Update to GRiDTaskBuilder & GRiDTask II User's Guide

and GRiDTask User's Guide

October 1986

The replacement pages that follow reflect changes to the **November** 1985 editions of the *GRiDTaskBuilder & GRiDTask II User's Guide* and the *GRiDTask User's Guide*. Replace the pages in these manuals with the pages in this update according to the following guide.

GRiDTaskBuilder & GRiDTask II User's Guide:

Add update page 9 to the end of the manual.

GRiDTask User's Guide:

Discard these Existing pages	**Replace with these Update Pages**
Table of Contents	Table of Contents
4-1/4-2	4-1/4-2, 4-2a/blank
4-11/4-12	4-11/4-11a, 4-12/blank
4-73/4-74	4-73/4-73a, 4-74/blank
4-87/4-88	4-87/4-87a, 4-88/blank
4-91/4-92	4-91/4-91a, 4-91b/4-91c, 4-92/blank
4-95/4-96	4-95/4-96
A-1/A-2	A-1/A-2
A-3/A-4	A-3/A-4
E-1/E-2	E-1/E-2
H-7/H-8	H-7/H-8
I-4/I-4a	I-4/I-4a
I-21/I-22	I-21/I-22
I-23/I-24	I-23/I-23a, I-24/blank
I-31	I-31

Order Number: 021250-45

Using Overlays You can use overlays in GRiDTask II to reduce the amount of RAM that is used by the program. With overlays, only part of the program is loaded into memory at a time.

GRiDTask II overlays are very easy to use. GRiDTask II does all the work for you, automatically loading the correct overlay whenever it is needed. The only thing you need to do is specify which source files go into which overlays when you invoke GRiDTaskBuilder. You do this by inserting the word "overlay" into the command line used to invoke GRiDTaskBuilder. For example:

```
GRiDTaskBuilder mainFile, part1, part2 OVERLAY part3, part4
OVERLAY part5
```

This invocation of GRiDTaskBuilder creates a root section that remains loaded in RAM when the program is executing, and two overlays. The root section consists of the statements and procedures defined in the files "mainFile", "part1", and "part2". The first overlay contains the procedures defined in the files "part3" and "part4". The second overlay contains the procedures defined in the file "part5".

When you execute a program, the root section and the first overlay are automatically loaded into RAM. Besides the root section, only the code in one overlay is loaded into RAM at one time. If a procedure is called or returned to in an overlay that is not currently loaded, GRiDTask II will free the memory occupied by the currently loaded overlay, and load the necessary overlay into RAM.

Note that there is a slight delay every time GRiDTask II loads another overlay since it must read from the program file on a permanent storage device.

You can use a maximum of 255 overlays in a GRiDTask II program.

TECHNICAL NOTE: The RAM used for GRiDTask overlays is not a static preallocated block as in Pascal overlays. Instead, the RAM required for each procedure in an overlay is allocated separately from the heap memory manager. The advantage of this method is that when a small overlay is loaded, more RAM is available for other purposes. The disadvantage is that you cannot know exactly how RAM is used at any point in time.

GRiDTask Manual
Table of Contents

Section One - GRiDTask VERBS

This chapter contains detailed descriptions of the verbs of the GRiDTask language, including functions, procedures and predefined variables. Note that this manual uses the terms "verbs" and "statements". GRiDTask verbs are the commands themselves, such as "PRINT", and a statement is one line in a GRiDTask application that uses a GRIDTask verb or verbs. The chapter is arranged alphabetically and there are some conventions used, as follows:

There is a group of GRiDTask verbs used to perform mathematical operations. These are placed in a section entitled "GRiDTask Real Number Functions" at the end of this chapter.

o GRiDTask verbs appear in capital letters.

e.g. APPENDFILE

o All variable names appear in lowercase letters. If a variable name consists of two or more words, then words after the first one may be capitalized.

e.g. apples itemNumber item$

Special Notes:

o You can continue a GRiDTask statement on a new line by entering an underscore character (_) as the last character in the line. (press RETURN to type the rest of the statement)

o Multiple GRiDTask statements can be placed on one line by separating them with a colon.

o You can place GRiDWrite text formatting commands (e.g., ^ep, ^nl, ^sl, etc.) in a GRiDTask program. GRiDTask ignores lines with a circumflex (^) as the first character.

o Many of the examples in this chapter are shown out of context. As such, they may not run exactly as shown. Also, some of the examples have not been tested.

o Appendix A is a quick-reference list of the verbs described in this chapter (4).

EXAMPLES

```
FILEFORM "`Bubble Memory`Memos`Call Summary~Text~"
ADDKEYS "¦."             ; Confirm the File form
ADDKEYS "¦e¦V¦.¦t¦."
```

This example retrieves a text file, erases its contents, and then saves the file. The first ADDKEYS statement confirms the File form. The second ADDKEYS statement is equivalent to pressing:

```
     CODE-E                    " ¦e "
     CODE-SHIFT-DownArrow      " ¦V "
     Confirm                   " ¦. "
     CODE-T                    " ¦t "
     Confirm                   " ¦. "
```

CHANGEKIND$

newPathName$ = CHANGEKIND$ (pathName$, Kind$)

NOTES

CHANGEKIND$ is a string function requiring two string parameters. The first parameter is a file pathname and the second is a file Kind. CHANGEKIND$ creates a new string which is the same as PathName$ except with the new Kind.

EXAMPLE

```
;-----------------------------------------
; Reformat Historical Quotes Text File
;-----------------------------------------
textFile$ = GETFILE$("Select Text file to be reformatted")
FILEFORM "Historical Quotes~Reformat~00"
ADDKEYS "|.|t|."
FILEFORM textFile$
ADDKEYS "|."
graphFile$ = CHANGEKIND$(textFile$, "Graph")
FILEFORM graphFile$ + "21"       ; get new file and application
ADDKEYS "|.|."
```

This program reformats a text file of data that has been retrieved from a mainframe. It writes the reformatted data to a graph file.

1) It starts by asking the user to fill in a File form selecting the text file to be reformatted.

```
textFile$ = GETFILE$("Select Text file to be reformatted")
```

2) It then retrieves a Reformat file.

```
FILEFORM "Historical Quotes~Reformat~00"
ADDKEYS "|.|t|."
```

3) It specifies the text file as the file to be reformatted.

```
FILEFORM textFile$
ADDKEYS "|."
```

4) It specifies the output file as having the same name as the input text file except with a kind of "Graph". It writes the new graphfile, then brings it into GRiDPlot.

```
graphFile$ = CHANGEKIND$(textFile$, "Graph")
FILEFORM graphFile$ + "21"       ; get new file and application
ADDKEYS "|.|."
```

CHARHEIGHT

```
height = CHARHEIGHT
```

NOTES

CHARHEIGHT is a function which returns the height (in pixels) of the capital letters in the current font, plus one for the descenders. The current font is the last font set with the FONT verb.

LINEHEIGHT is a similar function which returns the height in pixels of a line of text. Note that CHARHEIGHT returns just the size of the characters and is usually one or two pixels less than LINEHEIGHT.

EXAMPLE

```
TASKWINDOW      0,0,-1,-1
PRINT "The characters in this font are exactly "
PRINT  STR$(CHARHEIGHT) + " high"
```

This example gives the character height of the current font.

CHARWIDTH

```
width = CHARWIDTH
```

NOTES

CHARWIDTH is an integer-value function which returns the width of the current font in the Task window. The width is measured in pixels.

EXAMPLE

```
PRINT "This is your first message"
DELAY 2
PRINT "Your first message used 26 characters,"
PRINT "so it is " + STR$(26 * CHARWIDTH) + " pixels wide"
```

In this example, the width of the first message (printed in the current font) is calculated using CHARWIDTH.

MID$

```
portion$ = MID$ (wholeString$, start, length)
```

NOTES

MID$ is a string function which returns a portion of a specified string.

The first parameter is the string from which the portion is extracted. The second parameter is the character position at which to start the new portion string. The third parameter is the length of the portion string.

MID$ returns a zero-length string if _length_ is zero, or if _start_ is either zero or greater than the length of the string.

If _start_ + _length_ is greater than the length of the original string, then MID$ returns a string which only includes characters from _start_ to the end of _wholeString$_.

EXAMPLE

```
found = 0
i     = 0
WHILE i < LEN(inputstringX$)
  i = i + 1
  IF MID$(inputstringX$, i, 1) = "?"
    found = found + 1
  ENDIF
WEND
PRINT "I found " + STR$(found) + " question marks!"
```

This example counts the number of question marks in a string (inputstringX$). It prints a message indicating how many were found.

MSGHEIGHT

```
height = MSGHEIGHT
```

NOTES

MSGHEIGHT is a function which returns the height (in pixels) of messages in the current font. The current font is the last font set with the FONT verb.

MSGHEIGHT is the maximum of the following two values: LINEHEIGHT or CHARHEIGHT + 2.

EXAMPLE

```
TASKWINDOW 0,0,-1,-1
STACKMSG "This is the last message line"
STACKMSG "This is the second message line"
STACKMSG "This is the first message line"
HeightLeft = WINDOWHEIGHT - MSGHEIGHT * 3
PRINT "The available height of the window is "+STR$(HeightLeft)+"
pixels."
PAUSE ""
```

This example prints the available height of the Task window after three messages are stacked.

PAINT

```
PAINT x, y, "pathname"
```

NOTES

PAINT displays a canvas image in the Task window. Canvas files can be created and modified in GRiDPaint.

The parameters indicate the name of the canvas file to be displayed and the pixel coordinates within the Task window where the top-left corner of the canvas image is to be placed.

Any portion of the canvas image extending beyond the edge of the window is clipped.

It is important that the image be created and saved using GRiDPaint, as a file with Kind "Canvas". Screenimage files do not work.

If the pathname has no Kind, then "Canvas" is assumed. If no Device or Subject is specified, then GRiDTask looks in the current Device and Subject.

EXAMPLE

```
TASKWINDOW 0,0,-1,-1
PAINT 10,10, "Renoir~Canvas~"
```

When GRiDTask executes this, the image with Title "Renoir" and Kind "Canvas" in the current Device and Subject is displayed on the screen. The upper left corner of the Canvas image is placed 10 pixels from the left edge of the Task window and 10 pixels down from the top edge of the Task window. GRiDTask displays as much of the image as there is room for.

READFILE$

```
contents$ = READFILE$ (pathname$)
```

NOTES

READFILE$ is a string function which returns the contents of the file specified by pathname$. The file specified by pathname$ remains unchanged.

Note that if you do not specify the Kind in pathname$, then the Kind Text is assumed. If you don't specify a Device or Subject, then the current Device and Subject of the last file accessed through GRiDTask or the application window are assumed.

The maximum length allowed for a string variable is 64K bytes, so that if you attempt to read a file larger than 64K bytes, a GRiDTask error occurs.

READFILE$ sets the ERRORCODE variable to the number of any error that occurred. If no error occurs, then the ERRORCODE variable is set to (0) zero.

EXAMPLE

```
pathname$   =  "`Floppy Disk`BaseballCards`MickeyMantle~Text~"
statistics$ =         READFILE$ (pathname$)
```

The above example copies the contents of the file MickeyMantle into the string variable statistics$.

REMOVELIB

```
REMOVELIB library$
```

NOTES

REMOVELIB lets you de-install libraries of custom GRiDTask verbs while your program is executing; this frees memory associated with the library when you no longer need the library's functions. All installed libraries are automatically removed when a GRiDTask program exits.

Removing a library which has not been installed has no effect. See the INSTALL verb for more information on installing libraries.

NOTE: REMOVELIB is only available under GRiDTask II.

EXAMPLE

```
INSTALL "Sample~Library~"
.
.
.

REMOVELIB "Sample~Library~"
```

This example de-installs the library Sample. The custom functions provided by the library Sample are no longer available to the GRiDTask program.

RETURN

```
RETURN
```

NOTES

RETURN is used within procedures. When executed, GRiDTask exits from the procedure, and returns to the GRiDTask statement following the procedure call.

RETURN(s) are optional. If used, there may be more than one RETURN within a procedure, and RETURN verbs may be placed anywhere within the procedure body. A RETURN is not needed at the physical end of a procedure.

EXAMPLE

See the section in Chapter 4 entitled "Procedures" for an example using RETURN.

the Task window for ten seconds. Then it charges to the right across the Task window.

SETFORM

```
SETFORM form$
```

NOTES

SETFORM allows you to preset the results of forms that are to be displayed by an application in the application window. If a form is preset by SETFORM, then when the application tries to display the next form, GRiDTask intervenes and automatically causes the form to be confirmed with the preset values. The form is not actually displayed in the application window.

The form$ string uses two special characters:

tildes	~
vertical bars	¦

Tildes separate the three parts of an item setting: the item number, the choice number, and the choice text. Vertical bars separate form items. For example, assume that the next form to be displayed in the application window is the GRiDWrite Print Options form; the string--

```
printForm$ = "1~4~all~¦2~1~Heading~¦7~4~boldface~¦"
```

--causes the first item of the form to be set to the fourth choice which has the text "all". The second item of the form is set to the first choice with the text "Heading". The seventh item of the form is set to the fourth choice with the text "boldface". The form is automatically confirmed and is not displayed.

Always specify both the choice number and the choice text for an item, even though this seems redundant for non-editable items.

It's important to note that SETFORM must be executed before a form is displayed; therefore, place the SETFORM verb before the ADDKEYS verb that causes the form to be displayed.

Sometimes, several forms follow in immediate succession. You can preset up to five forms for later processing. To clear the queue of SETFORM verbs waiting to be processed, use the statement SETFORM "0"

SETFORM cannot be used with File forms. See the FILEFORM verb for more information.

TECHNICAL NOTE: In order for SETFORM to operate correctly, the application must call the GRiD Common Code function DataFormConfirmed. The only known GRiD applications where SETFORM won't work are GRiDDevelop, GRiDReformat, and GRiDFile, where it won't work with the Sort form.

EXAMPLE

```
FILEFORM "`Hard disk`Memos`Sample~text~"
ADDKEYS "¦."
SETMENU 9
SETMENU 2
SETFORM "1~4~all~¦2~1~Heading~¦7~4~boldface~¦"
ADDKEYS "¦t"
```

In this example, a text file is selected and confirmed in the application window; the Print item on the Transfer menu is selected; the Print Options item on the Print menu is selected; and then the print options are set by the SETFORM verb. Note that the ADDKEYS statement which sets the whole process in motion is placed after the menus and the form are preset.

SETMENU

SETMENU item

NOTES

SETMENU allows you to preset the results of menus that are to be displayed by an application in the application window. If a menu is preset by SETMENU, then when the application tries to display the next menu, GRiDTask intervenes and automatically causes the menu to be confirmed with the selected item. item is the item number that you want to select. (Menu items are numbered from the top down, starting at 1.) The menu is not actually displayed in the application window.

It's important to note that SETMENU must be executed before a menu is displayed; therefore, place the SETMENU verb before the ADDKEYS verb that causes the menu to be displayed.

Sometimes, several menus follow in immediate succesion. You can preset up to five menus for later processing. To clear the queue of SETMENU verbs waiting to be processed, use the statement SETMENU 0.

Using SETMENU has several potential advantages over using ADDKEYS to select and confirm an item in a menu. SETMENU is much faster, easier to use, and it prevents the menu from displaying.

TECHNICAL NOTE: In order for SETMENU to operate correctly, the application must call the GRiD Common Code function DataMenuConfirmed. All GRiD applications currently support SETMENU.

EXAMPLE

See the example for SETFORM.

SPEED

```
SPEED "str"
```

NOTES

SPEED controls how fast the keys specified by the ADDKEYS verb are fed to the application. SPEED also controls how fast characters are displayed by the CENTER and PRINT verbs.

The parameter string can be "Fast", "Medium" or "Slow". "Fast" represents no delay between characters, "medium" is 0.2 seconds delay and "slow" is 0.5 seconds delay between characters.

The parameter string can also represent the number of milliseconds delay between characters. To specify a 0.1 second delay between characters you would use the following statement.

```
SPEED "100"
```

The initial setting is "Fast".

Note that some programs, such as terminal emulators connected to hosts, may not accept keys at the fast rate.

EXAMPLE

```
SPEED "Fast"
PRINT "I am an Olympic typist"
SPEED "Medium"
PRINT "I have pudgy fingers"
SPEED "Slow"
PRINT "I have boxing gloves on"
```

In this example, the first message is printed on the screen with no delay between characters. The second message is printed on-screen with a .2 second delay between characters, and the third message is printed on-screen with one-half second delay between characters.

STOP

STOP mode

NOTES

STOP causes GRiDTask to stop running. The application window is returned to its original size.

The other condition causing GRiDTask to stop running is when it reaches the end of its main program file.

The optional parameter mode can be used to cause GRiDTask to reboot your computer when the STOP statement is executed. mode can be specified as follows:

1 Causes the system to reboot as if the power switch were turned off and then on

2 Causes the system to warm reboot (as if CODE-CTRL-SHIFT-HYPHEN were pressed)

These two modes of rebooting are identical under GRiD-OS; under InteGRiD, the second method exits InteGRiD and returns to MS-DOS.

You can use this optional parameter in an unattended GRiDTask program which downloads new versions of software and then automatically reboots the system.

EXAMPLE

```
mainMenu$ = "Status Reports|Mail|Exit"
msg$      = "Select activity and Confirm"

TASKWINDOW 0,0,-1,-1
WHILE TRUE
  choice = DOMENU (msg$, mainMenu$)
  IF choice = 1
    TASK "status"
  ELSE
  IF choice = 2
    TASK "mail"
  ELSE
  IF choice = 3
    STOP
  ENDIF: ENDIF: ENDIF
WEND
```

This example represents the main body of a Task program. It displays a menu with three items. If the third item, "Exit", is selected, GRiDTask executes a STOP statement and stops.

STR$

```
num1$ = STR$ (num)

        OR

num2$ = STR$ (num,precision)
```

NOTES

STR$ is a string function which converts a number to a string of decimal characters.

STR$ can have one or two parameters. The first parameter is the number to be converted. The optional second parameter indicates how many digits after the decimal point to display. If this second parameter is omitted, then STR$ returns a string containing the minimum number of characters required to precisely represent the number. See the examples.

To convert a string of digits to a decimal number, use the VAL function.

EXAMPLES

```
STR$ (9)          =>        "9"
STR$ (9/8)        =>        "1.125"
STR$ (9/8,0)      =>        "1"
STR$ (9/8,2)      =>        "1.13"
STR$ (9/8,6)      =>        "1.125000"
```

The result of each STR$(..) is shown above.

Appendix A

GRiDTask Verb Summary

GENERAL PURPOSE VERBS

ADDKEYS	ADDKEYS "encodedKeyStr", mode
APPENDFILE	APPENDFILE addString$, pathname$
ASC	num = ASC (anyString$)
BREAK	BREAK
BREAKONKEY	BREAKONKEY key$
BREAKRESET	BREAKRESET
CELL$	contents$ = CELL$
CENTER	CENTER "text....."
CHANGEKIND$	newPathName$ = CHANGEKIND$ (pathName$, kind$)
CHARHEIGHT	height = CHARHEIGHT
CHARWIDTH	width = CHARWIDTH
CHR$	stringX$ = CHR$ (num)
CLEARMSG	CLEARMSG
CLEARSCREEN	CLEARSCREEN
COMMANDLINE	COMMANDLINE command$, secondsDelay
COMMENT	; Place text here
CONCHARIN$	ch$ = CONCHARIN$
COPYFILE	COPYFILE sourcePath$, destinationPath$
CURSOR	CURSOR x, y
CURX , CURY	CURSOR CURX + 5, CURY - 10
DATE$	today$ = DATE$
DELAY	DELAY seconds
DEVICE$	dev$ = DEVICE$
DIRECTORY$	list$ = DIRECTORY$ (mode, path$, match$, delimiter$, sortOrder)
DO	DO taskStatements$
DOFORM$	form$ = DOFORM$ (msg$,form$,numLines)
DOMENU	choice = DOMENU (msg$, item$)
ELSE	ELSE
ENDIF	ENDIF
ENDP	ENDP
ERASEBOX	ERASEBOX topleftX, topleftY, extentX, extentY
ERASEFILE	ERASEFILE pathname$
ERRORCODE	ERRORCODE = number <u>or</u> number = ERRORCODE
ERRORSTR$	err$ = ERRORSTR$(errorNum)
FALSE	variable = FALSE
FILEFORM	FILEFORM "pathname"
FINDTITLE$	path$ = FINDTITLE$ ("Title~Kind~")
FONT	FONT "fontPathName"
FORMCHOICE	number = FORMCHOICE (form$, itemNumber)
FORMCHOICE$	choice$ = FORMCHOICE$ (form$, itemNumber)
FRAMEBOX	FRAMEBOX topleftX, topleftY, extentX, extentY
FREEFONT	FREEFONT "fontPathName"

GETFILE$	pathName$ = GETFILE$ (msg$)
IF/ELSE/ENDIF	IF <exp> / stmts / ELSE / stmts / ENDIF
INKEY$	someKey$ = INKEY$
INPUT$	value$ = INPUT$(prompt$, length, height, initValue$)
INSTALL	INSTALL pathname$
INSTR	location = INSTR (start, source$, find$)
INVERTBOX	INVERTBOX topleftX, topleftY, extentX, extentY
INVERTLINE	INVERTLINE X1, Y1, X2, Y2
ITEMCOUNT	numItems = ITEMCOUNT (list$, separater$)
LASTKEY$	key$ = LASTKEY$ or LASTKEY$ = stringX$
LASTMESSAGE$	LASTMESSAGE$ = message$ or message$ = LASTMESSAGE$
LEN	num = LEN (stringX$)
LINEHEIGHT	height = LINEHEIGHT
LOCATE	LOCATE x, y
MEMORY	space = MEMORY
MID$	portion$ = MID$ (wholeString$, start, length)
MSGHEIGHT	height = MSGHEIGHT
PAINT	PAINT x, y, "pathname"
$PARSEONLY	$PARSEONLY
PASSKEYS	PASSKEYS keysToPass$, keysToTerminate$
PAUSE	PAUSE keysToTerminate$
PLAY	PLAY musicStr$
PRINT	PRINT "text...."
PROCEDURE	PROCEDURE procedureName param1, param2$, ...
READFILE$	contents$ = READFILE$ (pathname$)
REMOVELIB	REMOVELIB library$
RETURN	RETURN
SCROLL	SCROLL distance, speed
SCROLLBOX	SCROLLBOX topleftX, topleftY, extentx, extenty, "direction", distance, speed
SETFORM	SETFORM form$
SETMENU	SETMENU item
SPEED	SPEED "str"
STACKMSG	STACKMSG "messageText"
STACKSIZE	stackSpace = STACKSIZE
STOP	STOP mode
STR$	num1$ = STR$(num) or num2$ = STR$(num,precision)
SUBJECT$	sub$ = SUBJECT$
SUBSTITUTE$	newStr$ = SUBSTITUTE$ (oldStr$, find$, replaceWith$)
SUBSTRING$	sub$ = SUBSTRING$ (source$, delimiter$, itemNumber)
TASK	TASK "pathname"
TASKWINDOW	TASKWINDOW topleftX, topleftY, extentx, extenty
TESTKEYS	TESTKEYS "encodedKeyStr"
TIME$	clock$ = TIME$
TITLE	TITLE "title text..."
TRUE	variable = TRUE
UPDATESCREEN	UPDATESCREEN

VAL	num = VAL (stringX$)
WEND	WEND
WHILE	WHILE <exp> / stmts / WEND
WINDOWHEIGHT	size = WINDOWHEIGHT
WINDOWMOTION	WINDOWMOTION "ON" <u>or</u> "OFF"
WINDOWWIDTH	width = WINDOWWIDTH
WRITEFILE	WRITEFILE information$, pathname$

MATHEMATICAL FUNCTIONS

ACOS	Arc Cosine	ACOS(number)
ATN	Arc Tangent	ATN(number)
COS	Cosine	COS(angle)
EXP	Exponential	EXP(exponent)
LOG	Natural Logarithm	LOG(number)
LOG10	Base 10 Logarithm	LOG10(number)
PI	The constant Pi	PI
RND	Random number	RND(1)
ROUND	Round to an integer	ROUND(number)
SIN	Sine	SIN(angle)
SQR	Square Root	SQR(number)
TAN	Tangent	TAN(number)
TRUNC	Truncate to an integer	TRUNC(number)

VERBS INSTALLED IN DataEntryForms LIBRARY (SEE APPENDIX I)

DISPOSEFORM	DISPOSEFORM formNum
EDITFORM$	key$ = EDITFORM$ (formNum, topLeftX, topLeftY, widthX, heightY, mode)
FORMINIT	formNum = FORMINIT (formStr$)
FORMINITFROMFILE	formNum = FORMINITFROMFILE (pathName$)
GETALLFIELDS$	values$ = GETALLFIELDS$ (formNum, delimiter$)
GETCURRENTFIELD	currentField = GETCURRENTFIELD (formNum)
GETFIELDVALUE$	value$ = GETFIELDVALUE$ (formNum, currentField)
INDEXFROMNAME	fieldIndex = INDEXFROMNAME (formNum, name$)
NAMEFROMINDEX$	name$ = NAMEFROMINDEX$ (formNum, currentField)
PARSEFORM$	parsedSpec$ = PARESEFORM$ (fileSpec$)
PRINTFORM	error = PRINTFORM (formNum, printMode, destination$,topMargin, bottomMargin,leftMargin, printSize, formFeed)
SETALLFIELDS	SETALLFIELDS formNum, values$, delimiter$
SETCURRENTFIELD	SETCURRENTFIELD formNum, currentField
SETFIELDVALUE	SETFIELDVALUE formNum, fieldIndex, newValue$

Appendix E

Reserved Words

The names in this list are reserved for GRiD's use. They should not be used as variable names in your GRiDTask programs. Included in this list are the functions and variables described in this manual, as well as names which are reserved for future use by GRiD. With few exceptions, if you try to use any of these names as variable names with either upper or lower-case letters, an error may occur.

ABS
ACOS
ADDKEYS
APPENDFILE
ASC
ASIN
ATN

BREAK
BREAKONKEY
BREAKRESET

CALL
CDBL
CELL$
CENTER
CHANGEKIND$
CHARHEIGHT
CHARWIDTH
CHR$
CINT
CLEARMSG
CLEARSCREEN
COMMANDLINE
CONCHARIN$
COPYFILE
COS
CSNG
CURSOR
CURX
CURY
CVD
CVS

DATE$
DELAY
DEVICE$
DIRECTORY$
DO
DOFORM
DOFORM$
DOMENU

ELSE
ENDIF
ENDP
EOF
EOLN
ERASEBOX
ERASEFILE
ERRORCODE
ERRORSTR$
EXP
EXIT

FALSE
FFPATCHOFF
FFPATCHON
FILEFORM
FINDTITLE$
FIRSTPAGE
FIX
FONT
FORMCHOICE
FORMCHOICE$
FRAMEBOX
FREEFONT

GETFILE$
GETPREFIX$

HEX$
HEXVAL

IF
IMP
INKEY$
INPUT$
INSTALL
INSTR
INT
INVERTBOX
INVERTLINE
ITEMCOUNT

LASTKEY$
LASTMESSAGE$
LEFT$
LEN
LINEHEIGHT
LOC
LOCATE
LOF
LOG
LOG10
LPRINT

MEMORY
MID$
MKD$
MKI$
MKS$
MSGHEIGHT

OCT$
OCTVAL

PAGE
PAINT
$PARSEONLY
PASSKEYS
PAUSE
PEEK
PI
PLAY
POKE
POS
PRINT
PROCEDURE

READFILE$
REMOVELIB
RETURN
RIGHT$
RND
ROUND

SCREENIMAGE
SCROLL
SCROLLBOX
SETFORM
SETMENU
SGN
SHAPE
SIN
SPACE$
SPEED
SQAR
SQR
STACKMSG
STACKSIZE
STOP
STR$
STRING$
SUBJECT$
SUBSTITUTE$
SUBSTRING$

TAB
TAN
TASK
TASKWINDOW
TESTKEYS
TIME$
TITLE
TRUE
TRUNC

UPDATESCREEN

VAL

WEND
WHILE
WINDOWHEIGHT
WINDOWMOTION
WINDOWWIDTH
WINDOWX
WINDOWY
WRITEFILE

Words Reserved When Using Data-Entry Forms Libraries

DISPOSEFORM

EDITFORM$

FORMINIT
FORMINITFROMFILE

GETALLFIELDS$
GETCURRENTFIELD
GETFIELDVALUE$

INDEXFROMNAME

NAMEFROMINDEX$

PARSEFORM$
PRINTFORM

SETALLFIELDS
SETCURRENTFIELD
SETFIELDVALUE

Notes on Custom Routines

The RegisterLibraryFunctions procedure (in Sample.Pas) uses the RegisterFunction procedure to:

1) supply the verb name to be used in GRiDTask.

2) supply the function or procedure address to GRiDTask.

3) supply the type of function or procedure. This is determined by the type of value returned. Possibilities are

* statement - no value is passed back
* string - a string value is passed back
* Integer - an integer value is passed back
* LongReal - a real value is passed back
* Boolean - a boolean value is passed back

4) supply the type and number of parameters passed from the installed verb statement in GRiDTask to the custom routine. The possibilities are:

Specification	Parameter Type
1) none	
2) "i"	integer
3) "s"	string
4) "r"	real

Any combination of "i", "s", "r" is allowed, up to eight maximum. Note that these parameters must match the TYPE of parameters in the custom routine.

For convenience, the example from INSTALL is printed here.

EXAMPLE

```
;-----------------------------------------------------
;  This task program illustrates how to install
;  the sample user-written library -
;  `Programs`Sample~Library~      - in GRiDTask.
;  The new functions are: CONCAT$, FLASH, MAX, and DIV
;-----------------------------------------------------
TASKWINDOW 0,0,-1,-1
INSTALL DEVICE$ + "Programs`Sample~Library~"
;--------------------------
str1$ = "One and "
str2$ = "two and ..."
PRINT CONCAT$ (str1$,str2$)
;--------------------------
FLASH: FLASH: FLASH
;--------------------------
PRINT "This is  MAX(4,5)"
PRINT STR$ (MAX(4,5))
;--------------------------
```

```
PRINT "This is  DIV (5,PI)"
PRINT STR$ (DIV (5,PI))
STACKMSG "Press any key to exit"
PAUSE ""
```

Using String Parameters with Installed Verbs
There are two important points to remember when using string parameters with installed verbs.

First, installed verbs which have string parameters must free these string parameters. This can be accomplished by calling the GRiD Common Code procedure FreeString or another function or procedure that frees the strings. GRiDTask assumes that all string parameters passed to installed verbs are freed.

Second, any string which is returned by an installed function is freed by GRiDTask. You must be careful *not* to reference any string which was previously passed to GRiDTask by an installed function.

EDITFORM$

```
lastKey$ = EDITFORM$ (formNum, topLeftX, topLeftY, widthX, heightY,
mode)
```

NOTES

EDITFORM$ displays the form identified by formNum (from FORMINIT) and waits for input from the user. After the user fills in the form and presses ESC, CODE-RETURN (for confirm), or any other CODE-key sequence, control is returned to the GRiDTask program. The string variable lastKey$ is set to the key pressed by the user. The form contains the latest data, regardless of the key pressed.

The font set when EDITFORM$ executes must be the same font set when FORMINIT initialized the form.

The parameters for EDITFORM$ are as follows:

formNum is a number returned by FORMINIT that identifies the form.

topLeftX and topLeftY specify the pixel location of the top left corner of the form. widthX and heightY identify the size of the space in which the form appears.

You must reserve one line at the bottom of the screen where messages can be displayed. The height of the line must be equal to the height of the current font. By specifying -1 for heightY, the forms library will automatically provide the maximum height for the display of the form and leave room for one message.

By specifying -1 for widthX, the forms library will automatically provide the entire width of the display for the form.

mode specifies how the form is to be displayed. The choices are as follows:

1 The form area is erased and the form is drawn normally. This option is the conventional method for displaying forms.

2 The form area is not erased and only the contents of the fields are drawn. This option is useful for displaying field data if the form is already displayed on the screen. It prevents an annoying screen refresh.

3 The form is not displayed, but all of the field values are recalculated. This option is useful for updating field values in forms whose fields are linked to data in other forms. This mode does not modify the display in any way.

4 The form area is not erased before the form is drawn. This option can be used to display a form on top of another image, such as a logo.

EXAMPLE

```
currentForm     = FORMINIT (READFILE$ (filename$))
lastkey$        = EDITFORM$ (currentForm, 0, 0, -1, -1, 1)
```

The width and height items are set so that the form extends to the edges of the display, reserving one line at the bottom of the screen for messages.

Field Options and Properties Options and properties - summarized in Table 1-1 on the next page - determine the format, appearance, and other characteristics of the various fields in the form. A field option consists of a keyword followed by one or more parameters. A field property consists of a field name followed by a colon and a space, a keyword, and one or more parameters.

The following rules apply to field options and field properties:

o The same keywords can be used both as a field option and a field property. A field option applies to every field definition in the form. A field property applies only to the definition specified by the field name. A field property overrides a conflicting field option.

o The field options must be preceded by the text :OPTION: .

o The first field property must be preceded by the text :PROPERTY:.

o You can abbreviate keywords and parameters, as indicated in each description. For example, you can specify either AL or ALIGN as a keyword, and either RI or RIGHT as its parameter. The abbreviated keyword must contain at least the first two letters of the keyword.

If a field option or property has more than one keyword, insert one of the following between each keyword and the preceding keyword parameter:

o One or more blank characters

o A comma

o One or more carriage-returns (press RETURN)

For example, the following three definitions for <u>ssnum</u> are all valid:

```
ssnum:  AL(RI) CO (UPPER)    EX(NO)
ssnum:  AL(RI), CO (UPPER), EX(NO)

ssnum:  AL (RI)
        CO (UPPER)
        EX (NO)
```

Summary of Option and Property Keywords

Table 1-1 briefly describes the option and property keywords, and their respective parameters. The sections following the table, given in alphabetical order, give a detailed description of each of these items

Table 1-1. Summary of Options and Properties

Option/Property	**Default**	**Function**
AL (LE ¦ RI ¦ CE)	Left.	Align. Places the characters in the form field to the right or left, or centers them.
CH (list)	None.	Choices. Specifies a choice menu for the field with the choices specified in list. The choice menu is displayed above the field, and the user must press CODE-= to move into it.
CO (UP ¦ NO)	None.	Convert. Changes alphabetic characters to uppercase when they are retrieved from the form.
ED (YES ¦ NO ¦ PR)	Yes.	Editable. When set to PR (for protected), the user cannot enter the field. When set to NO, the user can enter but cannot change data in the field to which the keyword applies.
EQ (expression)	None.	Equation. Sets the value of the field to expression, which can consist of any of the following: strings, numbers, field names, arithmetic operators, and logical operators.
EX (NO ¦ YES)	Yes.	Expandable. When set to Yes, the number of characters typed in by the user can exceed the length of the field: otherwise, the number of characters cannot be greater than the length in the field definition.
FO (n)	2	Format. The number of digits to the right of the decimal point to display.

HE (string)	None.	Help. Specifies help text retrievable by the user after pressing CODE-?.
HI (UN ¦ OU ¦ IN ¦ NO)	No.	Highlight. Specifies if the field is to be underlined (UN), enclosed in a box (OU for _outlined_), inverted (IN) or not highlighted (NO).
MA (mask) (message)	None.	Mask. Forces the user to enter characters in a specified format. Incorrectly entered characters cause a message to appear.
PR (message)	None.	Prompt. Causes a prompt to appear in the message line at the bottom of the screen.
RA (expression) (message)	None.	Range. Defines a maximum and minimum value that the user can enter in a field. An incorrectly entered range causes a help message to appear.
RE (YES ¦ NO ¦ LE)	No.	Required. When set to Yes, the user must enter at least one character in the field; when set to LE (for _length_), the user must enter a character in every position of the field.
TY (ST ¦ NU)	String.	When set to NU (for numeric), the user must enter numeric characters in the field. If you omit TY or specify ST, the user can enter any printable ASCII character (A-Z, a-z, 0-9, and punctuation and other special characters)

AL -- ALIGN

Default: AL (LEFT)

The ALIGNMENT keyword has the following format:

AL (LEFT | RIGHT | CENTER)

Enter <u>LE</u> for left, <u>RI</u> for right, or <u>CE</u> for centered to specify data alignment in the field as it is entered.

CH -- CHOICES

The CHOICES keyword has the following format:

CH (list)

CH causes a choice menu to appear above the field when the user moves into the field. The choices are listed in <u>list</u>. Each choice must be separated from the following choice by a vertical bar. For example:

fieldName: CH (apples|oranges|pears|peaches)

This specifies that "fieldName" has four allowable choices.

The choice menu is hidden from the user until the user moves into the field; then, the choice menu automatically appears above the field. The user must press CODE-= to move the cursor up into the choice menu. The arrow keys are used to move the cursor once it's inside the choice menu. The user must confirm to make a selection.

All other field properties are still in effect when you use choice menus. If you want to have a choice only field, you must specify that the field is non-editable. For example:

Sex: ED (no) RE (yes) CH (male|female)

CO -- CONVERT

Default: CO (NONE)

The CONVERT keyword has the following format:

CO (UPPER | NONE)

When you specify UPPER, all lower-case alphabetic characters are changed to uppercase when converted to strings using GETFIELDVALUE$ and GETALLFIELDS$.

ED -- EDITABLE

Default: ED (YES)

The EDITABLE keyword has the following format:

```
ED (YES | NO | PR)
```

If you specify PR (for <u>protected</u>), the user cannot enter the field to which the keyword applies; the cursor will skip over the field. Note that you can allow a protected field to be edited by using the SETCURRENTFIELD verb to specifically position the cursor there. If you specify NO, the user can enter but cannot change data in the field to which the keyword applies. Specify PR or NO if you want to display only data that should not be modified.

Note that the user cannot modify the data in fields for which you specify an equation (described in the next section).

EQ -- EQUATION

The EQUATION keyword has the following format:

```
EQ (expression)
```

EQ causes the field to be assigned the value of the expression. <u>expression</u> can consist of the following:

o Any numeric or string value

o Any fieldname or formula that produces a numeric value

o Arithmetic and logical operators

o IF/THEN/ELSE conditional expressions

o Built-in functions

All of the following are valid numeric expressions:

```
25.27
10 + 15.2
totalfld + taxfld
ABS (totalfld + taxfld)
```

A field name can be used in a conditional expression. For example, the following expression is valid:

```
IF fieldA > 4 THEN 0 ELSE fieldB
```

Note that the user cannot modify the data in a field for which you specify an equation.

	performs the same function in a single-line field, and in multi-line fields when the cursor is on the top line.
SHIFT-DownArrow	Moves the cursor to the field below the current field. Pressing the DownArrow key alone performs the same function in a single-line field, and in multi-line fields when the cursor is on the last line.
RETURN	In a multi-line field, moves the cursor to the next line. If the cursor is in the last line of a field, moves the cursor to the next field.
CODE-SHIFT-UpArrow	Moves the cursor to the first field in the form.
CODE-SHIFT-DownArrow	Moves the cursor to the last field in the form.
CODE-DownArrow	In a multi-page form, moves the cursor to the first field on the next page.
CODE-UpArrow	In a multi-page form, moves the cursor to the first field in the previous page.
CODE-=	In a field with a choice menu, moves the cursor up into the choice menu, allowing the user to select and confirm one of the listed choices.