mode, then just test the variable 'screenmode' for this information. Here are the possible variations of this variable:

```
#%10000000 = 128 in 80 column mode
#%00000000 = 128 in 40 column mode
#%01000000 = 64
```

In addition to this, the variable 'videomode' will tell you if the RGB mode of the 80 column screen has been selected. Bit 7 will reflect this. This can be set even if the computer is in 40 column mode. That way, when the user switches to 80 column mode, it will be put into RGB (color) mode.
When manually placing text or graphics on the screen, it is not a good idea to use the lower portion of the screen below scanline 176 since geoSHELL allows a 16K video ram equipped 128 to operate in color mode. You will notice that none of the standard functions in geoSHELL do anything in this lower area. Even the mouse is prevented from entering this area. So, keep this in mind unless the machine is not in color mode. But that would prevent your command from being used by those who likeusing geoSHELL's color on the 80 column screen.

APPENDIX D

THE INCLUDED SOURCE CODE

Included with this package are a number of source code files. Within these you will find many useful examples that you might be able to use in your own source code when creating a geoSHELL command. Feel free to make use of these, or to get ideas on how to develop your own routines. You can also find many examples throughout this manual. Almost every routine description has at least one source code example to go with it. Just use GeoWrite to cut and paste any of this source into your own source code files.
When creating a geoSHELL command, you might always want to start out with the 'Sample' source code files and build on them. This group of files includes the following:

```
Sample     ;the main source file.
SampleHdr  ;the header file.
Sample.lnk ;the linker file.
```

Just make copies of these files and then rename them appropriately for the command you are creating. Then start adding additional code to create your command.
The included file MiscSource contains several sample routines. Each routine is documented in the file. Look through this file and see if there are any routines that might be useful to you.
Also included is the complete source code for the command 'color80'. This was the first graphic command that operated outside of the geoSHELL window. While looking through the source for color80, you'll notice that it would really have been easy to change the colors of the screen with just a parameter following the command. But I created the command this way to show everyone that geoSHELL doesn't leave you stuck working with text in a window. The bulk of color80 deals with the screen it creates, and not the actual changing of the colors.
color80 is a 128 only command but will still give the 64 users an idea of how to get in and out of the geoSHELL window. The biggest thing you must remember is to not jump through NoMoreCmds until you've restored the window. It is your job to do this. The NoMoreCmds routine could have been made to check for this condition except the routine is used quite a lot and it would be disturbing to see the window redrawn on every error that occurs. Perhaps a variable could have been used to tell the routine if the window is active or not. In any case, just be sure to rebuild the window if you do anything to remove it before your command exits. There are routines that will do this for you. If you don't trash the window, then you need not worry about it.
In the file MiscSource you'll also find a way to blank the 40 column screen for when you need to use the extra screen memory for processing. The same methods here can be used to restore the screen. You will see just how easy it is. When operating in 80 columns, you can use the 40 column foreground memory without blanking the screen. The routines included in MiscSource will check the screen mode and take appropriate action for you.
REQUIRED FILES

When you are assembling and linking your commands, there will be some files that you will need in order to have access to all that geoSHELL makes available to you. In your main source code, you would naturally want geosSym (included in GEOPROGRAMMER) in order to be able to use the labels it provides for having access to the GEOS kernal along with various constants. You'll also need geosMac in order to be able to use the macros that you will find in most of the examples that are in this package. Both of these files are included with GEOPROGRAMMER. In the sample source code file that you will use as your template, these have already been .included'd at the beginning of the source file. In the sample linker file there is a call to GSVariables.rel. This is the other file that you need. It is included in this package. Also included is the source code that assembles into this .rel file.
You can load up these source files into GeoWrite if you want to study the different equates and constants contained in them. By linking GSVariables.rel with your source code, you will have access to these. This includes all of the built-in routines in geoSHELL and all of the accessible variables. Plus some additional constants are included for geoSHELL that you may or may not have a use for. Linking GSVariables.rel with your source does not add any bytes at all to the code that you are creating. It is all constants and equates and no actual program code. But link it ahead of your file so that GEOLINKER will already be aware of it when it links your file.
The following files are used to create GSVariables.rel:

GSVariables
GSVars1
GSVars2
ShellEquates

If you ever need to recreate GSVariables.rel, just assemble GSVariables. It .includes the other files when it is assembled. It would not be a good idea to alter the contents of any of these files unless you are merely adding some of your own constants or equates. If you alter the existing ones, then your code might put data where it doesn't belong.
Most of the equates and constants contained in these files are also discussed throughout this manual.

geoSHELL ROUTINES INDEXED BY TOPIC

The first section of this index contains all of the available geoSHELL routines listed by topic. Some routines can be found under more than one topic heading if they should pertain to more than one subject. Section 2 of this manual contains a full description of each routine in alphabetical order. Refer to the general index for the exact page numbers where each routine may be found plus other locations throughout the manual where a particular routine might be mentioned.

DISK OR DRIVE RELATED

CkAllDrives Check all drives for a file.
CkDrvLetter Tests if the accumulator is holding a value from 97 to 100 ('a', 'b', 'c', 'd') and converts it to a device number.
CkForDisk Check for a disk in any desired drive.
CkModeFlag Test if a file is compatible with the current mode.
CkOtherDrives Search all drives for a file that have not yet been searched.
CkPath Check to see if a file is on the path partition.
CkPathOrder Search a path partition for a file, but first check all available ramdisks, then the path partition, and then all remaining drives.
CkPresdrive Check the currently active drive for a file.
CkThisDrive Check a drive for a file.
DoRun Load and run a file.
FindCMDType Find the location of a CMD device.
FindParName Search the current drive for a desired file from a parameter the user entered.
FindRamLink Find a RamLink or RamDrive if one is currently being used.
GetCMDType Check if a device is a CMD device and if so, what type.
GetHeader Load the header block of a file into fileHeader.
GetRealDriveNum Get the 'real' device number of a drive.
InitCurPartition Open the currently active partition after changing to a different one.
OpenCurPartition Open the currently active partition after changing to a different one.
OpenGS Locate and open geoSHELL to gain access to it's VLIR records
OpenGSIndex Load geoSHELL's VLIR index block into fileHeader.
OpenPartition Open a partition on a CMD device.
OpenPathPartitionOpen the path partition.
OpenWrFile Load a file's VLIR index into memory.
PurgeAllTurbos Purge the turbo code from all the drives that are being used in order to perform low-level DOS routines.RdFnKeyDefs Load the default function key definitions from the current drive.
ReadChannel Read a drive's error channel.
ResetFromPath Reset a CMD device back from the path partition to the current partition the user is operating in.
SendCmd Send a string of bytes to the drive's command channel.
SendListen Get a drive's attention and send it a command.
SendTalk Get a drive's attention and send it a command to prepare it for talking.
SetPartParams Set up some internal variables for the current partition.

SetThisDevice	A substitute for the GEOS routine 'SetDevice'.

KEYBOARD RELATED ROUTINES

CkKeyboard	Check for CONTROL, STOP and other keys and process accordingly.
ClrKeyData	Eliminate any detected keypresses.
NoKeypress	Wait for the user to release a key.
RdFnKeyDefs	Load the default function key definitions from the current drive.
YesKeypress	Wait for the user to press a key.
Wait	Wait for the user to press a key.

INPUT RELATED ROUTINES

FindParName	Search the current drive for a desired file from a parameter the user entered.
ParamName	Point r6 and a2 to the filename that the user entered.

PRINTER ROUTINES

Open_GC_Channel	Check the user port via a geoCable for the presence of a printer.
Send_GC_Byte	Send a byte to a printer via a geoCable connected to the user port.

SCREEN AND WINDOW ROUTINES

ClearScreen	Clear the entire screen.
ClearWindow	Clear the geoSHELL window.
ColorScreen	Color an area of the 80 column screen.
EraseWindow	Clear only the geoSHELL window.
FixColors	Restore the default colors to the 40 column screen.
ReDoWindow	Redraw the geoSHELL window and any text that was already contained in it.
ResetScreen	Display a fresh geoSHELL screen in the correct mode.R_Icons Restore the default geoSHELL icon.
SetBrdrColor	Change the color of the 80 column screen's border.
Switch	Switch 40/80 column modes on a 128.

TEXT DISPLAYING

CarrReturn	Perform a carriage return.
CkESC_RULER	Used for advancing a pointer past bytes used with ruler escapes in a GeoWrite document.
CkTerminators	Check if the value is one of the geoSHELL text terminators.
CkTermNoSpace	Check if the value is one of the geoSHELL text terminators except for a space.
ClearLine	Clear the current line in the geoSHELL window.
ClearRemainder	Clear the window below the current cursor position.
ConvToAscii	Convert PetASCII text to ASCII text.
DateDisplay	Display the current date.
Dir	Display the directory of the current drive.

DispLetter Display the drive letter of the currently active drive.
DispText Display ascii text to the geoSHELL window up to a maximum of the width of the window.
DriveLetter Display the drive letter of the active drive and turn the cursor on.
DsplyLine Display a line of text to the geoSHELL window beginning at the left side of the current line.
DsplyString Display a string at the current cursor position.
FileNotAvail Display a 'Filename Not Available!' message and return control to the user.
GetMess Set a pointer to a message.
GetMessage Point at one of geoSHELL's built-in messages.
LoadProblem Display an error message and return control to the user.
Message Display an internal geoSHELL message.
MissDisk Display the message 'Missing Disk?!'.
MissgFilename Display the message 'Bad Parameter!' without returning.
MskCtrlCodes Remove control codes (any non-ASCII characters) from a text buffer.
NotFound Display a 'FILENAME Not Found!" message.
Only128 Inform the user when a 128 command is being used on a 64 and exit.
OtherMessage Display a message.
ReDisplayWindow Clear each line in the geoSHELL window and redisplay the text contained in it from the internal buffers.
save_XY_pos Save the current values of r1H and r11 into two variables.
Searching Display the 'Searching For Filename' message.
SetDrName Display drive letter, drive type, and disk name.set_XY_pos
Load r1H and r11 with the pixel values of the current cursor position.
Status Display information for each of the connected drives.
StripRulers Strip the GeoWrite rulers, escapes and any other unwanted characters from a GeoWrite page loaded into memory.
unsetPrompt Turn the blinking cursor off.

TEXT AND NUMBER CONVERSION ROUTINES

Asc_3_Byte Convert an ascii number (0-255) to a single byte.
Asc_BCD Convert a two digit ascii number to one byte in binary coded decimal (BCD) format.
Asc_Byte Convert a two digit ascii number to a one byte value.
ByteNZ_Ascii Convert a 16 bit number to a 5 digit ascii representation of the number with leading zeros converted to spaces.
ByteWZ_Ascii Convert a 16 bit number to a 5 digit ascii representation of the number with leading zeros.
CkDrvLetter Tests if the accumulator is holding a value from 97 to 100 ('a', 'b', 'c', 'd') and converts it to a device number.
CkESC_RULER Used for advancing a pointer past bytes used with ruler escapes in a GeoWrite document.
ConvertK Convert a 16-bit number to an ascii string representing kilobytes.
ConvToAscii Convert PetASCII text to ASCII text.
MskCtrlCodes Remove control codes (any non-ASCII characters) from a text buffer.

StripRulers	Strip the GeoWrite rulers, escapes and any other unwanted characters from a GeoWrite page loaded into memory.

ERROR RELATED ROUTINES

FileNotAvail	Display a 'Filename Not Available!' message and return control to the user.
LoadProblem	Display an error message and return control to the user.
MissDisk	Display the message 'Missing Disk?!'.
MissgFilename	Display the message 'Bad Parameter!' without returning.
nogood	Usually identifies a failure in a routine.
NoMoreCmds	Exit from just about anything and return complete control back to geoSHELL.
NotFound	Display a 'FILENAME Not Found!" message.
Only128	Inform the user when a 128 command is being used on a 64 and exit.
yesgood	Usually identifies a successful routine.

COMMAND RELATED

CkTerminators	Check if the value is one of the geoSHELL text terminators.
CkTermNoSpace	Check if the value is one of the geoSHELL text terminators except for a space.
ClrTrnsName	Clear a transient command's name from memory.
execStartup	Execute a string of commands located beginning at StUpBuffer.
ExitCommand	Exit a transient command cleanly.
IncCmdPointer	Position the command pointer past your command's parameter.

MEMORY RELATED ROUTINES

AdjTxtPointer	Used with DispText to advance r0 within a text buffer.
ClearBoth	Clear the current line on the screen and in the memory buffer.
ClrTrnsName	Clear a transient command's name from memory.
InitgeoSHELL	Start geoSHELL (Initial jump address).
PopStuff	Restore certain variables after a call to PushStuff.
PushStuff	Save certain zero page locations to a built-in stack.
SaveToREU	Copies geoSHELL and Input driver info to REU.

INDEX TO ROUTINES

INDEX TO VARIABLES AND OTHER MEMORY LOCATIONS
(any variables not listed here can be found in APPENDIX A)

MISCELLANEOUS INDEX

MISCELLANEOUS SOURCE CODE

This is a table of contents of the sample source code contained in this file. Feel free to make use of the code contained here for use in your own commands. This helps save time, instead of trying to invent something that has already been done for you. Just use GeoWrite to cut and paste any of these examples into your own source code.

```
;here is the code that will call the routines to blank the 40 column
;screen and restore them after your command finishes it's task.
;This method puts a message on the bottom of the 40 column screen and
;also deals with proper handling if the routine is run on an 80 column
128.
;You will be able to use memory from $a000 to $bdff.
;If you need the additional 320 bytes that the message on the bottom
uses,
;then use BlnkNMessage in place of BlnkWMessage. BlnkNMessage can be
found
;on the following page. Be sure to call RstrWindow before exiting your
command.
;Also, test your command for anything that might happen while the screen
is
;blanked. You must be able to call RstrWindow to get the window back on
the
;screen for the user.

      ...        ;any code ahead of this...
      jsr   BlnkWMessage     ;blank the screen with a message
                 ;at the bottom.
      jsr   DoYourThing ;this calls your own code.
      jsr   RstrWindow  ;put the geoSHELL window back.
      ...        ;any more additional code....

RstrWindow:
      clc
      jsr   StartMouseMode   ;turn the mouse back on.
      bit   screenmode ;is this 80 column mode?
      bmi   50$   ;branch if so. (80 columns didn't blank)
      jsr   ClearScreen;clear the screen.
      jsr   ReDoWindow ;redraw the window and it's contents.
                  ;this is all taking place while the
                  ;screen is still blanked, until we
      jsr   FixColors  ;put the color back on the screen.
 50$
      jmp   R_Icons    ;do this because the icon gets killed.

;this routine blanks the 40 column screen and displays a message at the
;bottom. If in 80 column mode, it will put the same message in the
;geoSHELL window since the screen does not have to be blanked. The
message
;will also be placed in the 40 column window in addition to the bottom of
;the screen.

BlnkWMessage:
      jsr   ClearMouseMode   ;shut the mouse off for now.
      LoadW r0,#prgrsText    ;point at our message.
      lda   #(IN_ONE|TR_CR)
      jsr   OtherMessage     ;put it on 40 or 80 columns.
      bit   screenmode ;now check for 80 columns.
      bmi   20$   ;branch if so and skip the rest.
      LoadW r0,#(24*40);blank only the upper 24 card rows.
      LoadW r1,#COLOR_MATRIX
      lda   screencolors     ;get the current screencolors.
      and   #%00001111 ;put the background color into
      sta   r2L
      asl   a
      asl   a
      asl   a
      asl   a
      ora   r2L   ;the foreground also.
      sta   r2L
      jsr   FillRam    ;and color the screen.
      PushB windowBottom     ;make sure that PutString can
      LoadB windowBottom,#199 ;get down to the bottom of the screen.
      LoadW r0,#prgrs40Text  ;point r0 for PutString.
      jsr   PutString  ;use GEOS to print this string.
      PopB  windowBottom     ;restore this.
 20$
      rts        ;and return.

prgrs40Text:
      .byte GOTOXY
      .word 24    ;start printing at the 24th pixel.
                  ;adjust this for the length of
                  ;your message.
      .byte 198

;put your message here, but leave a leading and trailing space in
```

```
;the message so that the background pattern doesn't brush up against
;the message, for a better appearance.

prgrsText:
      .byte " Put Your Message Here... ",0

;this routine blanks the 40 column screen and allows you to use the
;entire foreground screen memory area since this routine does not
;put a message at the bottom of the screen.
;80 column mode is still dealt with properly here. The message in this
;routine will only appear in the window on both the 40 and 80 column
screens
;and not at the bottom of the screen like it does with BlnkWMessage.

BlnkNMessage:
      jsr   ClearMouseMode   ;shut the mouse off for now.
      LoadW r0,#prgrsText    ;point at our message.
      lda   #(IN_ONE|TR_CR)
      jsr   OtherMessage     ;put it on 40 or 80 columns.
      bit   screenmode ;now check for 80 columns.
      bmi   20$   ;branch if so and skip the rest.
      LoadW r0,#(25*40);blank the entire screen.
      LoadW r1,#COLOR_MATRIX
      lda   screencolors     ;get the current screencolors.
      and   #%00001111 ;put the background color into
      sta   r2L
      asl   a
      asl   a
      asl   a
      asl   a
      ora   r2L   ;the foreground also.
      sta   r2L
      jsr   FillRam    ;and color the screen.
 20$
      rts        ;and return.

prgrsText:
      .byte "Put Your Message Here...",0

;These routines will allow you to send data to a printer whether it
;be connected through the serial port or the user port with a geoCable.
;Just let InitPrinter and ByteToPrinter know which device you want data
;sent to. This example uses a variable printDest to identify that. If
;bit 7 is set, data goes to serial, bit 6 set goes to geoCable. If both
;bits are set, data goes to both. The following example will load
printDest
;from a parameter the user supplies and proceed to do it's thing.
;One thing to keep in mind with using the printer: If you are sending
text
;to a printer and also displaying the same text on the screen using
;geoSHELL routines, the user will get double copies of the data on the
;printer if he is using the @p or @g commands. Normally, your use of the
;printer won't involve sending the same text to the screen anyway.
```

```
     ...
     ...        ;code leading up to this.
     ...
     lda   ParamTable+0
     cmp   #'p'
     beq   10$
     cmp   #'g'
     beq   20$
     jmp   MissgFilename    ;display a bad parameter.
 10$
     lda   #%10000000
     .byte $2c
 20$
     lda   #%01000000
     sta   printDest
     ...
     ...        ;whatever else code you might have.
     ...
     jsr   InitPrinter;make sure the printer is there.
     bit   goodflag   ;if the printer is not available
                ;you can deal with it however you wish.
                ;in this case we will just exit.
     bmi   30$   ;branch if printer is ready.
     jmp   NoMoreCmds ;you might display an error message
                ;before this exit if needed.
 30$
     ...
     ...
     jsr   InitForIO
     ...        ;whatever routines you need to load
     ...        ;the accumulator will go here.
     ...        ;usually from a buffer.
     ...
     jsr   ByteToPrinter    ;send the byte in the accumulator
                ;to the printer.
     ...
     ...        ;now you either go back and do
     ...        ;some more or just end your command.
     jsr   Unlsn ;in case we're using the serial printer.
     jsr   DoneWithIO
     ...
;this routine checks to make sure that the printer/s is/are connected and
;ready to accept data. printDest identifies which port/s to find the
;printer/s at. After this routine is called, if a serial printer is being
;used, then it will be left in 'listen' mode on the serial bus. Before
any
;disk access can occur, you must call the kernal routine 'Unlsn'. Then
;before further access to the printer you will have to call InitPrinter
again.

InitPrinter:
     bit   printDest  ;where's the printer?
     bmi   10$   ;branch if serial.
     bvs   50$   ;branch if geoCable.
```

```
      bpl   20$   ;branch if neither.
 10$
      lda   curPrinter ;printer device # set by 'pconf'.
      jsr   SetDevice
      jsr   InitForIO  ;pop out of GEOS.
      LoadB STATUS,#0  ;clear STATUS.
      lda   curPrinter
      jsr   Listen     ;tell the printer to listen.
      lda   curSecond  ;secondary address set by 'pconf'.
      ora   #$60
      jsr   Second     ;send a command to the printer.
      lda   STATUS     ;did the printer respond?
      beq   30$   ;branch if so.
      jsr   Unlsn ;tell the printer to stop listening.
                  ;(what printer? there wasn't any)
      jsr   DoneWithIO ;pop back into GEOS.
 20$
      jmp   nogood
 30$
      jsr   DoneWithIO ;pop back into GEOS.
                  ;serial printer is still in
                  ;listen mode.
      bit   printDest  ;are we also doing the geoCable?
      bvc   80$   ;branch if not.
 50$
      jsr   InitForIO  ;pop out of GEOS.
      jsr   Open_GC_Channel  ;check for a printer on the geoCable.
      txa         ;save it for a second.
      pha
      beq   60$   ;branch if printer responded.
      jsr   Unlsn ;otherwise unlisten the serial bus
                  ;in case the serial printer is also set.
 60$
      jsr   DoneWithIO ;pop back into GEOS.
      pla         ;well?
      bne   20$   ;branch if no printer.
 80$
      jmp   yesgood

printDest:
      .block     1
;this routine sends a byte to the printer/s.
;load the accumulator with the byte to send.
;printDest identifies which port/s to find the printer/s at.
;InitForIO must be called before this routine.
;InitPrinter must be called before this routine.
;call the kernal routine Unlsn after sending the last byte if a serial
;printer is being used.

ByteToPrinter:
      sta   byteToPrint
      bit   printDest  ;where's the printer?
      bmi   20$   ;branch if serial.
      bvs   50$   ;branch if geoCable.
 10$
      rts
 20$
      jsr   Ciout
      bit   printDest
      bvc   10$
      lda   byteToPrint
 50$
      jmp   Send_GC_Byte

byteToPrint:
      .block     1
;this example will clear the 80 column color mode screen and
;color some icons that you will place on the screen.
;the current color bytes that are being altered will be saved and
;then restored upon exit.
;Modify this routine and use it as desired.

      bit   screenmode ;is this a 128?
      bmi   20$   ;branch if so.
 10$
      LoadW r0,#colrOnlyText ;tell the user this command is
      lda   #(IN_TWO|TR_CR)  ;for 128 color mode only.
      jsr   OtherMessage
      jmp   NoMoreCmds ;and exit.
 20$
      bit   videomode  ;are we in color mode?
      bpl   10$   ;branch if not.
      MoveB backcolor,sbackcolor ;save the default background color.
      MoveB textcolor,stextcolor ;save the default foreground color.
      MoveB brdrcolor,sbrdrcolor ;save the default border color.
      MoveB clr80pattern,sclr80pattern ;save the default pattern.
      LoadB backcolor,#WHITE80 ;set background to white.
      LoadB textcolor,#BLACK80 ;set the foreground color to black.
      LoadB clr80pattern,#0  ;set background pattern #0
      jsr   ClearScreen;clear the screen.
      LoadB brdrcolor,#WHITE80 ;set the border color
      jsr   SetBrdrColor     ;to white.
      ...         ;through this section, use BitmapUp
      ...         ;to put some icons on the screen
      LoadW r0,#icnClrScrap  ;call ColorScreen to color the icons.
      LoadW r1,#icnClrCombos
      jmp   ColorScreen
      ...         ;now you can point GEOS at your own
      ...         ;icon table and rts to mainloop
      rts

;one of the icons could call a routine to exit your command such as
follows:
CloseCommand:
      MoveB sbackcolor,backcolor ;restore the default background color.
      MoveB stextcolor,textcolor ;restore the default foreground color.
      MoveB sbrdrcolor,brdrcolor ;restore the default border color.
      MoveB sclr80pattern,clr80pattern ;restore the default pattern.
      jsr   ClearScreen ;clear the screen.
      jsr   ReDoWindow  ;put geoSHELL just like it was.
      jmp   ExitCommand ;and exit.

colrOnlyText:
      .byte "Command Only Works In 128 Color!",0
sbackcolor:
      .block 1
stextcolor:
      .block 1
sbrdrcolor:
      .block 1
sclr80pattern:
      .block 1;this Color Scrap will color 8 icons that are placed in two
rows of 4
;similar to the way they are arranged on the Desktop. This only colors
the
;areas of the screen where the icons should be. You use BitmapUp to
;put the icons in place.
;To use this Color Scrap, point r0 at icnClrScrap and r1 at IcnClrCombos
;and then call ColorScreen.
;To understand this Color Scrap, each icon is 6 cards wide and 3 cards
tall.
;The upper left corner of icon #1 is at row 10, column 10. There are 4
;icons on this row. There is 6 cards separating each icon. The next row
of
;four icons is directly below the first row of icons and is at row 16.
;Each icon is given a different color.
;The third row of bytes in this example is repeated 3 times. It begins
;at column zero and skips over to column 10 without doing any coloring.
;Then, 6 cards are colored with color #0 from the color table. 6 more
cards
;are skipped and 6 are colored with color #1. This goes on until the
;rightmost icon gets it's coloring. Then 28 bytes are skipped to put the
;pointer at the start of the next row. Now this will repeat to color
;the second card row of each of the top icons. A similar thing happens
;for the second row of icons later on as you can see.
;Modify this for your own use.

icnClrScrap:
      .byte 10,0  ;row 10, column 0
      .byte 3,8   ;repeat 8 commands 3 times.
      .byte 10,255,6,0,6,255,6,1,6,255,6,2,6,255,6,3,28,255
      .byte 255   ;a new position command.
      .byte 16,0  ;row 16, column 0
      .byte 3,8   ;repeat 8 commands 3 times.
      .byte 10,255,6,4,6,255,6,5,6,255,6,6,6,255,6,7,28,255
      .byte       ;end of table.

icnClrCombos:
      .byte (DK80BLUE<<4)|LT80CYAN ;dk blue foregrnd,light cyan backgrnd.
      .byte (BLACK80<<4)|LT80YELLOW
      .byte (DK80PURPLE<<4)|WHITE80
      .byte (DK80RED<<4)|LT80RED
      .byte (LT80GREY<<4)|BLACK80
      .byte (DK80BLUE<<4)|LT80GREY
      .byte (DK80GREEN<<4)|BLACK80
      .byte (WHITE80<<4)|DK80RED
;this example will draw a framed rectangle and then color the
;frame of the rectangle and the space within the rectangle.
;This particular rectangle will be drawn exactly where the
;geoSHELL window resides. It will be given a black frame, while the space
;inside the rectangle will have a white background with a dark blue
;foreground. This example only works in 80 column color mode.
;Modify this for your own use.

DoColorFrame:
      jsr   ClearScreen ;clear the screen with default
                  ;foreground and background colors
                  ;and background pattern.
      lda   #0    ;use pattern number #0.
      jsr   SetPattern
      jsr   i_Rectangle ;draw the rectangle.
      .byte 32,167
      .word 16,607
      jsr   i_FrameRectangle ;and put a frame around it.
      .byte 32,167
      .word 16,607
      .byte 255
      LoadW r0,#frameScrap
      LoadW r1,#frameColors
      jmp   ColorScreen

frameScrap:
      .byte 4,2   ;row 4, column 2
      .byte 1,1   ;repeat 1 command set once.
      .byte 74,0  ;color 74 cards, the top of the rectangle.
      .byte 15,5  ;repeate 5 command sets 15 times.
      .byte 2,255,1,0,72,1,1,0,4,255
                  ;color the next 15 rows of the
                  ;rectangle.
      .byte 1,2   ;repeat 2 commands sets once.
      .byte 2,255,74,0 ;color the bottom of the rectangle.
      .byte 0

;use two different color combinations with this rectangle.
frameColors:
      .byte (BLACK80<<4)|WHITE80
      .byte (DK80BLUE<<4)|WHITE80

;here is a routine that would give color to geoSHELL when in 40 column
mode.
;Just change the equates here to whatever you like.
;geoSHELL has a built-in routine for doing this same thing in 80
;column mode. The routine is called ColorScreen. It is much simpler
;to create a Color Scrap for that routine to use. For the 40 column
;screen, though, geoSHELL has no coloring routines except for FixColors
;which restores the original screen colors (one foreground and one
;background color).

BACKBACK    = LTBLUE    ;background color in background area.
BACKFORE    = BLUE      ;foreground color in background area.
SHELLCLR    = RED ;frame color of geoSHELL.
PADCLR      = WHITE     ;pad color of geoSHELL.
TEXTCLR     = BLACK     ;text color in pad area.

Color40:
      LoadW r0,#COLOR_MATRIX ;start of 40 column color area.
      ldy   #0
      lda   #((BACKFORE<<4)|BACKBACK)
 10$
      sta   (r0),y      ;color the upper four rows of screen.
      iny
      cpy   #160
      bne   10$
      AddVW #160,r0     ;increment r0 to point at the top row
            ;of the shell frame.
      ldx   #3    ;do this segment three times.
 15$
      ldy   #0
      lda   #((BACKFORE<<4)|BACKBACK)
      sta   (r0),y      ;color empty space at left of frame.
      iny
      lda   #((SHELLCLR<<4)|PADCLR)
 20$
      sta   (r0),y      ;color the top of the frame.
      iny
      cpy   #38
      bne   20$
      lda   #((BACKFORE<<4)|BACKBACK)
 30$
      sta   (r0),y      ;color the two empty spaces at the right
            ;of the frame.
      iny
      sta   (r0),y
      jsr   Add40R0
      dex
      bne   15$
      ldx   #12   ;now do 14 rows of the screen in the
            ;pad area.
 40$
      ldy   #0
      lda   #((BACKFORE<<4)|BACKBACK)
      sta   (r0),y      ;the empty space at the left.
      iny
      lda   #((SHELLCLR<<4)|PADCLR)
      sta   (r0),y      ;one card on the left border of frame.  iny
      lda   #((TEXTCLR<<4)|PADCLR)
 50$
      sta   (r0),y      ;color 37 cards in the pad.
      iny
      cpy   #37
      bne   50$
      iny
      lda   #((SHELLCLR<<4)|PADCLR)
      sta   (r0),y     ;color the border at the right side.
      iny
      lda   #((BACKFORE<<4)|BACKBACK)
      sta   (r0),y     ;color the two empty spaces at the right.
      iny
      sta   (r0),y
      jsr   Add40R0    ;point at the next row down.
      dex        ;count down from 12.
      bne   40$   ;branch if more to do.
      ldx   #2
 55$
      ldy   #0    ;now we do the bottom of the frame.
      lda   #((BACKFORE<<4)|BACKBACK)
      sta   (r0),y
      iny
      lda   #((SHELLCLR<<4)|PADCLR)
 60$
      sta   (r0),y
      iny
      cpy   #38
      bne   60$
      lda   #((BACKFORE<<4)|BACKBACK)
      sta   (r0),y
      iny
      sta   (r0),y
      jsr   Add40R0
      dex
      bne   55$
      ldy   #0
      lda   #((BACKFORE<<4)|BACKBACK)
 70$
      sta   (r0),y
      iny
      cpy   #160
      bne   70$
      rts

Add40R0:
      AddVW #40,r0
      rts
;when you need to look ahead in the command buffer for a text string or
;another command, this example will show you how to do it.
;The 'getkey' command has to look forward through an exec or startup file
;for a left-curly brace followed by a character that represents the key
;that the user has pressed. This example does it in a similar manner.
;a4 always points to the next command or your parameter if there is one.
;From that point until the end of the buffer is all text until a null-
byte
;is encountered, signifying the end of the command buffer, or the
;startup buffer. It makes no difference whether your command is coming
;from a startup file in the startup buffer or the keyboard in the
```

```
;command buffer. The only difference is that the startup buffer can hold
;more commands.
;In this example, we are simply looking for a string
;called 'skip'. If goodflag is set upon return then r0 is pointing at
;skip. No ifs, ands, or buts.
;a4 does not get altered, however we can skip ahead if we'd like
;by copying r0 to a4. This particular code loops around perhaps a little
;too much, but it works perfectly and will return with r0 either pointing
;at skip for sure or definitely not pointing at it.

FindSkip:
      MoveW a4,r0 ;point r0 at the start of what's left
                  ;in the buffer.
      LoadW r1,#skipText     ;point r1 at the text to compare to.
      bra   15$   ;go check the first byte.
                  ;(the first byte doesn't have to be
                  ;a space)
 10$
      ldy   #0    ;let's look for the next terminator.
                  ;because there has to be a space before
                  ;and after 'skip'.
      lda   (r0),y     ;get a byte.
      beq   90$   ;branch if end of buffer.
 13$
      jsr   CkTerminators    ;is this a terminator?
      jsr   IncR0 ;increment r0 before checking goodflag.
      bit   goodflag   ;so, was it?
      bmi   15$   ;branch if so.
      bpl   10$   ;or branch if not and keep looking.
 15$
      lda   (r0),y      ;get the next byte.
      beq   90$   ;branch if end of buffer.
      cmp   skipText+0 ;does this match the 's'in 'skip'?
      bne   13$   ;branch in case it's a terminator.
      ldx   #r0   ;let GEOS kernal check
      ldy   #r1   ;the two strings.
      lda   #4    ;four characters.
      jsr   CmpFString ;are they equal?
      beq   60$   ;branch if so.
      jsr   IncR0
      bra   10$   ;branch always.
;at this point, we have one more check to make. We make sure that the
;user is using the correct syntax by putting a space after 'skip'.
 60$
      ldy   #4    ;point at the byte past 'skip'.
      lda   (r0),y      ;and fetch that byte.  jsr   CkTerminators
      ;maybe this string here is 'skippy'
      bit   goodflag   ;or something else.
      bmi   80$
      jsr   IncR0 ;point at the next byte.
      bra   10$   ;branch if not 'skip' with a space
                  ;and keep looking.
 80$
      rts         ;goodflag is already set.
 90$
      jmp   nogood     ;no skip in this buffer.
skipText:
      .byte "skip"
IncR0:
      inc   r0L   ;increment r0 to the next byte.
      bne   10$   ;branch if it didn't roll over.
      inc   r0H   ;otherwise increment the high byte.
 10$
      rts
```

```
;***************************************************
;
;           GSVariables
;
;     Used with transient commands in geoSHELL
;
;     Copyright 1994 by Maurice Randall
;     Charlotte, Mich. 48813
;
;     For Use Only By Registered Owners Of The
;     geoSHELL Programmer's Development Package
;
;***************************************************


;this file is merely assembled and the resulting .rel file is added
;to the .lnk file instead of .include(ing) it in the transient
;command source code. This along with geosSym would cause a symbol
;table overflow.


.include   GSVars1
.include   GSVars2
.include   ShellEquates

DummyLabel:      ;without this, GeoAssembler will
                 ;mess up.
```

```
;*****************************************************
;
;          GSVars1
;
;     Used with transient commands in geoSHELL
;
;     Copyright 1994 by Maurice Randall
;     Charlotte, Mich. 48813
;
;     For Use Only By Registered Owners Of The
;     geoSHELL Programmer's Development Package
;
;*****************************************************

;.included in GSVariables

StOfRamsect= $0400     ;used as a reference point for
                 ;initializing this area.

screenmode  = $0400
PRESDRIVE   = $0401
goodflag    = $0402
DRVFOUNDON  = $0403
STUPMODE    = $0404
FKEYMODE    = $0405
NUM_IN_NAME= $0406
ParamTable  = $0407     ;.block 44
           ;a couple extra null bytes added here
                 ;to protect any following data.
comchars    = $0433
cur_Y_pos   = $0434
cur_X_pos   = $0435
cur_line    = $0437
cur_char    = $0438

tempFileName      = $0439     ;.block 17
                 ;this may be used by routines for storing
                 ;filenames temporarily.
curTransName      = $044a     ;.block 17
           ;transient command that is currently
                 ;in memory.
scrnwidth   = $045b
domessages  = $045c     ;a zero here will prevent messages
           ;from being displayed. bit 7 set will
                 ;allow messages.
cur_keyData= $045d
CMD_POINTER= $045e
tempDrive   = $045f     ;used for status and directory displays.
typeset     = $0460
cmpwidth    = $0461
paramsize   = $0462     ;holds the size of the parameter that
                 ;is loaded in ParamTable.
pathload    = $0463     ;this is set to 128 when a transient
                 ;command is loaded from the specified
                 ;path.

Asc_String = $0464    ;.block 6
endecho    = $046a
sourceType = $046b
destType   = $046c
sourceDrive= $046d
destDrive  = $046e
sourcePartition  = $046f
destPartition    = $0470
StOfDefs   = $0471

videomode  = $0471    ;used only for 80 columns.
                 ;0 = monochrome
                 ;1 = color
distype    = $0472
fastdirectory    = $0473
defStName  = $0474    ;.block 17
           ;"Startup",0,0,0,0,0,0,0,0,0,0
coltable   = $0485
backcolor  = $0485
shellcolor = $0486
textcolor  = $0487
padcolor   = $0488
brdrcolor  = $0489
back40pattern    = $048a    ;default background pattern for 40 column.
back80pattern    = $048b    ;default background pattern for 80 column.
clr80pattern     = $048c

;$048d-$048f is unused.

curPrinter = $0490
curSecond  = $0491
crtextstring     = $0492    ;.block 3
curCharSet = $0495    ;  0 = Ascii
                 ;128 = PetAscii

;$0496-$049f is reserved for future use and is also saved
;when the custom command is issued. It is loaded whenever geoSHELL
;loads in.

;EnOfDefs:

;$04a0-$04c3 is reserved for future use.

RAMLINK    = $04c4    ;the real device number for RamLink
                 ;or RamDrive is set here after a jsr
                 ;to FindRamlink. This is normally only
                 ;needed if there is a RAM1581 used.

cmdtype    = $04c5    ;.block 3
           ;cmdtype gets loaded with a string such
           ;as "RL" or "HD" after accessing the
           ;routine GetCMDType.
```

```
            ;cmdtype+2 gets loaded with a byte
            ;from 0-4 representing one of the
            ;following:
            ;0=non-CMD type device
                  ;1=FD Series drive
                  ;2=HD Series hard drive
                  ;3=RD RamDrive
                  ;4=RL RamLink


KeyBuffStart     = $04c8

Key1Buff   = $04c8    ;.block 81
Key2Buff   = $0519    ;.block 81
Key3Buff   = $056a    ;.block 81
Key4Buff   = $05bb    ;.block 81
Key5Buff   = $060c    ;.block 81
Key6Buff   = $065d    ;.block 81
Key7Buff   = $06ae    ;.block 81
Key8Buff   = $06ff    ;.block 81

KeyBuffEnd  = $0750
```

```
;****************************************************
;
;           GSVars2
;
;     Used with transient commands in geoSHELL
;
;     Copyright 1994 by Maurice Randall
;     Charlotte, Mich. 48813
;
;     For Use Only By Registered Owners Of The
;     geoSHELL Programmer's Development Package
;
;****************************************************

;.included in GSVariables

restartbyte= $0750
CmdNumber  = $0751
zeroflag   = $0752
hotkeycommand    = $0753    ;.block 2
top_rectangle    = $0755
bot_rectangle    = $0756
left_rectangle   = $0757    ;.block 2
right_rectangle  = $0759    ;.block 2
temp_line  = $075b
temp_check = $075c    ;used for temporary storage of
                 ;a,x or y registers.
ck_year    = $075d
ck_month   = $075e
ck_day     = $075f
ck_hour    = $0760
ck_minute  = $0761
CMD_COUNT  = $0762
lastblock  = $0763    ;.block 2
filesiz    = $0765    ;.block 2
lstbyte    = $0767    ;.block 2
headlocation     = $0769    ;.block 6
FIRSTDRIVE = $076f
DRIVECHECK = $0770
foundtrans = $0771
inserty    = $0772
stparsize  = $0773
scancheck  = $0774
check_key  = $0775
nosearchmessage  = $0776
saver0     = $0777    ;.block 2
numfillbytes     = $0779
numfillcopies    = $077a
numfillstrings   = $077b
stringlocation   = $077c    ;.block 2
dublclickspot    = $077e
drivesChecked    = $077f    ;.block 4
PRNTMODE   = $0783    ;bit 7 set if characters are to go to
                 ;the printer in addition to the screen.
```

```
strtoprint = $0784     ;.block 2
           ;temporarily save r0 here until ready
                 ;to send the string that it points at
                 ;to the printer.
savex = $0786
line_count = $0787
cursorStatus     = $0788     ;high bit set means cursor is inverted.
           ;cleared means cursor is normal.
cursorOn    = $0789     ;high bit set means cursor is active.
           ;clear means cursor is inactive.
st_keyData = $078a
reg24 = $078b
gs_stack    = $078c     ;.block 256
stackpointer     = $088c
stupsequence     = $088d
ActHWide    = $088e
ActLeft     = $088f
svkeyVector = $0890     ;.block 2
Dir_Entry   = $0892     ;.block 2
Dir_Type    = $0894
StripZFlag  = $0895
ParentName  = $0896     ;.block 17
PermName    = $08a7     ;.block 17
DiskName    = $08b8     ;.block 17
filetoload  = $08c9     ;.block 17
WILDCARD    = $08da
DRV_CKD     = $08db
FILECOUNTER = $08dc
NxtBlock    = $08dd     ;.block 2
X_PARAM     = $08df     ;.block 2
Y_PARAM     = $08e1
MoveXTable  = $08e2     ;.block 2
MoveYTable  = $08e4     ;.block 2
NOSPACE     = $08e6
stackReturn = $08e7     ;.block 2
stripStack  = $08e9
NMCreturn   = $08ea     ;.block 2
DISPBUFFER  = $08ec     ;.block 84
WrModeFlag  = $0940
test1string = $0941     ;.block 17
test2string = $0952     ;.block 17
clrlineflag = $0963
clearcount  = $0964
messflag    = $0965
msgchars    = $0966
messlocation     = $0967     ;.block 2
horizcount  = $0969
kboardoff   = $096a
tempr0      = $096b     ;.block 2
gstructype  = $096d
gfiletype   = $096e
cfiletype   = $096f
texttrsc    = $0970     ;.block 2
nozero      = $0972
```

```
gscolcount  = $0973
headcolor   = $0974
colbyte     = $0975
staddress   = $0976     ;.block 2
clickspot   = $0978
tempy = $0979     ;used for temp. storage of y register.
TrRtnLAddr  = $097a     ;.block 8
TrRtnHAddr  = $0982     ;.block 8
TrRtnNum    = $098a
tcomnum     = $098b
pathcheck   = $098c
REALDRIVE   = $098d
cmdlength   = $098e
curparnum   = $098f
curpartype  = $0990
curdrvtype  = $0991
errorck     = $0992
partdir     = $0993
syscount    = $0994
cmdtosend   = $0995
lastck      = $0996
runorprint  = $0997

textpointer= $0998
interpret   = $0999
endoftext   = $099a     ;.block 2
            ;this contains the byte past the last
                 ;byte read in.

txttype     = $099c     ;0=SEQfile  (CBM or GEOS)
                 ;1=SCRAPfile (Text Scrap)
                 ;2=WRITEfile (GeoWrite file)
                 ;3=VLIRfile (but not GeoWrite file)
                 ;4=ALBUMfile (text album)

;this next group of variables must stay grouped together, since they are
;cleared out in one batch.

nextbatch   = $099d     ;.block 2
ovrflbyte   = $099f
curpage     = $09a0
scrcode     = $09a1     ;.block 2
asciicode   = $09a3     ;.block 2
petcode     = $09a5     ;.block 2

cur_table   = $09a7           ;variables defining the location and
                 ;size of the active shellwindow.
lft_shell   = $09a7     ;.block 2
                 ;left position word (0-319 or 0-639).
rht_shell   = $09a9     ;.block 2
                 ;right position word (0-319 or 0-639).
left_shell  = $09ab           ;left position of window on screen in bytes.
wid_shell   = $09ac           ;width of the bitmap in bytes.
lft_window  = $09ad     ;.block 2
```

```
rht_window = $09af     ;.block 2
col0  = $09b1     ;.block 2
rht_column = $09b3     ;.block 2

en_cur_table      = $09b5

buff_0      = $09b5     ;.block 81
buff_1      = $0a06     ;.block 81
buff_2      = $0a57     ;.block 81
buff_3      = $0aa8     ;.block 81
buff_4      = $0af9     ;.block 81
buff_5      = $0b4a     ;.block 81
buff_6      = $0b9b     ;.block 81
buff_7      = $0bec     ;.block 81
buff_8      = $0c3d     ;.block 81
buff_9      = $0c8e     ;.block 81
buff_10     = $0cdf     ;.block 81
buff_11     = $0d30     ;.block 81

;$0d81-$0d83 not used.

;most recent command strings stored here.
bufmem10    = $0d84     ;.block 80
bufmem9     = $0dd4     ;.block 80
bufmem8     = $0e24     ;.block 80
bufmem7     = $0e74     ;.block 80
bufmem6     = $0ec4     ;.block 80
bufmem5     = $0f14     ;.block 80
bufmem4     = $0f64     ;.block 80
bufmem3     = $0fb4     ;.block 80
bufmem2     = $1004     ;.block 80
bufmem1     = $1054     ;.block 80

CmdBuffer   = $10a4     ;.block 82

;$10f6-$10ff is currently unused.

StUpBuffer  = $1100     ;.block $600
            ;buffer for startup file.
            ;exec command also uses this buffer.
EnOfStUpBuffer    = $16ff

EnOfRamsect = $16ff

Trans1Start = $6800

Trans2Start = $1100     ;.block 4
T2Name      = $1104     ;.block    17
T2Instruct  = $1115
;the next three addresses form a jump table into the Trans2 Command.
T2Jump      = $1116     ;.block    3
                  ;enter here when the user types the
                  ;command, if bit 7 is set in T2Instruct.
T2ReDraw    = $1119     ;.block    3
```

```
                  ;enter here whenever the screen is
                  ;redrawn if bit 5 is set in T2Instruct.
T2Clear     = $111c     ;.block 3
                  ;this jump is for clearing the command
                  ;from memory. It allows the command to
                  ;perform any necessary action before it
                  ;is removed.
;the remainder of Trans2Area occupies memory up to $13ff.
;It is also possible for one command to occupy Trans3area also.

Trans3Start= $1400     ;.block 4
T3Name      = $1404     ;.block    17
T3Instruct  = $1415
;the next three addresses form a jump table into the Trans3 Command.
T3Jump      = $1416     ;.block    3
                  ;enter here when the user types the
                  ;command, if bit 7 is set in T3Instruct.
T3ReDraw    = $1419     ;.block    3
                  ;enter here whenever the screen is
                  ;redrawn if bit 5 is set in T3Instruct.
T3Clear     = $141c     ;.block 3
                  ;this jump is for clearing the command
                  ;from memory. It allows the command to
                  ;perform any necessary action before it
                  ;is removed.
;the remainder of Trans3Area occupies memory up to $16ff.

;the layout of these two transient command areas is different from the
main
;transient command area that is located at $6800. These two have to
follow
;a different set of rules. Before the command will be executed, geoSHELL
has
;to find the string "TCOM" at the start of the respective area.
;Following this will be the null-terminated name of the command, padded
with
;zeros out to a total of 17 bytes. If the user types the name of the
command,
;geoSHELL will execute it based on the bits in the byte located at
;T2Instruct/T3Instruct. Refer to the programming manual for further info
;on these two command areas.

;The bytes at T2Instruct/T3Instruct tell geoSHELL how to deal with the
;command. The bits are explained below.
;Following this byte is a jump table for geoSHELL to access. The address
;that will be jumped to is based on T2Instruct/T3Instruct.

;bit 7 set  jmp to T2Jump/T3Jump.
;bit 6 set  reload the command from disk and execute it based on
;           this same byte after it is loaded.
;bit 5 set  jmp to T2ReDraw/T3ReDraw everytime the screen is redrawn.

;if T2Instruct/T3Instruct = 0 then the command will be ignored
altogether.
```

```
;****************************************************************
;
;           ShellEquates

;     Copyright 1994 by Maurice Randall
;     Charlotte, Mich. 48813
;
;     For Use Only By Registered Owners Of The
;     geoSHELL Programmer's Development Package
;
;****************************************************************

;.included in GSVariables

TRANSCOMM   =     21    ;the GEOS filetype assigned to
                  ;geoSHELL commands.

;The following define colors that may be used on the 128's 80 column
screen.

BLACK80     =     0
DK80GREY    =     1
DK80BLUE    =     2
LT80BLUE    =     3
DK80GREEN   =     4
LT80GREEN   =     5
DK80CYAN    =     6
LT80CYAN    =     7
DK80RED     =     8
LT80RED     =     9
DK80PURPLE  =     10
LT80PURPLE  =     11
DK80YELLOW  =     12
LT80YELLOW  =     13
LT80GREY    =     14
WHITE80     =     15


;Miscellaneous equates:

maxlines    = 12  ;number of lines in the window.
buff_size   = 81  ;size of certain buffers.
wind_left   = $08 ;pixels from left of screen.
wind_top    = $20 ;pixels from top of screen.
firstrow    = wind_top+30    ;first text row.
shell_height     = 136 ;pixels.
shell_width= 37  ;bytes.

top_window  = wind_top+16    ;top of pad area.
bot_window  = wind_top+shell_height-9 ;bottom of pad area.
row0  = firstrow
Quit_X      = (wind_left+$ff)/8 ;x position for icon (bytes from left
                  ;of screen)
Quit_Y      = wind_top+2     ;y position for icon (pixels from top)
```

```
Quit_Wide   = 3   ;icon width
Quit_Height= 14  ;and height.
ActHLeft    = 5
ActHHeight  = 16

LODRIVE     = 8   ;drive A.
HIDRIVE     = 11  ;drive D. (highest number allowed)

TComCk      = $6700     ;used when searching for a transient.
EnOfShell   = $6700     ;error channels are transferred here.

HotKeyArea  = $6c00     ;this is where the hotkeys are loaded
                  ;in from disk whenever the user types
                  ;a two-letter command. geoSHELL then
                  ;checks this area for a matching
                  ;hotkey definition. Whenever this is
                  ;done, if a transient command is
                  ;present, it is flushed from memory
                  ;since it would corrupt the area.



STOP_KEY    = 63
CTRL_KEY    = 58
PAUSE_KEY   = 87
XSCAN = $d02f
Y_KEY = 25
N_KEY = 39

SEQfile     = 0   ;equates used to identify the type
SCRAPfile   = 1   ;of file for use with the 'type'
WRITEfile   = 2   ;command.
VLIRfile    = 3
ALBUMfile   = 4


drv64path   = $c3e9     ;specifies the drive to use for the
                  ;path when searching for a
                  ;transient command.
drv128path  = $c9d7
part64string     = $c3ea     ;and this is the partition to use.
                  ;It is in the form of a 3 byte ascii string.
part128string = $c9d8
;these are equates used for message displaying.
;lda with any of these values before calling 'Message' and
'OtherMessage'.
;A routine would look like this:
;
;       ...
;       LoadW     r0,#texttodisplay ;point to the text.
;       lda  #(IN_TWO|TR_CR)  ;insert 2 spaces, add a carriage return.
;       jsr  OtherMessage     ;go display it.
;       ...
;texttodisplay:
;       .byte     "This Text Will Be Displayed!",0
;       ...

IN_CR = %10000000 ;initial carriage return.
                  ;otherwise begin at present cursor
                  ;location.
TR_CR = %01000000 ;carriage return after text.
                  ;otherwise leave cursor at next
                  ;character past text.
IN_ONE      = %00000001 ;one leading space
IN_TWO      = %00000010 ;two leading spaces...
IN_THREE    = %00000011 ;etc...
IN_FOUR     = %00000100
IN_FIVE     = %00000101
IN_SIX      = %00000110
IN_SEVEN    = %00000111
TR_ONE      = %00001000 ;one trailing space
TR_TWO      = %00010000 ;two trailing spaces...
TR_THREE    = %00011000 ;etc...
TR_FOUR     = %00100000
TR_FIVE     = %00101000
TR_SIX      = %00110000
TR_SEVEN    = %00111000
;The Jump Table:
InitgeoSHELL      =     $1700
Only128     =     $1703
OpenGS      =     $1706
NoMoreCmds  =     $1709
ResetScreen =     $170c
ClearScreen =     $170f
ClearWindow =     $1712
EraseWindow =     $1715
Asc_3_Byte  =     $1718
ReDoWindow  =     $171b
R_Icons     =     $171e
OpenWrFile  =     $1721
PushStuff   =     $1724
PopStuff    =     $1727
ResetFromPath     =     $172a
CkForDisk   =     $172d
ParamName   =     $1730
DoRun =     $1733
IncCmdPointer     =     $1736
CkPath      =     $1739
DriveLetter =     $173c
DispLetter  =     $173f
OpenPathPartition=     $1742
Switch      =     $1745
;$1748 is currently unused
ClrKeyData  =     $174b
YesKeypress =     $174e
NoKeypress  =     $1751
nogood      =     $1754
yesgood     =     $1757
DsplyLine   =     $175a
```

```
DsplyString =     $175d
CarrReturn  =     $1760
set_XY_pos  =     $1763
save_XY_pos =     $1766
ClearLine   =     $1769
ClearBoth   =     $176c
ClearRemainder    =     $176f
StripRulers =     $1772
AdjTxtPointer     =     $1775
ReDisplayWindow   =     $1778
Message     =     $177b
OtherMessage      =     $177e
GetMessage  =     $1781
Asc_Byte    =     $1784
ByteWZ_Ascii      =     $1787
ByteNZ_Ascii      =     $178a
ConvertK    =     $178d
Asc_BCD     =     $1790
CkESC_RULER =     $1793
DispText    =     $1796
GetMess     =     $1799
execStartup =     $179c
CkPathOrder =     $179f
FindParName =     $17a2
Wait  =     $17a5
ExitCommand =     $17a8
unsetPrompt =     $17ab
ColorScreen =     $17ae
SetBrdrColor      =     $17b1
SetThisDevice     =     $17b4
SetDrName   =     $17b7
FileNotAvail      =     $17ba
Searching   =     $17bd
MissgFilename     =     $17c0
NotFound    =     $17c3
Dir   =     $17c6
CkTerminators     =     $17c9
CkTermNoSpace     =     $17cc
MissDisk    =     $17cf
CkAllDrives =     $17d2
;this spot is currently unused ($17d5)
CkKeyboard  =     $17d8
CkPresdrive =     $17db
CkThisDrive =     $17de
PurgeAllTurbos    =     $17e1
CkOtherDrives     =     $17e4
DateDisplay =     $17e7
Byte_Time   =     $17ea
;$17ed is currently unused
FixColors   =     $17f0
MskCtrlCodes      =     $17f3
ConvToAscii =     $17f6
GetHeader   =     $17f9
LoadProblem =     $17fc
```

```
OpenGSIndex =     $17ff
RdFnKeyDefs =     $1802
Open_GC_Channel  =     $1805
Send_GC_Byte     =     $1808
SaveToREU   =     $180b
ClrTrnsName =     $180e
CkDrvLetter =     $1811
Status      =     $1814
SendCmd     =     $1817
ReadChannel =     $181a
SendListen  =     $181d
SendTalk    =     $1820
FindRamLink =     $1823
GetCMDType  =     $1826
FindCMDType =     $1829
OpenPartition    =     $182c
SetPartParams    =     $182f
InitCurPartition =     $1832
OpenCurPartition =     $1835
CkModeFlag  =     $1838
GetRealDriveNum  =     $183b

;room for 3 more jump addresses for future use.
```

```
;*****************************************************
;
;           Sample.lnk
;
;     Link file for the 'sample' transient command for geoSHELL

;     Copyright 1994 by Maurice Randall
;     Charlotte, Mich. 48813
;
;     For Use Only By Registered Owners Of The
;     geoSHELL Programmer's Development Package
;
;*****************************************************
.output           sample     ;name for output file.
.header           SampleHdr.rel    ;name of file containing header block.

.seq

.psect     $6800

GSVariables.rel
Sample.rel
```

```
;************************************************
;
;           SampleHdr
;
;     Header file for 'sample' command for geoSHELL
;     Use this as a template for any new command.
;     Copyright 1994 by Maurice Randall
;     Charlotte, Mich. 48813
;
;     For Use Only By Registered Owners Of The
;     geoSHELL Programmer's Development Package
;
;************************************************

.header                ;start of header section.

      .word 0
      .byte 3
      .byte 21

      .byte $83   ;Commodore file type, with bit 7 set.
      .byte TRANSCOMM   ;Geos file type. (21)
      .byte 0     ;Geos file structure type.
      .word Trans1Start ;start address of command (where to
                  ;load to - $6800)
      .word $7f3f ;highest end address.
      .word Trans1Start ;init address of command (where to JMP to).

;permanent command name must be 12 characters including spaces
;plus version number. The final byte here is the system byte. $40 allows
;this command to run on any GEOS system. (refer to appendix C)
      .byte "Sample      v1.0",0,0,0,$40

;twenty character author name including the null-terminator.
;change this to your own name.
      .byte "Maurice Randall    ",0

;don't change this! If you change it, your command will not be
recognized.
      .byte "CmdProcessorV2.0",0,0,0,0
      .block     23

;place the text here that you wish to have
;displayed in the Desktop's info box.

      .byte "Just change the wording here to "
      .byte "whatever you want displayed.",0

.endh
```

```
;************************************************************
;
;           Sample
;
;     External transient command source code
;     for the 'sample' command used with geoSHELL.
;     This may be used as a template for any new command.

;     Copyright 1994 by Maurice Randall
;     Charlotte, Mich. 48813
;
;     For Use Only By Registered Owners Of The
;     geoSHELL Programmer's Development Package
;
;************************************************************

.if   Pass1
.noeqin
.noglbl
.include   geosSym
.include   geosMac
.glbl
.eqin
.endif

      .psect

Sample:

;add any program code into this area.

      jmp   ExitCommand


SmplRamsect:

      .ramsect

;insert your own ramsect variables here.

EndSmplRamsect:

EndOfSample:

;from this point, on up to $7fff, may be used for a work area.
```

```
;************************************************
;
;           Color80.lnk
;
;     Link file for the 'color80' transient command for geoSHELL

;     Copyright 1994 by Maurice Randall
;     Charlotte, Mich. 48813
;
;     For Use Only By Registered Owners Of The
;     geoSHELL Programmer's Development Package

;************************************************
.output           color80    ;name for output file.
.header           Color80Hdr.rel   ;name of file containing header block.

.seq

.psect     $6800

GSVariables.rel
Color80src.rel
```

```
;*****************************************************
;
;           Color80Hdr
;
;     Header file for 'color80' transient command for geoSHELL

;     Copyright 1994 by Maurice Randall
;     Charlotte, Mich. 48813
;
;     For Use Only By Registered Owners Of The
;     geoSHELL Programmer's Development Package

;*****************************************************



.header                 ;start of header section.

      .word 0     ;first two bytes are always zero.
      .byte 3           ;width in bytes.
      .byte 21    ;and height in scanlines of:

      .byte $83   ;Commodore file type, with bit 7 set.
      .byte TRANSCOMM  ;Geos file type.
      .byte 0     ;Geos file structure type.
      .word Trans1Start ;start address of program (where to
                  ;load to).
      .word $7f3f ;highest end address. Not really
                  ;important for a geoSHELL command.
      .word Trans1Start ;init address of program (where to JMP to).

      .byte "Color80     v1.0",0,0,0,$c0
                  ;permanent command filename: 12 characters,
                  ;followed by 4 character version number,
                  ;followed by 3 zeroes,
                  ;followed by mode flag.
                  ;$c0 is for 80 column only.

      .byte "Maurice Randall    ",0
                  ;twenty character author name.

      .byte "CmdProcessorV2.0",0,0,0,0
      .block     23
      .byte "Use this command to set the screen colors "
      .byte "for geoSHELL while in 80 column mode.",0
.endh
```

```
;****************************************

;     Color80src

; transient command source code. This command
; may be used to set the screen colors while using
; geoSHELL in 80 column mode.

;     Copyright 1994 by Maurice Randall
;     Charlotte, Mich. 48813
;
;     For Use Only By Registered Owners Of The
;     geoSHELL Programmer's Development Package

;****************************************

.if   Pass1

.noglbl
.noeqin
.include   geosSym
.include   geosMac
.eqin
.glbl


.endif


      .psect

Color80:
      bit   screenmode ;what mode are we in?
      bmi   10$   ;branch if 80 columns.
      bvc   5$    ;branch if 128-40.
      jmp   Mode_64    ;handle the 64.
 5$
      jmp   Mode_40_128 ;handle the 128-40.
 10$
      jmp   Mode_80_128 ;go for it.
;a few equates are defined here.
lftmonlocation   = 10
uppmonlocation   = 48
lftsquare  = 44
topsquare  = 48
toptext          = topsquare+6

;color selections are stored here.
tempBackground:
      .block     1
tempShell:
      .block     1
tempText:
      .block     1
```

```
tempPad:
      .block     1
tempBorder:
      .block     1

;tell the 64 user that this command only works in 128 mode.
Mode_64:
      jsr   Only128
      jmp   NoMoreCmds ;this was in the original source.
                 ;Only128 now jumps through NoMoreCmds
                 ;making this step unnecessary.

;tell the user that this command only works in 80 column mode.
Mode_40_128:
      LoadW r0,#only80text
      lda   #(IN_THREE|TR_CR)
      jsr   OtherMessages
      jmp   NoMoreCmds

only80text:
      .byte "This is an 80 column command!",0
onlycolortext:
      .byte "Issue the RGB command first!",0

MonitorPic:


MonPicWidth = picW
MonPicHeight = picH

SolidSquare:

squarewidth = picW
squareheight = picH

EmptySquare:

;there is a real icon here. It is actually
;a blank photo scrap.
BlankSquare:



Mode_80_128:
      bit   videomode  ;are we in color mode?
      bmi   10$   ;branch if so.
      LoadW r0,#onlycolortext
      lda   #(IN_THREE|TR_CR)
      jsr   OtherMessages    ;tell the user to use RGB mode.
      jmp   NoMoreCmds ;and exit.
 10$
      ldx   #0
 20$
      lda   backcolor,x ;move the current colors
      sta   tempBackground,x ;to a temporary storage area.
      inx
      cpx   #5
      bne   20$

      jsr   unsetPrompt ;eliminate the cursor blink.

;at this point we will set our own colors for color80 independently
;of those that may already be set.

      LoadB brdrcolor,#LT80GREY
      jsr   SetBrdrColor     ;change the border color.
      LoadB backcolor,#LT80GREY ;set the background color.
      LoadB textcolor,#BLACK80 ;set the text color.
      jsr   ClearScreen ;now ClearScreen will use our new colors.

      jsr   i_BitmapUp ;display the picture of the monitor.
      .word MonitorPic
      .byte lftmonlocation
      .byte uppmonlocation
      .byte MonPicWidth
      .byte MonPicHeight

      jsr   SetColTable ;color the little monitor.

      jsr   SetIcons    ;set up our own icons.
      jsr   AddText     ;display neccessary text to the screen.
      jsr   DispColors  ;display the color bar.

      rts         ;exit to main loop and let GEOS
                  ;get user input for the icons.

SetIcons:
      LoadW r0,#IconTable
      jsr   DoIcons     ;point GEOS to our little icons.
      php
      sei
      LoadW otherPressVector,#$00 ;make sure this is zero'd for
                  ;this command. (geoSHELL leaves the
                  ;mouse running)
      plp

      jsr   MouseOff
      LoadB mouseBottom,#168 ;keep the mouse off the lower part
                  ;of the screen. Or it will do
                  ;funny things to the screen.
      jmp   MouseUp
IconTable:
      .byte 22
      .word 0
      .byte 0

IconPointer:
      .word SolidSquare
      .byte lftsquare
      .byte topsquare
      .byte squarewidth
      .byte squareheight
      .word DoColorSource

      .word EmptySquare
      .byte lftsquare
      .byte topsquare+16
      .byte squarewidth
      .byte squareheight
      .word DoColorSource

      .word EmptySquare
      .byte lftsquare
      .byte topsquare+32
      .byte squarewidth
      .byte squareheight
      .word DoColorSource

      .word EmptySquare
      .byte lftsquare
      .byte topsquare+48
      .byte squarewidth
      .byte squareheight
      .word DoColorSource

      .word EmptySquare
      .byte lftsquare
      .byte topsquare+64
      .byte squarewidth
      .byte squareheight
      .word DoColorSource

      .word EmptySquare
      .byte lftsquare
      .byte topsquare+80
      .byte squarewidth
      .byte squareheight
      .word DoExit

;the icon table from the previous page continues here.

ColorIcons:
      .word BlankSquare
      .byte 16,152,3,8
      .word DoCBar
      .word BlankSquare
      .byte 19,152,3,8
      .word DoCBar
      .word BlankSquare
      .byte 22,152,3,8
      .word DoCBar
      .word BlankSquare
      .byte 25,152,3,8
      .word DoCBar
      .word BlankSquare
      .byte 28,152,3,8
      .word DoCBar
      .word BlankSquare
      .byte 31,152,3,8
      .word DoCBar
      .word BlankSquare
      .byte 34,152,3,8
      .word DoCBar
      .word BlankSquare
      .byte 37,152,3,8
      .word DoCBar
      .word BlankSquare
      .byte 40,152,3,8
      .word DoCBar
      .word BlankSquare
      .byte 43,152,3,8
      .word DoCBar
      .word BlankSquare
      .byte 46,152,3,8
      .word DoCBar
      .word BlankSquare
      .byte 49,152,3,8
      .word DoCBar
      .word BlankSquare
      .byte 52,152,3,8
      .word DoCBar
      .word BlankSquare
      .byte 55,152,3,8
      .word DoCBar
      .word BlankSquare
      .byte 58,152,3,8
      .word DoCBar
      .word BlankSquare
      .byte 61,152,3,8
      .word DoCBar

;this routine just puts the text on the screen that you see when
;you execute color80. GEOS PutString is used for this.
AddText:
      jsr   i_PutString
      .word 8*lftsquare+48
      .byte toptext

      .byte "Background Color"
      .byte GOTOXY
      .word 8*lftsquare+48
      .byte toptext+16
      .byte "Shell Color"
      .byte GOTOXY
      .word 8*lftsquare+48
      .byte toptext+32
      .byte "Text Color"
      .byte GOTOXY
      .word 8*lftsquare+48
      .byte toptext+48
      .byte "Pad Color"
      .byte GOTOXY
      .word 8*lftsquare+48
      .byte toptext+64
      .byte "Border Color"
      .byte GOTOXY
      .word 8*lftsquare+48
      .byte toptext+80
      .byte "Exit",0

      rts


;here's where we draw the colorbar that is near the bottom.

DispColors:
      jsr   i_FrameRectangle ;first build a frame around the area.
      .byte 151
      .byte 160
      .word 127
      .word 512
      .byte 255

      lda   textcolor   ;put the text color into the
      asl   a     ;high nybble.
      asl   a
      asl   a
      asl   a
      ldx   #0
 10$
      sta   cbarColors,x     ;and build a table of colors
      clc         ;with each possible background
      adc   #1    ;color in the low nybble.
      inx         ;we're building the table because
      cpx   #16   ;textcolor can vary.
      bne   10$

      LoadW r0,#cbarTable    ;use a geoSHELL routine to put the
      LoadW r1,#cbarColors   ;colorbar colors on the screen.
      jsr   ColorScreen
                  ;originally, some more code was
                  ;going in here, that's why the jsr
                  ;followed by an rts.
      rts

;this is a table of compressed bytes that ColorScreen recognizes.
;refer to the appendix and the ColorScreen routine for info on
;how this works. You might call this table a 'color scrap'.

cbarTable:
      .byte 19,16 ;start at row 19, column 16.
      .byte 1     ;do the next
      .byte 16    ;16 commands one time.
      .byte 3,0,3,1,3,2,3,3,3,4,3,5,3,6,3,7
      .byte 3,8,3,9,3,10,3,11,3,12,3,13,3,14,3,15
      .byte 0     ;a zero command ends the scrap.

;this is a table of color combinations that ColorScreen reads
;to use with the compressed bytes above. This table is constructed above.
cbarColors:
      .block     16
colorsource:           ;identifies the area that is currently
      .block     1     ;selected, such as 'Border Color'.

;this is jumped to when the user clicks on one of the area icons.
DoColorSource:
      MoveB r0L,colorsource  ;GEOS sets r0L. We save it here.
      jsr   ClearSquares     ;make all the icons point to an
                 ;empty icon picture.
      lda   colorsource
      asl   a
      asl   a
      asl   a
      tax
      jsr   SetSquare   ;set one pointer to the icon
                 ;that has been clicked on.
      jmp   SetIcons    ;and let GEOS redraw the icons.

;this is jumped to when the user clicks on 'exit'.
DoExit:
      ldx   #0
 10$
      lda   tempBackground,x ;put all selected colors into
      sta   backcolor,x ;the table that geoSHELL sees.
      inx
      cpx   #5
      bne   10$
      jsr   ClearScreen ;clear the entire screen.
      jsr   ReDoWindow  ;and get the geoSHELL window back.
      jsr   R_Icons     ;make sure the default icon is set.
      jmp   ExitCommand ;and exit cleanly.

;point all icons in the table at an empty square.
ClearSquares:
      ldx   #0
 10$
      lda   #[EmptySquare
      sta   IconPointer,x
      lda   #]EmptySquare
      sta   IconPointer+1,x
      txa
      clc
      adc   #8
      tax
      cpx   #40
      bcc   10$
      rts

;point the desired icon at a solid square.
SetSquare:
      lda   #[SolidSquare
      sta   IconPointer,x
      lda   #]SolidSquare
      sta   IconPointer+1,x
      rts

;when the user clicks on a color in the colorbar, GEOS jumps here.
DoCBar:
      lda   r0L
      sec
      sbc   #6
      ldx   colorsource ;get the current area that we will change.
      sta   tempBackground,x ;and store the color there.
      jsr   SetColTable ;this should have been made to just
      rts         ;jump through to the next routine.

;here is where we change the colors on the color80 screen as the
;user selects a different color for a certain area. This will
;alter the color table that ColorScreen will use when it colors
;the little monitor that is drawn on the screen.

SetColTable:
      lda   tempBorder
      asl   a
      asl   a
      asl   a
      asl   a
      ora   tempBackground
      sta   ColMon0
      lda   tempShell
      asl   a
      asl   a
      asl   a
      asl   a
      ora   tempPad
      sta   ColMon1
      lda   tempText
      asl   a
      asl   a
      asl   a
      asl   a
      ora   tempPad
      sta   ColMon2
      LoadW r0,#ColMonTable  ;point r0
      LoadW r1,#ColMon0 ;and r1
      jmp   ColorScreen ;and recolor the little monitor.

;this is the color scrap that is used to color the little monitor.
```

```
ColMonTable:
      .byte uppmonlocation/8+1
      .byte lftmonlocation+2
      .byte 1,4
      .byte 27,0,53,255,3,0,21,1
      .byte 3,6
      .byte 3,0,53,255,3,0,1,1,19,2,1,1
      .byte 1,9
      .byte 3,0,53,255,3,0,21,1,3,0,53,255,27,0,53,255,27,0
      .byte 0
;and this is the table of colors that is used with the color scrap
;for the little monitor.
;for this we only need three different color combinations to
;color the little monitor on the screen.
;this table is built as the user clicks on the colorbar.
ColMon0:
      .block     1
ColMon1:
      .block     1
ColMon2:
      .block     1
```