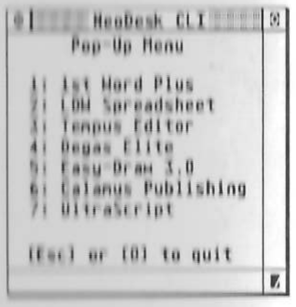
NeoDesk CLI

Features Include:



```
NeoDesk CLI
    Pop-Up Menu

 1: 1st Word Plus
 2: LDW Spreadsheet
 3: Tempus Editor
 4: Degas Elite
 5: Easy-Draw 3.0
 6: Calamus Publishing
 7: UltraScript

[Esc] or [0] to quit
```

- Easy to learn and use.
- Works in a GEM window.
- Built-in help for all commands.
- Can run any GEM or TOS program.
- Extensive library of built-in functions and commands.
- UNIX™ and MS-DOS style commands.
- Relatively small file size.
- Calls many of the *NeoDesk* functions directly.
- Uses the same, high speed, multiple file copy system as *NeoDesk*.
- Automate complex file operations.
- Different font sizes.
- Special MENU command allows you to create interactive menus (see above).
- Includes sample batch files.
- Compatible with any TOS, ST^E^, Stacy, TT, Moniterm, Turbo16 and ISAC systems.
- With *NeoDesk* resident, you can use it from inside any GEM program.

Suggested Retail: $29.95

Requires *NeoDesk 2.0*, version 2.05 or later.

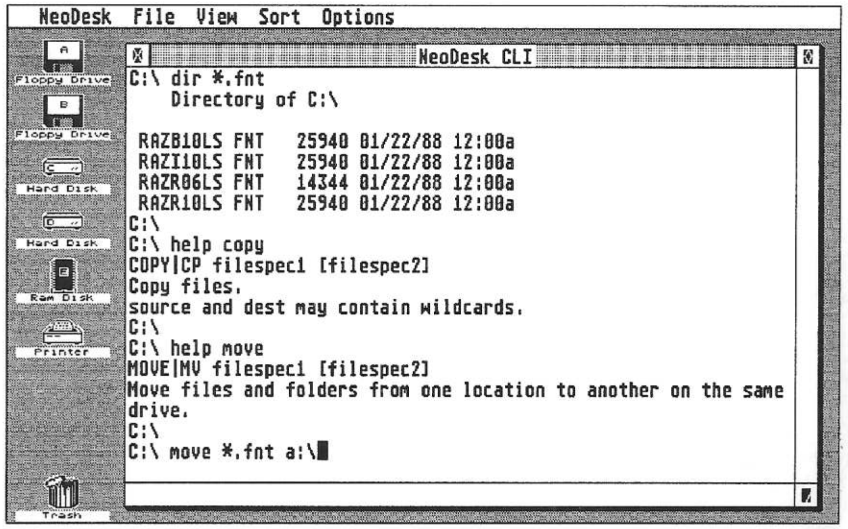
SERIAL NUMBER:

GRIBNIF SOFTWARE

P.O. Box 350 • Hadley, MA 01035 • USA • (413) 584-7887

NeoDesk CLI

A complete GEM based command line interpreter accessory for the NeoDesk desktop replacement package.



```
NeoDesk CLI
C:\ dir *.fnt
    Directory of C:\

 RAZB10LS FNT    25940 01/22/88 12:00a
 RAZI10LS FNT    25940 01/22/88 12:00a
 RAZR06LS FNT    14344 01/22/88 12:00a
 RAZR10LS FNT    25940 01/22/88 12:00a
C:\
C:\ help copy
COPY|CP filespec1 [filespec2]
Copy files.
source and dest may contain wildcards.
C:\
C:\ help move
MOVE|MV filespec1 [filespec2]
Move files and folders from one location to another on the same
drive.
C:\
C:\ move *.fnt a:\
```

Add the power and flexibility of a GEM based command line interpreter to your *NeoDesk* desktop. Automate complex operations, create interactive menus, manipulate large numbers of files efficiently, and much more!

Works with all Atari ST compatible systems, requires NeoDesk 2.0, version 2.05 or later.

GRIBNIF SOFTWARE

NeoDesk CLI

A complete GEM based command line interpreter accessory for *NeoDesk* version 2.05 or later.

from
GRIBNIF SOFTWARE

by
Dan Wilga

Manual by Rick Flashman

Program written by Dan Wilga
Program design by Dan Wilga & Rick Flashman
Manual written by Rick Flashman
Manual edited by Tricia Metcalf

The NeoDesk CLI was written in Borland International's Turbo C. This manual was written in 1st Word Plus and typeset with Calamus Desktop Publishing on an SLM804 Laser Printer.

Gribnif Software
P.O. Box 350
Hadley, MA 01035
U.S.A.
Tel: (413) 584-7887
Fax: (413) 584-2565

Printed in the United States of America
First Edition

Table of Contents

Introduction

• Program Overview

The NeoDesk CLI is a *NeoDesk Accessory* which adds a complete command line interpreter to NeoDesk 2.05 or later. This CLI runs out of its own GEM window and allows you to enter both MS-DOS and UNIX™ style commands. With version 2.05 of NeoDesk this accessory runs as a standard desk accessory, with later versions of NeoDesk you also have the option to run it as separate *NeoDesk Program*.

The NeoDesk CLI does not attempt to offer an environment of its own, instead it adds a whole new world of capabilities including batch files, interactive menus, and much more to NeoDesk.

By hooking into NeoDesk itself, the CLI is able to directly call many of the NeoDesk functions, something which gives you several benefits:

- A relatively small CLI accessory. This is because the CLI is able to call many of the NeoDesk functions directly, and thus they do not have to be duplicated inside the CLI itself.
- Very reliable and efficient file management utilities. It uses the NeoDesk file management functions which have already been tested and proven. You also benefit from the high-speed multiple file copying feature of NeoDesk.
- Ability to execute any program with the same high level of compatibility that NeoDesk offers.
- A clean and efficient union between NeoDesk and the CLI which results in a more efficient, complete, and consistant environment.

• Problems & Technical Support

Gribnif Software has tried very hard to make this software package as free from defects as possible. But, like all computer software, it is never quite finished, so please make sure to mail in your warranty card so we may notify you of major upgrades of the NeoDesk CLI as they become available. We also require that your warranty card be on file with us before we are able to give out any technical support for the package.

If you have any questions or problems regarding the NeoDesk CLI, you can write to us at: *Gribnif Software, P.O. Box 350, Hadley, MA 01035, USA*. If you choose, you may call our technical support line at (413) 584-7887 during normal business hours (Eastern Time). You may also send us a FAX at (413) 584-2565. We will try to answer your questions as quickly as possible.

We are also available on GEnie (General Electric's Network for Information Exchange); you will find us there under the electronic mail address of GRIBNIF or in the Atari ST Roundtable, Category 17.

• About This Manual

This manual tells you everything you need to know in order to operate the NeoDesk CLI on any Atari ST compatible computer. You will find the CLI to be a very easy and intuitive program to use, but if you do not read the entire manual you will probably miss out on a number of very useful and powerful commands.

This manual takes for granted that you are already familiar with the operation of NeoDesk and the Atari ST computer system. If you have any questions or problems with either of these, please consult your NeoDesk or computer manual.

If any changes or corrections have been made to this program or documentation since the printing of this manual, you will find a file on the disk named **"READ_ME.NOW"**. Open this file (double-click on it from NeoDesk) to display its contents. You might find it useful to print this file, since it will contain important changes and additions to this manual.

• About NeoDesk Accessories & Programs

NeoDesk version 2.05 (and above) supports what are commonly referred to as *NeoDesk Accessories*. These are special desk accessories which have been designed to hook directly into NeoDesk itself. This allows them to access many of the NeoDesk functions directly. The freeware *NeoDesk Recoverable Trashcan* is a good example of this.

In versions of NeoDesk later than 2.05 there is also the additional support for *NeoDesk Programs*. These are specially written programs which can also hook directly into NeoDesk itself.

The NeoDesk CLI is able to behave both as a *NeoDesk Accessory* and a *NeoDesk Program*. If you are using a version of NeoDesk later than 2.05, simply rename it from its ".ACC" file ending to the ".NPG" or ".NTP" file ending to use it as a GEM based *NeoDesk Program*. For more information check your NeoDesk manual.

Getting Started

• Making a Backup Copy

You should make a backup copy of the NeoDesk CLI disk as soon as possible. Floppy disks are magnetic media, and you never know what can happen to them. We **DO NOT** recommend that you use the *original* NeoDesk or NeoDesk CLI disks for daily use. Instead we recommend you use a backup copy and keep the original disks stored away in a safe location.

• Installing the NeoDesk CLI

The NeoDesk CLI is most commonly used as a desk accessory. Because of this, you will want to install it on your boot disk or hard drive. The NeoDesk CLI must be located on the root directory (which is what you see when you are looking at a disk's window without looking inside any of its folders) of your boot disk or hard drive. This should be the same place that any other desk accessories you are using (such as the *NeoDesk Control Panel*) are stored.

You will also have to tell NeoDesk that you are now using the NeoDesk CLI. With version 2.05 this is done by creating a **NEO_ACC.INF** file which resides in the same location that your copy of NeoDesk is loaded from. This file tells NeoDesk which *NeoDesk Accessories* you currently have loaded. If using a version of NeoDesk later than 2.05, consult your NeoDesk manual for the correct information on how to install a *NeoDesk Accessory*.

If you are using a version of NeoDesk later than 2.05 then you can also run the NeoDesk CLI as a separate *NeoDesk Program*. Check your NeoDesk manual for additional information.

• Using the Automatic Installation Program

To assist you in installing the NeoDesk CLI we have included a simple program which will automate the entire installation procedure. This installation program may be used again if necessary.

If you are loading NeoDesk off a boot disk (and are not using a hard disk) you will probably want to install the NeoDesk CLI to that same floppy disk. If you are loading NeoDesk off a hard disk you will probably want to install the NeoDesk CLI on the C: partition.

The automatic installation program will copy the NeoDesk CLI accessory to the root directory of your boot disk or hard drive. It will also ask you where your copy of NeoDesk is stored so that it can copy the **NEO_ACC.INF** file to it. To use this program, follow these simple instructions:

1. Insert your original NeoDesk CLI disk into drive A.
2. Open a window to drive A. You should see the **INSTALL.PRG** icon.
3. Run the **INSTALL.PRG** program. (Open it or Double-Click on it with the mouse.)
4. Follow the simple on-screen instructions.

• Installing the CLI Manually

If you do not feel like using the automatic installation program you may install the CLI manually. The NeoDesk CLI accessory must reside in the boot disk or hard drive from which you normally load your desk accessories. For NeoDesk 2.05, the **NEO_ACC.INF** file must be placed in the same directory from which NeoDesk is loaded. This file tells NeoDesk (version 2.05) which *NeoDesk Accessories* (like the NeoDesk CLI) you are currently running. Once you have done this, you will have to reboot the system so that the NeoDesk CLI will load into memory. If you are using more than one *NeoDesk Accessory*, be sure to read the next section.

• NeoDesk 2.05 and the NEO_ACC.INF File

The **NEO_ACC.INF** is a very special file which tells NeoDesk version 2.05 which *NeoDesk Accessories* it should look for. This file should be placed in the same directory that your copy of NeoDesk is normally loaded from. NeoDesk looks for this file every time it is loaded.

Anything in the **NEO_ACC.INF** file which starts with a semicolon is treated as a comment and ignored. In this file, NeoDesk expects to find a list of all the accessories that it should initialize. The file ending after the name of each accessory (.ACC) is optional. If you add a new *NeoDesk Accessory* to your system you must make sure to add its name to the list in the **NEO_ACC.INF** file or NeoDesk will never know it is there. The filenames must be in uppercase. If you edit this file using a word processor, be careful to save it as an ASCII file without formatting.

Placing an asterix "*" to the left of any accessory name tells NeoDesk that it should use that file as a *Batch File Interpreter*. This will supersede any other batch file interpreter which you may have entered under the "Set Preferences..." menu entry.

If the specified batch file interpreter accessory (such as the NeoDesk CLI) is not found and you tried to run a batch file, an error message will appear. Any other accessories which are not found will be ignored.

Versions of NeoDesk above 2.05 do not use the **NEO_ACC.INF** file and instead they are configured using the "Set Preferences..." menu entry. Consult your NeoDesk manual for more information.

Using the NeoDesk CLI

Once you have installed the NeoDesk CLI (see *Getting Started*), rebooted, and run NeoDesk, you are ready to use the NeoDesk CLI. NeoDesk should now be aware that the NeoDesk CLI is loaded into memory. You can verify this by pressing [Control][B]; the CLI window should open. If it does not open, be sure to verify your **NEO_ACC.INF** file and your NeoDesk version. You can close the CLI window by selecting the Close Box on the upper left hand corner of the window.

If you open a window to where you have the NeoDesk CLI stored on your drives, you will be able to drag its icon out to the NeoDesk Desktop. Once it is on the desktop you will be able to open the NeoDesk CLI by simply double-clicking on its icon just as if you were opening a disk drive. The NeoDesk CLI will also execute any batch files you open or drag to it. Check your NeoDesk manual for more information on desktop icons.

You can also open the NeoDesk CLI by selecting its menu entry under the "Desk" menu. Whenever you are at the NeoDesk desktop, it may also be opened by pressing [Control][B] as mentioned above.

If you set the "Use Master to Execute" or "Reload after Execute" (depending on your version of NeoDesk) option under the "Set Preferences..." menu entry to "No" you will be able to use the NeoDesk CLI while running other GEM programs from NeoDesk. If you unload NeoDesk while running a program, or quit NeoDesk, the CLI will shut down until NeoDesk is loaded back into memory.

If you keep NeoDesk in memory while running another program the CLI will operate just like it does from the NeoDesk desktop, but since NeoDesk is no longer the current executing application several commands (SHOW, QUITNEO, and LOADINF) will be disabled. You also will not be able to run any programs from the CLI until you have returned to NeoDesk.

You can type in the NeoDesk CLI just as though you were typing in a text editor or word processor. Whenever you reach the right edge of the window the text will be wrapped around to the next line. You will notice that the CLI has a prompt in the left column which lets you know when it is waiting for input. The cursor is where your characters will appear as you type.

In an attempt to conserve memory, the NeoDesk CLI does not remember anything it displays. If for any reason the CLI window has to be redrawn its contents will be erased. This usually occurs whenever you move or resize the CLI window, but can also be affected by other windows and dialog boxes. This can actually be used as an easy way to clear the CLI window, just click with the mouse on the drag bar of the CLI window.

The window itself works just like any other standard GEM window. You can resize it, move it, and close it. The Full Size button on the upper right hand corner of the window will force the window to jump between the display size of your screen and its current size. Just like a NeoDesk window, if you resize the full screen size of the CLI window it will remember that size until you close the window.

• Executing a Batch File or Program

You can execute any program or batch file simply by typing its name at the prompt in the CLI, or by dragging its icon to the CLI desktop icon. The CLI will search in the current CLI path (or any of the paths specified in the PATH environmental variable) for the file to be executed.

When it finds a program to be executed, it gives its name to NeoDesk which in turn executes the program for the CLI. Because of this, you cannot run programs while using the CLI inside other applications. NeoDesk needs to be the current application in order to run any programs.

If you wish to try out any of the sample batch files in this manual, simply use the TEXT command to create a test batch:

```
text test.bat
```

You can then run this batch file by typing:

```
test.bat
```

• Using Paths

Paths are used to tell the CLI where things are stored. Their usage is:

[drive:] [\path]

You can optionally specify the complete path. If you do not include a path, then the current CLI path is used. As a short cut, two periods can be used to represent the current path, one folder back. Examples of paths:

```
d:\data\stuff
```
(this path points to the "stuff" folder, which is inside the "data" folder, which in turn is in the root directory of drive D)

```
d:
d:\
```
(these both point to the root directory of drive D)

If you are currently in the path "d:\data\junk" and you wish to change to the path "d:\data\stuff" there are three ways you can do this:

```
cd ..
cd stuff
```
(this moves you back one folder and then into the "junk" folder)

```
cd d:\data\stuff
```
(this moves you directly into the "stuff" folder)

```
cd ..\stuff
```
(this takes you back one folder and into the "stuff" folder)

Do not confuse the **path** "D:" which points to the root directory of drive D with the **command** "CD D:" which will change to the directory the previous path of drive D, if there was one.

• Batch Files

A batch file is a series of CLI commands written into a text file. By giving the CLI the name of the batch file it will automatically execute all the commands in that file. This allows you to automate a complex series of commands and build advanced loops and conditional commands. In the **SAMPLES** folder of the NeoDesk CLI disk you will find a number of sample batch files.

As you probably already know, NeoDesk has some pretty sophisticated support for batch files built-in. If you double-click on any batch file NeoDesk will automatically open the specified batch file interpreter (such as the NeoDesk CLI) and give it the name of the batch file to execute.

NeoDesk supports two different types of batch files. The first type is simply known as a "Batch File". It ends in the standard ".BAT" file ending. The second is known as "Batch File Takes Parameters" and ends in the ".BTP" file ending. If you want to run a batch file with parameters from the CLI you may simply type its name followed by any parameters it might require. For example, if you had a BTP file named "SAMPLE.BTP" that required a single parameter, "data.txt", you could execute it directly from the CLI by typing:

```
sample data.txt
```

If you tried to execute the same file from the NeoDesk desktop you would be prompted with the Parameters dialog box. The path and name of the batch file will already be entered as the first parameter. Do not change this since it is needed by the CLI to find the batch file. After the path and name, enter any parameters needed by the batch file, such as "data.txt" in the above example.

• Environmental Variables

The NeoDesk CLI looks for two environmental variables. These can be specified under the NeoDesk "Edit Environment..." menu entry. If these variables are found, the CLI will use the information within them. The names of the environmental variables must be entered in uppercase and followed by an equals sign (=). Both the semi-colon (;) and comma (,) can be used to separate parameters in the environmental variables. See your NeoDesk manual for additional information.

PATH

The PATH variable defines where the CLI should look for executables and batch files. Whenever you enter a string that is not a built-in command of the CLI, the CLI looks for the PATH variable. If it exists then each path in it is searched for a file matching the string you entered. A typical example would be:

```
PATH=;c:\;c:\batch;c:\utils
```

The CLI always searches the paths from left to right. Notice that the first semi-colon is needed to indicate that the current CLI path should be searched first. In this example the CLI would first check the current CLI path, then the "c:\" path, followed by the "c:\batch" path, and finally the "c:\utils" path.

SUFF

The SUFF (stands for suffix) variable defines what file extensions the CLI should use to look for executables or batch files. If it does not exist, the CLI will then attempt to execute any file, no matter what its extension is. A common example would be:

```
SUFF=.prg,.app,.tos,.ttp,.btp,.bat,.exe
```

This example adds the EXE extension to the standard system extensions. NeoDesk is still in charge of deciding how to run each extension. Any non-standard extension is executed as a "PRG" type program. If you wish to execute any file without caring about the extension, enter a comma or semi-colon. The SUFF variable is also searched from left to right. If you are using both the PATH and SUFF variables then all the extensions in the SUFF variable are searched for the first path in the PATH variable, they are then searched for the next path, and so on until there are either no more paths or a match is found.

• Variables

The CLI supports up to 15 separate user-defined variables. These can contain any specified value of your choice. The CLI only

recognizes the first eight characters of a variable name, any extra characters are ignored. Their usage is:

$variablename

Variablename is the name of the variable. The "$" character indicates to the CLI that a variable name follows and it should substitute the name with its contents. This is also known as "variable expansion". To give a variable a specific value you use the SET command:

```
set number 42
```
(sets the variable "number" to the value "42")

Notice that the SET command does not use the "$" character when creating a variable. That is because you do not want the variable expanded yet, you simply are giving it the name of the new variable.

Any extra user variables can be removed with the UNSET command:

```
unset number
```
(removes any value from the "number" variable)

There are a number of system variables which cannot be modified. These can be displayed using the SET command. User variables appear at the bottom of the list:

```
0 =        }
1 =        }
2 =        }
3 =        }
4 =        }  (command parameter variables which are
5 =        }  passed to a batch file when executed)
6 =        }
7 =        }
8 =        }
9 =        }
# =         0       (total number of batch file parameters)
TOS_VER =   1.2     (the current GEM/TOS version)
NEO_VER =   2.05    (the current NeoDesk version)
CLI_VER =   1.0     (the current NeoDesk CLI version)
AT_NEO =    TRUE    (is NeoDesk the current application?)
```

```
.INPUT =      con:     (present input device (console))
.OUTPUT =     con:     (present output device (console))
COL =         17       (the current cursor column)
ROW =         15       (the current cursor row)
MAX_COL =     76       (the maximum possible column with the
                          present window and font size)
MAX_ROW =     19       (the maximum possible row with the
                          present window and font size)
FONT =        1        (the current font size)
CHAR_W =      8        (the current character width in pixels)
CHAR_H =      8        (the current character height in pixels)
LINE =        0        (the number of the last line read from a
                          batch file)
WINDOW =  10 10 500 150   (the current window size)
MAX_WIND = 0 11 640 189    (the largest possible window
                              size for that resolution)
DEBUG =       FALSE    (is the debug option on?)
PRMT_FULL =   TRUE     (is the prompt FULL option set?)
ITEM =        C:\DATA.INF    (the current item waiting to be
                                processed)
CWD =         A:\      (the current CLI path)
NAME =        Dave     (the value of a user defined variable)
```

• Labels

Labels are used in batch files to mark a special place in a batch file. You can use these to force the CLI to jump to a specific part of a batch file. Their usage is:

labelname; [*command*]

The *labelname* can be up to eight characters, uppercase or lowercase. Additional characters can be used for programming clarity, but they will be ignored by the CLI. The semi-colon at the end of a *labelname* indicates to the CLI that it is a *labelname*. Optionally, a *command* may be placed to the right of the *labelname*. Labels are used by the GOTO and MENU commands. For example:

```
echo "this line WILL get printed"
goto next
echo "this line will NOT get printed"
next;
echo "this line WILL also get printed"
```

• Parameters

Parameters are anything which goes to the right of a command. You can have up to 19 parameters in one line separated by tabs, spaces, or commas. A line like this has eight parameters:

```
echo This is a sample of the echo line.
```

If you wish to have a line with more than 19 parameters then you must use quotes, for example:

```
echo "1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5"
```

When a batch file runs, there are several special variables which contain parameters that were sent to it in addition to its own name. Note that a batch file can only receive up to nine parameters at any given time. These variables are:

$0	- Name of the batch file.
$#	- Number of parameters passed to the batch file.
$1..$9	- The actual parameters passed to the batch file.

• Redirection

The NeoDesk CLI allows for the redirection of the input and output of any CLI command or batch file. It does not allow for the redirection of the input or output of programs run by the CLI. NeoDesk, however, supports input and output redirection for any standard "TTP" program; check your NeoDesk manual under the "Open" command. The CLI keeps track of where the current CLI input and output are going to by means of two special variables:

.INPUT	Keeps track of the current input source.
.OUTPUT	Keeps track of the current output destination.

These can be the path and filename of any existing file (if only a filename is specified then the current CLI path is used), or one of the standard system devices:

CON:	*Console* (default)
AUX:	*RS-232 Port*
MIDI:	*Midi port*

IKBD: *Intelligent Keyboard*
NULL: *Null device*

The only input that the NULL: device ever returns is an end of file condition. It discards any output sent to it.

When redirection is used on the command line to initiate a batch file, .INPUT and .OUTPUT default to the names of the redirection file that was specified. Input and output can be temporarily redirected for any command in the batch file by using "->", "->>", "<-", and "<->" redirection characters or they can be changed for the duration of the batch file by SETting .INPUT or .OUTPUT. These variables can only be set from inside a batch file via the SET command.

• Special Characters

There are a series of special characters which have special meaning in the CLI. They are as follows:

; Semi-colon - when the first character on any line in a batch file, it indicates that the rest of the line should be treated as a comment and thus ignored. Otherwise, it is used to indicate a label name.

$ Dollar sign - precedes a variable name to be expanded to its stored value.

% Percent sign - precedes a function's name; tells the CLI that this is a function to be processed.

‹space›
‹tab›
, Space, Tab, and Comma - these can all be used to separate command parameters. To the CLI, the line "if,$x,==,1" is the same as "if $x == 1".

' Single quote - indicates that any text between the quotes should not be expanded and should be treated as one single parameter. For example, the following lines:

```
set foo 10
echo '$foo'
```

Would produce the output "$foo" rather than the value of the $FOO variable which is "10".

" Double quotes indicate that any text between the quotes should be expanded, but still treated as a single parameter. This simple set of commands shows how single and double quotes work:

```
set foo '1 == 1'

echo $foo
  TRUE
echo "$foo"
  1 == 1
echo '$foo'
  $foo
```

If you wish to insert one type of quote in a string you can always use the opposite type of quotes. For example, the following lines are valid:

```
echo "He's here."
  He's here.
echo '"Where is he?", he said.'
  "Where is he?", he said.
```

\ Within quotes, a backslash allows for the following constants:

\\$	- Insert the "$" character without expecting a variable.
\\"	- Insert the " character.
\\'	- Insert the ' character.
\\a	- Ring the system bell.
\\f	- Perform a form feed (clears the CLI window).
\\t	- Insert a horizontal tab.
\\r	- Insert a carriage return without linefeed.
\\n	- Insert a carriage return with linefeed.
\\	- Insert the \\ character.
\*NNN*	- Insert the character with the ASCII value of *NNN* (in octal).
\\x*NN*	- Insert the character with the ASCII value of *NN* (in hexadecimal).

\033
\x1b Escape characters - when followed by one of the following letters can be used to insert any of the Atari VT-52 escape codes (note that some are ignored):

A - Move the cursor up one line. Will not wrap around the edge of the window.
B - Move the cursor down one line. Will not wrap around the edge of the window.
C - Move the cursor right one space. Will not wrap around the edge of the window.
D - Move the cursor left one space. Will not wrap around the edge of the window.
E - Clear the CLI window, move the cursor on the upper lefthand corner of the CLI window. Same as \f.
H - Move the cursor to the upper lefthand corner of the CLI window.
I - Move the cursor up one line, scrolling if necessary.
J - Erase from the present cursor position to the end of the CLI window.
K - Erase from the present cursor position to the end of the current line.
L - Insert an empty line at the present cursor position, move the cursor to column 0.
M - Delete the line at the present cursor position, move the cursor to column 0.
Y *row column* - Position the cursor to the specified *row* and *column*. Row and column numbers are '\037' plus the number. For example, to set to row 1, column 1: '\033Y\040\040'
Z - Supress the carriage return at the end of an ECHOed line.
b *x* - Set the foreground color to *x*. Ignored by the CLI.
c *x* - Set the background color to *x*. Ignored by the CLI.
d - Erase from the beginning of the CLI window to the present cursor position.
e - Enable cursor. Ignored by the CLI.
f - Disable the cursor. Ignored by the CLI.
j - Save the present cursor position.
k - Move the cursor to the position stored with **Esc j**, default is row 0, column 0.
l - Erase the line at the present cursor position, move the cursor to column 0.

o - Erase from beginning of the current cursor line up to the present cursor position.
p - Enter reverse video mode.
q - Exit reverse video mode.
v - Wrap extra text at end of line.
w - Discard extra text at end of line.

The reverse video mode is turned off and the wrap mode is turned back on after every CLI command. These codes are not interpreted if the CLI's output has been redirected.

-> Redirect output - redirects the output of the specified command or batch file to a specified file or device. Its usage is:

command -> destination

Where *command* is the CLI command or batch file whose output is to redirected to the specified *destination*. The *destination* can be any valid path and filename. If only a filename is specified then the current CLI path is used. You can also redirect the output to one of the standard system devices (see **Redirection**). Any output sent to the NULL: device is discarded. The "->" characters must be the next to last parameter.

If used on the command line that starts a batch file, it will cause all the output of that batch file to be redirected, unless the batch file explicitly SETs .OUTPUT or uses the "->" character with one of its commands.

For example, if you wanted to redirect the output of the DIR command to the printer and thus get a printed directory listing you would type:

```
dir -> prn:
```

->> Append output - Appends the output for the specified command or batch file to the specified file or device. Its usage is identical to "->".

The major difference from the "->" character is that now the output will be appended to any file it is output to.

<- Redirect input - redirect the input for the specified command or batch file from the specified file or device. Its usage is:

command <- source

The *command* is the CLI command or batch file to receive input from the specified *source*. The *source* should be the path and filename of an existing file. If only a filename is specified then the current CLI path is used. You can also redirect the input from one of the system devices (see **Redirection**). The only input that the NULL: device ever returns is an end of file condition. The "->" character must be the next to last parameter.

For example, by using the input and output redirection you could actually write a very primitive terminal program with the CLI:

```
for
  if %hasch
    getch a
    echo nl- "$a" -> aux:
  endif
  if %hasch <- aux:
    getch a <- aux:
    echo "\O33Z$a"
  endif
endfor
```

<-> Redirect input and output - redirects the input and output of a batch file or command to the specified file or device. Its usage is:

command <-> item

Where *command* is the name of the batch file or command whose input and output is to be redirected to and from the specified *item*. The *item* can be any valid path and filename of an existing file. If no path is specified then the current CLI path is used. You can also redirect the output to one of the standard system devices (see Redirection). The NULL: device returns no input other than an end of file condition and discards all output.

For example, if you wanted to run the "sample.bat" batch file via the RS-232 port you would type:

```
sample <-> aux:
```

• Operators

The NeoDesk CLI supports a series of special operators. These are evaluated from left to right when they are found. Each operator must have one operand to the left and another one to the right. Arithmetic operators require that both operands be integers in decimal. Arithmetic calculations are performed to 31 bits of precision, so the possible range is -2147483648 to 2147483647. Operators with the highest precedence are first in the list:

MULT, DIV	(multiply, divide)
+, -	(add, substract)
~	(concatenate)
<, <=, >, >=	(less than, less than or equal to, greater than, greater than or equal to)
==, !=	(equal to, not equal to)
&&, ||	(logical AND, logical OR)

In the case of the DIV operator, the result is always the smallest possible integer. Both the MULT and DIV operators must be in uppercase to be treated as operators. To use "MULT" or "DIV" as a filename either enclose them in quotes or use lowercase.

The comparison operators (<, <=, >, >=, ==, !=) will first try to treat both operators as numbers. If either operator is not an integer, however, a string comparison is done. One string is greater than another if it is later alphabetically (for example A is less than B).

The && and || operators require that both operands be either TRUE or FALSE, example:

```
echo 'Enter your name:'
getstr name
if $name == 'Dave' || $name == 'Dan'
  echo "Hi, "$name
else
  echo "I don't know who you are!"
endif
```

The concatenate operator "~" allows for the concatention of two strings. This is useful when trying to append the results of one function to another. An example is:

```
set a c:\
set b *.*
dir $a ~ $b
```

• Order of Evaluation

This is the order in which everything is evaluated by the NeoDesk CLI:

1. Variable substitution	(left to right)
2. Function substitution	(right to left)
3. Operators:	(left to right)
A. MULT, DIV	(multiply, divide)
B. +, -	(add, substract)
C. ~	(concatenate)
D. <,<=,>,>=, ==, !=	(less than, less than or equal to, greater than, greater than or equal to, equal to, not equal to)
E. &&, ||	(logical AND, logical OR)

• Single Line Editing

Any text being entered at the CLI prompt or at a GETSTR prompt can be edited using the following editing keys:

The [Backspace] key will delete the last character typed.

The arrow keys can be used to move the cursor within the text being entered.

The [Insert] key will switch between the default overwrite mode and the text insertion mode. The overwrite mode is indicated by the standard block cursor while the insert mode is indicated by the thin vertical line cursor.

If you wish to start again, the [Control][X] and [Clr/Home] keys can be used to clear the input line.

The [Help] key will recall the last non blank line you entered at the CLI prompt.

Functions

These are the functions used by the CLI. They are in alphabetical order. After the name of each function you will find the definition and syntax for that function. The "%" character indicates to the CLI that the text following it is a function. An example is:

%EXAMPLE *parameter*

The actual function appears first in uppercase, though it can be entered in lowercase. Items in italics are to be replaced with the correct *parameter* which is explained after the syntax.

%ALERT

Alert box function. This functions places the specified alert box on the screen and returns the number of the button that the user selects. Its usage is:

%ALERT *alertstring*

An alert box consists of three items: an icon, text, and exit buttons. The *alertstring* should be in the following format:

[*iconnumber*][*textstring*][*buttonstring*]

The square brackets are entered as part of the string. The *iconnumber* is the number of the icon you want displayed inside the alert box. The choices are:

0 - no icon
1 - exclamation mark
2 - question mark
3 - stop sign

The *textstring* is the text to be displayed inside of the alert box. The alert box has a limit of 5 lines of text, each no longer than 30 characters. A vertical bar "|" character represents a new line. The *buttonstring* contains the exit button (or buttons) for the alert box. You can have up to 3 buttons by separating them with the vertical bar "|" character. The buttons in the *buttonstring* cannot be longer than 10 characters each. The leftmost button is always the default button. Examples are:

```
echo %alert "[2][Are you sure that|you want to quit?][Yes|No]"
null %alert "[3][|File corrupted!|Must quit.][Argh!]"
```

Using %ALERT without the correct parameters will result in an error message along with a brief description of how the alert function works. To see this description type:

```
echo %alert
```

%DATE

File date function. This function returns a file's modification date or a folder's creation date. Its usage is:

%DATE *filespec*

The *filespec* is the path and name of the item for which you want the date. If no path is specified then the current CLI path is used. The date is always returned in yy/mm/dd format, which allows for easy date comparisons. A folder will always return its creation date. The item must exist. An example is:

```
set foo c:\neodesk.dat
if %exists $foo
  echo "the modification date of $foo is" %date $foo
endif
```

%DRIVE

Drive function. This function returns the drive of the specified path and/or item. Its usage is:

%DRIVE *filespec*

The *filespec* is the path and/or name to be analyzed by the drive function. If no path is specified then the current CLI path is used. The returned drive includes the colon ":" character. Examples are:

```
echo %drive c:\data\foo.txt
echo %drive data.txt
```

%EXISTS

File exists function. This function returns the constants TRUE or FALSE, depending on whether or not the specified file or folder exists. Its usage is:

%EXISTS *filespec*

The *filespec* is the path and name of the item to be analyzed. If no path is specified then the current CLI path is used. An example would be:

```
set foo c:\neo_cli.acc
if %exists $foo
  echo "You have the NeoDesk CLI installed"
endif
```

%EXTN

File extension function. This function returns the specified item's extension, including the period. Its usage is:

%EXTN *filespec*

The *filespec* is the path and name of the item you want the extension for. An example would be:

```
set foo backup.bat
if %extn $foo == ".bat"
  echo $foo "is a batch file"
endif
```

%FIRST

First file occurrence function. This function searches for the first item that matches the specified path and template. Its usage is:

%FIRST *filespec*

The *filespec* is the path and template which indicates which item (file or folder) to search for. It returns the constant TRUE if there was an an item that matched or the constant FALSE if there was not. If an item was found, its complete path and name is placed in the $ITEM variable. You can combine this with the %NEXT function to get a listing of all the files matching the specified *filespec*. An example would be:

```
echo "enter a path (including template wildcards):"
getstr path $path
if %first $path
  repeat
  echo %fname $item
  until %not %next
else
  echo "error: no files found!"
endif
```

%FNAME

File name function. This function returns just the filename and extension of the specified item (file or folder) without the drive letter or path. Its usage is:

%FNAME *filespec*

The *filespec* is the path and name of the item that you want the filename for. If no path is specified then the current CLI path is used. An example would be:

```
set foo c:\stuff\data.txt
echo "the filename is" %fname $foo
  (would output "data.txt")
```

%HASCH

Has character function. This function returns the constant TRUE if there is a character waiting in the keyboard buffer or FALSE if one is not. Its usage is:

%HASCH

This can be useful in loops which wait for some user input to stop the loop. For example:

```
echo "press any key to terminate \n\O33j"
repeat
  echo "\O33k" %systime
until %hasch
```

This command can also be affected by input redirection, for example:

```
if %hasch <- aux:
  echo "There is a character waiting in the RS-232 buffer."
else
  echo "There is nothing waiting in the RS-232 buffer."
endif
```

%HIDDEN

Hidden file function. This function returns the constant TRUE if the specified item (file or folder) is hidden and FALSE if not. Its usage is:

%HIDDEN *filespec*

The *filespec* is the path and name of the item you want analyzed. Since folders cannot be hidden, they will always respond FALSE with this function. The item must exist. An example would be:

```
set foo \neodesk.dat
if %exists $foo
  if %hidden $foo
    echo "the NeoDesk Volume Name is hidden"
  else
    echo "the NeoDesk Volume Name is not hidden"
  endif
else
  echo "No NeoDesk Volume Name on this drive!"
endif
```

%ISFILE

Is file function. This function returns the constant TRUE if the specified path points to an item that is a file or FALSE if not. Its usage is:

%ISFILE *filespec*

The *filespec* is the path and name of the item to be analyzed. If no path is entered then the current CLI path is used. The item must exist. An example is:

```
echo "A listing of all the files in the root of drive C:\n"
if %first c:\*.*
 repeat
  if %isfile $item
   echo %fname $item
  endif
 until %not %next
endif
```

%ISFOLD

Is folder command. This function returns the constant TRUE if the specified path points to an item that is a folder or FALSE if not. Its usage is:

%ISFOLD *filespec*

The *filespec* is the path and name of the item to be analyzed. If no path is entered then the current CLI path is used. The item must exist. An example is:

```
echo "A listing of all the folders in the root of drive C:\n"
if %first c:\*.*
 repeat
  if %isfold $item
   echo %fname $item
  endif
 until %not %next
endif
```

%NAME

Item name function. This function returns the name of the specified item, that is, the part before the period, without the drive letter or path. Its usage is:

%NAME *filespec*

The *filespec* is the path and name of the item to be analyzed. All the text to the left of the period up to the first backslash is returned, not including the period. An example would be:

```
echo %name c:\neodesk\neomastr.prg
 (would output "neomastr")
```

%NEXT

Next file function. This function works with the %FIRST function and searches for the next item (file or folder) which matches the specification from the previous %FIRST function and places the result in the $ITEM variable. Its usage is:

%NEXT

It returns the constant TRUE if there is another item or FALSE if there is not. See the %FIRST function for more information. An example is:

```
echo "All the items in the current directory:\n"
if %first *.*
 repeat
  echo %fname $item
 until %not %next
endif
```

%NOT

Not function. This function forces a NOT condition in an evaluation, which effectively translates TRUE to FALSE and vice-versa. Its usage is:

%NOT *condition*

The *condition* is the condition to be negated. You should be careful when using this function with operators because operators are evaluated after functions. See the %NEXT command for a good example of this function.

%PATH

File path function. This function returns the specified item's (file or folder) path. Its usage is:

%PATH *filespec*
The *filespec* is the path and name of the item to be analyzed. If no path is specified then the current CLI path is used (and thus returned). The returned path does not include the drive (or its colon). An example would be:

```
echo %path c:\foo\stuff\data.txt
```
  (this would return "\foo\stuff\")

%SIZE

File size function. This function returns the specified item's (file or folder) size in bytes. Its usage is:

%SIZE *filespec*

The *filespec* is the path and name of the item you want to know the size for. If no path is specified then the current CLI path is used. The item must exist. An example is:

```
set foo c:\neodesk\neodesk.exe
if %exists $foo
  echo %size $foo
endif
```

%SYSDATE

System date function. This function returns the current system date. Its usage is:

%SYSDATE

This date comes from the computer's system clock. The date is always returned in yy/mm/dd format. An example is:

```
echo "the current system date is" %sysdate
```

%SYSTIME

System time function. This function returns the current system time. Its usage is:

%SYSTIME

The time comes from the computer's built-in clock. It is returned in military hh:mm:ss format, where the hour is from 00 to 23. An example is:

```
echo "the current system time is" %systime
```

%TIME

File time function. This function returns a file's modification time or a folder's creation time. Its usage is:

%TIME *filespec*

The *filespec* has the path and name of the item you want the time for. If no path is specified then the current CLI path is used. The time is returned in military hh:mm:ss format, where the hour is from 00 to 23. The item must exist. An example is:

```
echo "select a file:"
select "\*.*" "" "select a file:"
if $item != 'FALSE' && %exists $item
  echo "that file's last modification time is" %time $item
endif
```

%UPCASE

Uppercase function. This function returns its argument in uppercase. Its usage is:

%UPCASE *argument*

An example is:

```
repeat
  echo "Enter Q or q to quit"
  getch a
until %upcase $a == 'Q'
```

%WRITE

Write status function. This function returns the constant TRUE if the specified item (file or folder) can be written to or FALSE if not. Its usage is:

%WRITE *filespec*

The *filespec* is the path and name of the item that you want to check the write status of. If no path is specified then the current CLI path is used. The item must exist. An example would be:

```
echo "select a file:"
select "\*.*" "" "select a file:"
if $item != 'FALSE' && %exists $item
  if %write $item
    echo "That file is in read/write mode."
  else
    echo "That file is in read-only mode."
  endif
endif
```

Commands

These are all the CLI commands. They are listed in alphabetical order. The CLI commands are loosely based upon those of the MS-DOS and UNIX™ operating systems.

After the name of each command you will find the definition and syntax for that command. An example is:

EXAMPLE *filespec* [*option1*|*option2*|YES]

The actual command appears first in uppercase, though it can be entered in lowercase. Items in *italics* are to be replaced with the correct option which is explained right after the syntax. Items in uppercase are to be entered as shown, unless specified otherwise. Items surrounded by the "[]" brackets are considered optional and need not be entered. Items separated by the "|" character are multiple choices. For example, the string "[YES|*option*]" indicates that you can optionally either enter a YES or a specified *option*. Unless specified, all options can also be entered in lowercase.

ATTRIB

File attribute command. This command allows you to change the attribute bits of a file (or group of files). Its usage is:

ATTRIB *filespec* [WRITE+|WRITE-] [HIDDEN+|HIDDEN-] [SYSTEM+|SYSTEM-] [ARCHIVE+|ARCHIVE-]

The *filespec* is the path and template of the files whose attribute bits are to be changed. Only the first letter of each attribute is required and they can be in uppercase or lowercase. Each attribute you specify must be followed by a "+" symbol to turn it on or a "-" symbol to turn it off. Unspecified attributes are not changed. The attributes of folders cannot be changed. Wildcards are allowed. Examples are:

```
attrib *.acc w+ s- a- h-
```
(sets the write attribute and clears all the others for all ".acc" files)

```
attrib d:\data\info.txt archive+
```
(turns the archive bit on for file "info.txt")

CAT

Display text file command. This command displays the specified file directly in the CLI window. Its usage is:

CAT *filespec*

The *filespec* is the path and name of the file to be displayed. If no path is specified, then the current CLI path is used. Examples are:

```
cat d:\stuff\info.doc
cat foo.txt -> prn:     (sends the specified file to the printer)
```

The CAT command together with the redirection characters can also be used to copy multiple files into a single file:

```
cat data.txt -> new.txt
cat stuff.txt ->> new.txt
```

CD

Change directory command. This command changes the current CLI path to a new path. Its usage is:

CD [*drive*:][*path*]

Entering the ".." characters as a path element will take you back one directory. Examples are:

```
cd ..        (goes back one directory)
cd \         (goes to the root of the present drive)
cd c:        (goes to the current path of the C drive)
cd c:\       (goes to the root directory of the C drive)
cd d:\stuff  (goes to the specified path and drive)
cd info      (goes to the "info" directory in the present CLI
               path)
cd st\info   (goes to the specified directory in the present CLI
               path)
cd ..\foo    (go back one directory and then into the "foo"
               directory)
```

Entering "CD" by itself will report back the current CLI path, just like the PWD command.

CHMOD

Change file mode command. Same as the ATTRIB command. See ATTRIB.

CLOSE

Close CLI window command. This command will close the NeoDesk CLI window. Its usage is:

CLOSE

The CLOSE command will pause any batch file in progress until the CLI window is reopened. This can be quite useful. An example would be:

```
close
   (close the CLI window)
echo "I am still here!"
   (echo this line the next time the CLI is opened)
```

If the CLI window is CLOSEd within a batch file and opened by running a batch file from NeoDesk, the old batch file is terminated.

COLDBOOT

Coldboot command. This command forces a cold system reset. Its usage is:

COLDBOOT

This is the same effect as turning the computer off and then back on. Reset-proof utilities will not survive this type of reset.

COPY

File copy command. This command copies any items matching the specified path and template from one location to another (files and folders). Its usage is:

COPY *filespec1* [*filespec2*]

The *filespec1* is the source, path, and template of any of the items to be copied. If no path is specified, then the current CLI path is used. The *filespec2* is the destination path and template that they should be copied to. If no *filespec2* is specified, then the current CLI path is used. The template at the end of the *filespec2* is optional and wildcards are allowed. All items which match the specified path and template are copied; if any of the items are folders their contents are copied also. Examples are:

```
copy *.* a:\
   (copies all the items in the current CLI path to drive A)
copy a:\*.*
   (copies all the items in drive A to the current CLI path)
copy info.txt d:\stuff\data.txt
   (copies the file "info.txt" in the current CLI path to the folder
     "stuff" in drive D with the new name of "data.txt")
copy *.doc *.bak
   (copies all the files ending in "*.doc" in the current CLI
     directory to the same file name but with the ending of
     "*.bak")
```

CP

File copy command. Same as the COPY command. See COPY command.

DEBUG

Special debugging tool. This command displays valuable information from the commands entered to assist in debugging. Its usage is:

DEBUG ON|OFF

When the ON option is set, the CLI will display on the screen the actual line as it was read from a batch file. If the command is to be interpreted, it will also display the line number of the batch file and the expanded command line. If not in a batch file, then only the expanded line is displayed. For example, the following batch file:

```
set number 42
echo $number
```

Which normally would result in the following output:

```
42
```

Would result in the following output if run with the debug ON option:

```
>>set number 42<<   (the first line of the batch file)
1: SET•number•42•   (how the CLI interprets that line)
>>echo $number<<    (the second line of the batch file)
2: ECHO•42•         (how the CLI interprets that line)
42                  (the output produced by the CLI)
```

The special "•" character is used to indicate all separators. These can originally be spaces, tabs, or commas.

DEL

File delete command. This command will delete any items matching the specified path and template (files and folders). Its usage is:

DEL *filespec* [FULL]

The *filespec* is the path and template of any item to be deleted. If no path is specified, then the current CLI path is used. Normally it will ONLY delete files but when used with the FULL option it will delete all files AND folders in the specified path (similar to the desktop trashcan). For safety purposes the FULL option must be entered in uppercase. This command deletes all items permanently so it should be used with care. Examples are:

```
del report.txt
   (deletes the file "report.txt" from the current CLI path)
del d:\stuff\*.ps
   (deletes only the files ending in ".ps" in the "stuff" folder in
     drive D, ignoring any folders ending in ".ps")
del d:\stuff\*.ps FULL
   (deletes all the files and folders ending in ".ps")
```

DELETE

File delete command. Same as the DEL command. See DEL.

DIR

Show directory command. This command produces an MS-DOS style directory of the specified path. Its usage is:

DIR [*filespec*]

The *filespec* is the path and template of the items to be displayed. If no path is specified then the current CLI path is used. Examples are:

```
dir
   (displays all the files in the current CLI directory)
dir *.acc
   (displays all the ".acc" files in the current CLI directory)
dir a:\*.prg
   (displays all the files ending in ".prg" in drive A)
```

A sample output would be:

```
   Directory of D:\TEST\

INFO     TXT     1234 01/01/90 12:34p
FOLDER          <DIR> 01/01/90 12:34p
```

The filenames are listed to the left followed by the extensions. Directories are indicated by the "<DIR>" string instead of a file size. These are followed by the date and time the item was last modified or created, whichever is more recent. The output from this command can be redirected.

ECHO

Output echo command. This command displays (echoes) a line of text to the CLI window. Its usage is:

ECHO [NL-] *parameters*

If any *parameters* are present, all the information entered in the same line as the ECHO command is sent to the CLI window. There is a strict limit of 19 *parameters*. Any text included within quotes counts as a single *parameter*. The optional "NL-" (new line off) parameter indicates that the CLI should suppress the carriage return at the end of that line.

Examples are:

```
> echo Hello there!
Hello there!
> echo "Welcome to the NeoDesk CLI"
Welcome to the NeoDesk CLI
> echo A     B
A B
> echo "A     B"
A     B     (the quotes are needed to register the two spaces)
> echo nl- "Hello!"
Hello!>
```

Note that in the last example the prompt is at the end of the output, since the carriage return was suppressed. The "NL-" option is similar to the escape code [Esc][Z] ("\033Z"), but is useful for when you are echoing and output has been redirected to a file or device, in which case the "\033Z" would not be interpreted.

ELSE

Else if command. This command indicates a condition branch in an IF structure. See IF command.

ELSEIF

Else if command. This command indicates a condition branch in an IF structure. See IF command.

ENDIF

End if command. This command terminates an IF structure. See IF command.

ENDWHILE

End while command. This command terminates a WHILE loop. See WHILE command.

ERASE

Erase file command. See DEL command.

EXIT

Exit command. This command terminates and exits a batch file. Its usage is:

```
EXIT
```

If executed in a batch file, the CLI window is restored to its previous state (open or closed) and the mouse is reset back to normal. If the command is typed at the prompt the CLI window is closed.

FONT

Font command. This command specifies which of the three built-in fonts the CLI should use. Its usage is:

```
FONT 0|1|2
```

Font 0 will force the CLI to use the 6x6 icon font.
Font 1 will force the CLI to use the 8x8 color font.
Font 2 will force the CLI to use the 8x16 monochrome font.

FOR

For conditional loop command. This command allows for controlled loops based on a substituted value when inside of a batch file. Its usage is:

FOR [*varname* IN *value1*[..*value2*]]
  ...commands...
ENDFOR

If no arguments are given to the FOR command then the loop will continue infinitely until it hits an EXIT command, an error, or uses the GOTO command to branch out. The ENDFOR command indicates the end of a FOR structure. *Varname* is the variable which is set to the appropriate value by the FOR command. If only *value1* is given then the loop is executed once with with *varname* set to *value1*. If *value1* is followed by two dots and *value2* then the loop is executed once for each value in the series, regardless of direction. If *value1* is a string containing multiple elements separated by standard separators then *varname* will be assigned each element in succession. Values can be either a decimal number or a single

character. Examples are:

```
; this example prints the alphabet backwards
for letter in z..a
  echo $letter
endfor

; this one prints the numbers from -1 to 20
for number in -1..20
  echo $number
endfor

; this one separates the contents of the window variable
set i 1
for element in "$window"
  set wind$i $element
  set i $i + 1
endfor
echo "The window coordinates are:"
echo $wind1 $wind2 $wind3 $wind4
```

GETCH

Get character command. This command will read one character from the keyboard and place it in the specified variable. Its usage is:

GETCH *varname* [*scancode*]

The CLI will pause after this command until the user presses a key. *Varname* is the variable in which the CLI will place the character pressed. The optional *scancode* is the variable in which the CLI will place the actual scan code of that character, which might be needed for processing special keys like the arrow keys. Notice that the "$" character is not used when specifying the name of the variables to be used. An example would be:

```
echo "Please press any letter on the keyboard"
getch letter scan
echo "You pressed the $letter key, with the $scan scan code."
```

This command is affected by input redirection.

GETSTR

Get string command. This command will read an entire string from the keyboard and place it in the specified variable. Its usage is:

GETSTR *varname* [*default*]

The CLI will pause after this command, display a cursor, and let the user enter and edit a line of input until the [Return] key is pressed. *Varname* is the variable in which the CLI will place the resulting string. If a *default* string is entered then it will be placed in the input line with the cursor after the string. The user can then edit that string before pressing [Return]. Notice that the "$" character is not used when specifying the name of the variables to be used. An example would be:

```
set dave "Dave"
echo "Please enter your name:"
getstr name $dave
echo "Hello $name, how are you?"
```

The GETSTR command will only allow up 80 characters to be entered. When its input has been redirected it will read the input until it detects a carriage return with a new line, just a new line, a null character (ASCII 0), or 80 characters.

GOTO

Go to label command. This command causes the execution of a batch file to continue at a specified label in a batch file. Its usage is:

GOTO *labelname*

Labelname is the name of the label that it should branch to. Be careful when using GOTO within nested IF's, FOR's, etc. This could cause undesirable effects. Notice that the semicolon that goes after a labelname is not used with the GOTO command. An example of a batch file using the GOTO command would be:

```
echo "Enter a name:"
getstr name
```

```
if $name == "Dave" || $name == "John"
  goto one
else
  echo "I don't know who you are."
endif
exit
one;    echo "Hello $name, how are you?"
        exit
```

HELP

Help command. Displays the main CLI help screen. If entered with a command name as a parameter, it will display additional help for that command. Its usage is:

HELP [*commandname*]

The *commandname* is the name of any command that you want specific help for. You should note that there is no on-line help for functions. Examples are:

```
HELP
```
(displays help on all the CLI commands, functions, etc.)
```
HELP move
```
(displays additional help on the "move" command)

IF

IF conditional command. This command is used in batch files to analyze conditions and thus decide what additional commands should be executed. Its usage is:

IF *value*
  ...commands...
[ELSEIF *value*]
  ...commands...
[ELSE]
  ...commands...
ENDIF

Technically, the only valid *value* is TRUE or FALSE. These, however, also result from expanded operators and functions. You can have up to 20 nested IF statements. Examples are:

```
if %exists c:\neocntrl.acc
  echo "You are running the NeoDesk Control Panel"
else if %exists c:\control.acc
  echo "You are running Atari's Control Panel"
else
  echo "I don't know which control panel you are using."
endif

echo "press a key"
getch key
if %upcase $key == "A"
  echo "You pressed the [A] key"
else
  echo "You didn't press the [A] key"
endif

set one 10
set two 15
if $one == 10 && $two == 15
  echo "the numbers are right"
endif
```

KICK

Kick drive command. This command forces a media change on the specified drive. Its usage is:

KICK *drivename*

This will force a media change on the specified *drivename*, which is one letter. Any characters to the right of the first character are ignored. This forces the computer to reread the disk as if it was new. It will also force any NeoDesk windows that are open to that drive to be updated. This is the same effect as pressing the [Esc] key on any topped window in NeoDesk.

LOADINF

Load "NeoDesk Information File" command. This asks NeoDesk to load the specified information file. Its usage is:

LOADINF *filespec*

Filespec is the path and name for a "NeoDesk Information File" which matches your version of NeoDesk. If no path is specified then the present CLI path will be used. This command is only available if NeoDesk is the current application. The file must exist. Examples are:

```
loadinf c:\neodesk\new.inf
loadinf data.inf
```

LS

Alternate show directory command. This command supplies a UNIX style file directory. Its options and parameters are identical to those of the DIR command. The only major difference is the output which will also display each item's attributes. A sample output would be:

```
-rwhsa SAMPLE.TXT     12:34:56 01/01/90 1234
drw--- FOLDER         12:34:56 01/01/90 0
```

The item's attributes are displayed to the left of the output. Any attribute that is turned off is indicated by the "-" character. Following the attributes are the item's filename, its time, its date, and its file size. Notice that under the current TOS versions a folder's size is always zero. The format for the item's attributes is as follows:

drwhsa

Where the first character is either a "d" to indicate the item is a directory or a "-" to indicate it is a file. The second character is always an "r" to indicate the item can be read. The third character is a "w" to indicate that the item can be written to or a "-" if not. The fourth character is an "h" to indicate if the file is hidden and a "-" if not. The fifth character is an "s" if the file is a system file and a "-" if not. The sixth character is an "a" if the file's archive bit is set and a "-" if not.

MENU

Menu conditional command. This command is used in batch files to indicate the beginning and end of a MENU structure. When used correctly this can be one of the most powerful batch file commands. Its usage is:

```
MENU
  [menustring]
  [key labelname]
MENU
```

The first MENU command indicates where the menu structure begins. The second MENU command indicates where it ends. The *menustring* is a text line in quotes which is displayed in the CLI window. You can have an unlimited number of *menustrings*. The *key* is the key to be pressed by the user and the *labelname* is the label to GOTO once that *key* has been pressed. There can be up to 10 *key* and *labelname* combinations per menu. An example would be:

```
menu
  "Choose an option"
  "1: Say hello"
  "2: Say Goodbye"
  "Esc : to exit"
  1 sayhi
  2 saybye
  '\O33' quit
menu
sayhi;
  echo "Hello!"
  exit
saybye;  echo "Goodbye!"
quit;    exit
```

Quotes can also be used to signify non-standard characters (see the Special Characters section):

```
' ' quit
  (here a single space is used as a menu option)
'\O33' quit
  (here the escape key {ASCII value 33 octal} is used)
```

Note that the order of the information in between MENU commands does not matter. You can have menu options and strings mixed together. The CLI parses the menu structure by simply seeing if there are one or two parameters on each line (anything in quotes counts as a single parameter). The following example would work fine:

```
menu
  "choose an option"
  " 1: system date"
  1 date
  " 2: system time"
  2 time
menu
date;
  echo "The system date is" %sysdate
  exit
time;
  echo "The system time is" %systime
  exit
```

MKDIR

Make directory command. This command creates a folder with the specified path and name. If no path is given then the present CLI path is used. Its usage is:

MKDIR *path*

Path is the path and name of the folder to be created. Examples are:

```
mkdir data
  (makes a folder called "data" in the current CLI path)
mkdir d:\stuff
  (makes a folder called "stuff" in drive D)
```

MOUSE

Mouse control command. This command will turn the mouse on or off and will also reset it back to its normal state. Its usage is:

MOUSE ON|OFF|RESET

Mouse ON and OFF calls are cumulative. If you turn the mouse OFF 10 times you will have to turn it back on 10 times before the mouse reappears. If you use the RESET option it will go back to normal (mouse on once). The mouse is always reset after executing a program, if a CLOSE command is used, or when a batch file exits.

MOVE

File move command. This command moves any items matching the specified path and template from one location to another (files and folders). Its usage is identical to that of the COPY command, see it for more information.

If the items are moved to a new location in the same drive then the system "move" function is used, which is basically identical to renaming those items to a new path. If the files are moved to a different drive then the items are first copied and then deleted from from the source.

MV

File move command. Same as the MOVE command. See MOVE command.

NULL

Null command. This command does absolutely nothing. Its usage is:

NULL [*parameters*]

This command is useful when it is desirable to call a function and ignore the value returned. In the following example the value returned by the %ALERT function is ignored:

```
null %alert "[1][This is a test][Ok]"
```

PAUSE

Pause command. This command pauses the CLI until a key has been pressed. Use it as an easy way to pause a batch file until a user is ready to continue. It usage is:

PAUSE

The pause command will print "Press any key to continue" when called and then wait for the user to respond.

PRINT

File print command. This command will print the specified files directly to the printer. Its usage is:

PRINT *filespec*

The *filespec* is the path and name of the files to be printed. If no path is specified then the present CLI path is used. It does not use the NeoDesk Printer Queue and is very similar to printing from NeoDesk without the Queue loaded. Wildcards are allowed, but at least one item must exist. Examples are:

print data.txt	(the file "data.txt" is sent to the printer)
print a:\*.doc	(all files in the root of drive A ending in ".doc" are sent to the printer)

PROMPT

Prompt control command. This command allows you to specify the type of prompt that the CLI will use. Its usage is:

PROMPT F[ULL]|B[RIEF]

Only the first letter is required. In FULL mode the CLI displays the current drive and path as a system prompt. In BRIEF mode it just displays the ">" character. Examples are:

prompt full	(sets the CLI prompt to the FULL mode)
prompt b	(sets the CLI prompt to the ">" character)

PWD

Print working directory command. This command prints the present CLI drive and path. Its usage is:

PWD

This command is useful when you are using a BRIEF prompt and you wish to know what the current CLI drive and path are.

QUITNEO

Quit NeoDesk command. It terminates any batch file running, closes the CLI, and tells NeoDesk to quit. Its usage is:

QUITNEO

Once you quit NeoDesk you will be unable to operate the CLI until you load NeoDesk again.

RENAME

File rename command. This command renames the specified items to the new specified path and name (files and folders). Its usage is:

RENAME *filespec1 filespec2*

Filespec1 is the path and name of the items to be renamed. If no path is specified then the present CLI path is used. *Filespec2* is the new path and name that the items should be renamed to. The RENAME command can actually be used as the MOVE command as long as you do not change the drive. Folders can only be renamed with TOS 1.4 or above. Folders cannot be renamed to a different path. Wildcards are allowed. Examples are:

```
rename calc.acc calc.acx
   (renames "calc.acc" to "calc.acx")
rename *.acc *.acx
   (renames all files ending in ".acc" to ".acx")
rename calc.acc \accs\calc.acc
   (moves "calc.acc" to a new directory)
```

REPEAT

Repeat conditional loop command. This command allows for a loop within a batch file until a specified condition is satisfied. Its usage is:

```
REPEAT
   ...commands...
UNTIL value
```

The REPEAT command indicates the beginning of a repeat loop. The UNTIL command indicates the condition that will end the loop. Notice that the loop runs at least once before the UNTIL condition is tested. *Value* should result in one of the constants TRUE or FALSE. An example is:

```
echo "press any key to begin and again to quit \n\O33j"
getch a
set x O
repeat
   set x $x + 1
   echo "\O33k"$x
until $x == 1OOO || %hasch
getch a
```

RM

File remove command. Same as the DELETE command. See DELETE command.

RMDIR

Remove directory command. This command removes the specified empty folders from the disk. Its usage is:

RMDIR *filespec*

The folders must be empty. The *filespec* is the path and name of the folders to be removed. This command does not affect files. Wildcards are allowed. Just like the DEL command, the effects of this command are permanent and it should be used with care. Examples are:

```
rmdir stuff
   (removes the empty folder "stuff" from the current CLI path)
rmdir *.*
   (removes all the empty folders in the current CLI path)
```

To remove folders that are not empty use the REMOVE command with the FULL option.

RN

File rename command. Same as the RENAME command. See the RENAME command.

SELECT

Item Selector command. This command calls the built-in GEM Item Selector. Its usage is:

SELECT *filespec* [*filename* [*comment*]]

The *filespec* is the path and template to be used by the item selector in its directory line. The *filename* is the default filename in the item selector and can be left blank. The *comment* is a string that is placed on the top of the item selector if using TOS 1.4 or greater. Older versions of the operating system do not support the *comment* and it will be ignored. The result from the select call is placed in the $ITEM variable. It is either the path and name of the item selected by the user or the constant FALSE if the Cancel button was selected. An example is:

```
if $tos_ver < 1.4
  null %alert "[1][Choose a file:][Ok]"
endif
select "c:\*.*" "" "Choose a file:"
if $item != 'FALSE' && %fname $item != ""
  echo "You selected file:" $item
endif
```

Notice how the TOS version is checked so that the *comment* can be printed on the screen if the item selector does not support a comment field. Also you must make sure to check that the Cancel button was not selected and that a blank file name wasn't returned.

SET

Variable setting command. This command allows you to either set the value of a specific variable or display the value of one or all of the CLI variables. Its usage is:

SET [*varname*] [*value*]

The *varname* is the name of a variable to be set to the specified value. Please note that the variable names used in the *varname* part of the SET command do not need the "$" character before them. The *value* can contain any number of parameters. Examples are:

```
set name Dave
  (sets the $name variable to contain "Dave")
set time %systime
  (sets the $time variable to contain the system time)
set number 10 MUL 2
  (sets the $number variable to contain the value 20)
```

```
set test 5 > 2
  (sets the $test variable to contain "TRUE")
set foo 1,2,3,4
  (sets the $foo variable to "1 2 3 4")
```

If the SET command is used without a specified *value* it will return the value for that variable. If it is used without any parameters it will display all the CLI variables. Many of these cannot be changed. For example:

```
set name 10
set name
  (this second SET without a value would return "10")
```

The UNSET command can be used to remove any user variables. See UNSET for more information. See the Variables section for a complete listing of all the CLI variables.

SHOW

Show file command. This command displays a file directly to the screen. This command is only available when NeoDesk is the current application. Its usage is:

SHOW *filespec*

The *filespec* is the path and name of the file to be displayed. If no path is specified, then the current CLI path is used. This is the same effect as the "show" command in NeoDesk (when you double-click on a file). See the NeoDesk manual for additional information. Examples are:

```
show data.txt
show d:\stuff\stuff.dat
```

TEXT

Text file entry command. This command allows you to create text or batch files directly from the CLI. Its usage is:

TEXT *filespec*

The *filespec* is the path and name of the text file to be created. If no path is specified, the current CLI path is used. If a file exists with the same name as entered, it will be overwritten. Anything entered after this command is sent to that file. The file is closed when you press [Control][z] (hold down the control key while pressing the "z" key). The standard single line editing controls are available. Wildcards are not allowed.

TOUCH

File touch command. This command changes the specified item's modification date and time to the present system date and time (files only). More commonly known as "touching" a file. Its usage is:

TOUCH *filespec*

The *filespec* is the path and template of the items that you want touched. If no path is entered then the current CLI path is used. Wildcards are allowed. It is not possible to touch a folder. Examples are:

```
touch *.*
   (touches all the files in the current directory)
touch data.txt
   (touches the "data.txt" file)
touch d:\*.doc
   (touches all the ".doc" files in drive D)
```

TYPE

Display file command. Same as the CAT command. See CAT.

UNSET

Unset variable command. This command is used to clear the value of any specific user variable or all the user variables. Its usage is:

UNSET *|*varname*

The *varname* is the name of the variable whose value is to be cleared. If the "*" character is entered then all the user variables are cleared. Examples are:

```
unset *       (clears all the user variables)
unset name    (clears the $name variable)
```

UNTIL

Until conditional loop command. Terminates a REPEAT loop. See REPEAT.

WARMBOOT

Warmboot command. This command forces a warm system reset. Its usage is:

WARMBOOT

This is the same effect as pressing the RESET button on the the Atari ST. Reset-proof utilities can usually survive this type of reset.

WHILE

While conditional loop command. This command allows you to control a loop based on a specified condition. Its usage is:

WHILE *value*
   ...commands...
ENDWHILE

Value must evaluate to one of the constants TRUE or FALSE. The loop will not be executed if the first WHILE is false. An example is:

```
while %not %hasch
   echo 'press a key to quit'
endwhile
```

WINDOW

Window size command. This command allows you to specify a new window size. Its usage is:

WINDOW *xposition yposition width height*

The *xposition* and *yposition* are the upper left-hand coordinates of the CLI window on the screen in pixels. The *width* and *height* are the width and height (in pixels) of the CLI, respectively. Any parameter that is less than 0 is ignored. Parameters that are equal to 0 are set to the smallest possible size. Do not confuse this command with the $WINDOW variable which contains the actual size of the window. Due to limitations in the way GEM handles windows it is possible to use the WINDOW command to make the CLI window smaller than the mouse can resize it. Examples are:

```
window $max_wind   (set the CLI to the largest possible size)
window 0 0 0 0     (set the CLI to its smallest possible size)
window -1 -1 $char_w MULT 20 $char_h MULT 20
    (leave the window at its current location and set it to a
     width and height of 20 characters based on the current
     character size)
```

?

Same as the HELP command. See HELP.

Control-C

Abort command. Aborts any operation in the CLI, including batch files. Hold down the [Control] key while pressing the [C] key.

Control-Q

Resume output. Resumes any output that was paused with the Control-S command. Hold down the [Control] key while pressing the [Q] key.

Control-S

Pause output. Pauses any output in the CLI window until it is resumed with the Control-Q command. Hold down the [Control] key while pressing the [S] key.

Control-W

Close CLI window. This command closes the CLI window. Matches the keyboard equivalent in NeoDesk for closing windows. Hold down the [Control] key while pressing the [W] key.

Control-X

Clear input line. This command clears any text entered into an input line in the CLI (anything you type to the right of a

prompt before you press return). Hold down the [Control] key while pressing the [X] key.

[Help] key

Recall last CLI input. This command restores the last text entered at the CLI prompt.

ASCII Table

This is a listing of all the standard ASCII characters. The extended ST characters (128-255) are not in ths list. It includes their OCTAL value, DECIMAL value, HEX value, actual character, key used to send that ASCII character (for a US keyboard), and the definition of that character (if applicable).

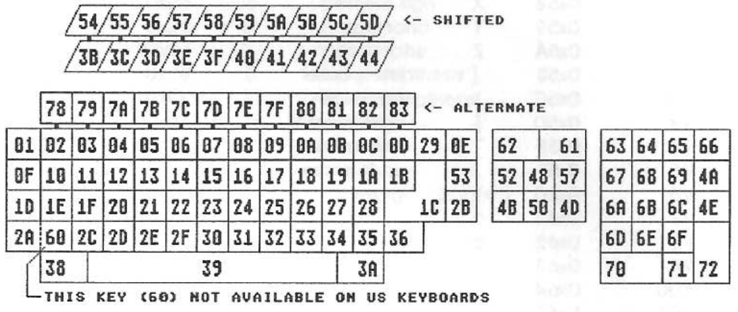
Octal	Decimal	Hex	Char	Key	Definition
000	0	0x00	NUL	ctrl-@	NUL character
001	1	0x01	SOH	ctrl-A	start of header
002	2	0x02	STX	ctrl-B	start of text
003	3	0x03	ETX	ctrl-C	end of text
004	4	0x04	EOT	ctrl-D	end of transmission
005	5	0x05	ENQ	ctrl-E	enquiry
006	6	0x06	ACK	ctrl-F	positive acknowledge
007	7	0x07	BEL	ctrl-G	ring bell
010	8	0x08	BS	ctrl-H	backspace
011	9	0x09	HT	ctrl-I	horizontal tab
012	10	0x0A	LF	ctrl-J	line feed
013	11	0x0B	VT	ctrl-K	vertical tab
014	12	0x0C	FF	ctrl-L	form feed
015	13	0x0D	CR	ctrl-M	carriage return
016	14	0x0E	SO	ctrl-N	shift out
017	15	0x0F	SI	ctrl-O	shift in
020	16	0x10	DLE	ctrl-P	data link escape
021	17	0x11	DC1	ctrl-Q	device control 1 (XON)
022	18	0x12	DC2	ctrl-R	device control 2 (tape on)
023	19	0x13	DC3	ctrl-S	device control 3 (XOFF)
024	20	0x14	DC4	ctrl-T	device control 4 (tape off)
025	21	0x15	NAK	ctrl-U	negative acknowledge
026	22	0x16	SYN	ctrl-V	synchronize
027	23	0x17	ETB	ctrl-W	end of transmission block
030	24	0x18	CAN	ctrl-X	cancel
031	25	0x19	EM	ctrl-Y	end of medium
032	26	0x1A	SUB	ctrl-Z	substitute
033	27	0x1B	ESC	ctrl-[	escape
034	28	0x1C	FS	ctrl-\	form separator
035	29	0x1D	GS	ctrl-]	group separator
036	30	0x1E	RS	ctrl-^	record separator
037	31	0x1F	US	ctrl-_	unit separator

Octal	Decimal	Hex	Char	Key
040	32	0x20	SP	space
041	33	0x21	!	exclamation mark
042	34	0x22	"	quotation mark
043	35	0x23	#	pound sign
044	36	0x24	$	dollar sign
045	37	0x25	%	percent sign
046	38	0x26	&	ampersand
047	39	0x27	'	apostrophe
050	40	0x28	(	left parenthesis
051	41	0x29	)	right parenthesis
052	42	0x2A	*	asterisk
053	43	0x2B	+	plus sign
054	44	0x2C	,	comma
055	45	0x2D	-	minus sign (hyphen)
056	46	0x2E	.	period
057	47	0x2F	/	virgule (slash)
060	48	0x30	0	zero
061	49	0x31	1	one
062	50	0x32	2	two
063	51	0x33	3	three
064	52	0x34	4	four
065	53	0x35	5	five
066	54	0x36	6	six
067	55	0x37	7	seven
070	56	0x38	8	eight
071	57	0x39	9	nine
072	58	0x3A	:	colon
073	59	0x3B	;	semicolon
074	60	0x3C	<	less than
075	61	0x3D	=	equal sign
076	62	0x3E	>	greater than
077	63	0x3F	?	question mark
100	64	0x40	@	at sign
101	65	0x41	A	
102	66	0x42	B	
103	67	0x43	C	
104	68	0x44	D	
105	69	0x45	E	
106	70	0x46	F	
107	71	0x47	G	
110	72	0x48	H	
111	73	0x49	I	
112	74	0x4A	J	
113	75	0x4B	K	
114	76	0x4C	L	
115	77	0x4D	M	
116	78	0x4E	N	
117	79	0x4F	O	
120	80	0x50	P	
121	81	0x51	Q	
122	82	0x52	R	

Octal	Decimal	Hex	Char	Key
123	83	0x53	S	
124	84	0x54	T	
125	85	0x55	U	
126	86	0x56	V	
127	87	0x57	W	
130	88	0x58	X	
131	89	0x59	Y	
132	90	0x5A	Z	
133	91	0x5B	[	left bracket
134	92	0x5C	\	backslash
135	93	0x5D	]	right bracket
136	94	0x5E	^	circumflex
137	95	0x5F	_	underscore
140	96	0x60	`	grave
141	97	0x61	a	
142	98	0x62	b	
143	99	0x63	c	
144	100	0x64	d	
145	101	0x65	e	
146	102	0x66	f	
147	103	0x67	g	
150	104	0x68	h	
151	105	0x69	i	
152	106	0x6A	j	
153	107	0x6B	k	
154	108	0x6C	l	
155	109	0x6D	m	
156	110	0x6E	n	
157	111	0x6F	o	
160	112	0x70	p	
161	113	0x71	q	
162	114	0x72	r	
163	115	0x73	s	
164	116	0x74	t	
165	117	0x75	u	
166	118	0x76	v	
167	119	0x77	w	
170	120	0x78	x	
171	121	0x79	y	
172	122	0x7A	z	
173	123	0x7B	{	left brace
174	124	0x7C	|	vertical bar
175	125	0x7D	}	right brace
176	126	0x7E	~	tilde
177	127	0x7F	DEL	delete character

Scan Codes

These are the scan codes returned by the Atari keyboard, in hexadecimal. Note that some keys produce different scan codes when used with the [Shift] or [Alternate] keys.


54 55 56 57 58 59 5A 5B 5C 5D <- SHIFTED
3B 3C 3D 3E 3F 40 41 42 43 44
78 79 7A 7B 7C 7D 7E 7F 80 81 82 83 <- ALTERNATE
01 02 03 04 05 06 07 08 09 0A 0B 0C 0D 29 0E 62 61 63 64 65 66
0F 10 11 12 13 14 15 16 17 18 19 1A 1B 53 52 48 57 67 68 69 4A
1D 1E 1F 20 21 22 23 24 25 26 27 28 1C 2B 4B 50 4D 6A 6B 6C 4E
2A 60 2C 2D 2E 2F 30 31 32 33 34 35 36 6D 6E 6F
38 39 3A 70 71 72
THIS KEY (60) NOT AVAILABLE ON US KEYBOARDS

This codes can best be used with the GETCH command, which can return the scan code of the key selected. An example would be:

```
echo "Press [Help] then [Undo] to quit."
top:
getch a b
if "$b" == '\x62'
  echo "You selected the [Help] key."
elseif "$b" == '\x61'
  echo "You selected the [Undo] key, exiting."
  exit
endif
goto top
```

NeoDesk Developer's Kit

The NeoDesk CLI was created using the *NeoDesk Developer's Kit*. It allows you to write special *NeoDesk Accessories* which hook directly into NeoDesk itself. These accessories can be used to add whole new features to NeoDesk or to simply make use of its functions. With versions of NeoDesk later than 2.05 you can even run these accessories as separate *NeoDesk Programs*.

The kit includes all the documentation, sample code, "include" files, and other information needed to write your own *NeoDesk Accessories* and *NeoDesk Programs*. It is designed for use with either the C language or 68000 assembly. Knowledge and experience in the writing of desk accessories and the use of pointers is recommended.

The kit is available for $19.95. In order to keep its price down, this kit is only available directly from us. To place an order send us a note with the following information and items:

1. Your name and address.
2. Your NeoDesk serial number.
3. The correct amount of money in U.S. funds (see chart):

NeoDesk Developer's Kit	$19.95
Shipping and Handling	$ 3.00
If ordering from outside the U.S. add the following to cover additional shipping and handling costs	$ 2.00
If in Mass. add 5% sales tax	$ 1.00
Total	$_____

4. A daytime phone number.

If you wish to order by credit card (MasterCard or Visa) be sure to include your credit card number, expiration date, name as it appears on the card (print neatly!), your signature, and your NeoDesk serial number. Send the order to the following address:

Attn: Dev Kit Order
Gribnif Software
Post Office Box 350
Hadley, MA 01035

Please allow up to 3 to 4 weeks for delivery. If you have any questions you can call us at (413) 584-7887.

Notes