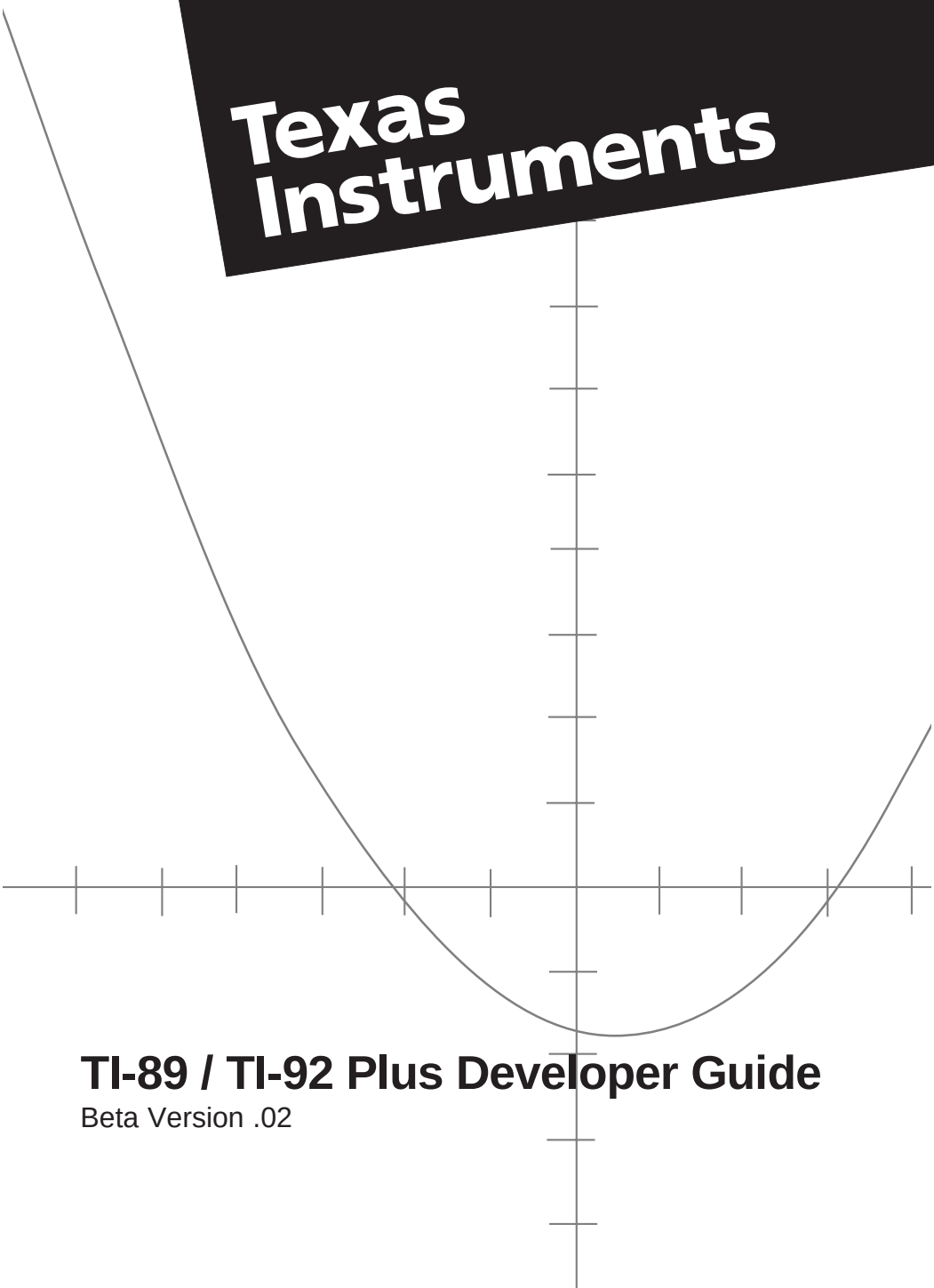
Texas Instruments



TI-89 / TI-92 Plus Developer Guide

Beta Version .02

Important information

Texas Instruments makes no warranty, either expressed or implied, including but not limited to any implied warranties of merchantability and fitness for a particular purpose, regarding any programs or book materials and makes such materials available solely on an “as-is” basis.

In no event shall Texas Instruments be liable to anyone for special, collateral, incidental, or consequential damages in connection with or arising out of the purchase or use of these materials, and the sole and exclusive liability of Texas Instruments, regardless of the form of action, shall not exceed the purchase price of this product. Moreover, Texas Instruments shall not be liable for any claim of any kind whatsoever against the use of these materials by any other party.

The latest version of this Guide, along with all other up-to-date information for developers, is available at www.ti.com/calc/developers/.

Table of Contents

Figures

Tables

1. Introduction

1.1. Purpose of this Guide

The purpose of this guide is to provide the application developer with a thorough understanding of the ideas and concepts necessary for application design on the TI-89 / TI-92 Plus Operating System (OS). This Operating System is referred to as the Advanced Mathematics Software (AMS). Key components of the AMS such as event-driven architecture, memory management, and the user interface are discussed in detail. Sample code is provided for developers of both Assembly Language Programs (ASM) and Flash applications.

1.2. Chapter Layout

Chapter 2, *The 68000 TI AMS Operating System Overview*, introduces the event-driven architecture of the AMS. It also discusses the three types of applications a user may develop: TI-BASIC programs, ASMs, and downloadable Flash applications.

Chapter 3, *The TI-89 / TI-92 Plus Hardware Overview*, provides block diagrams and tables that include information on memory, interrupt vectors, and ASIC registers.

Chapter 4, *User Interface Overview*, provides brief explanations and examples of windows, menus, toolbars, pop-ups, dialog boxes, fonts, and the status line.

Chapter 5, *Flash Applications vs. ASM Programs*, presents a side-by-side comparison of downloadable Flash applications and Assembly language programs. The limitations and advantages of each are emphasized. This is a good reference for ASM developers who are considering Flash application development.

Chapter 6, *Assembly Language Programming Overview*, discusses the general ideas and concepts necessary for ASM design. A sample ASM program is provided.

Chapter 7, *Flash Application Layout*, contains detailed descriptions of the crucial components and physical layout of an application. Sample applications are provided. Also included are instructions on how an application can take advantage of TI-BASIC extensions, a shared-code library, and language localization.

Chapter 8, *Integrating a Flash Application*, discusses areas of the operating system that an application may choose to integrate itself with such as the catalog, the mode screen, and VAR-LINK. More details are provided for interfacing with TI-BASIC. This chapter also provides tips on optimizing code space and identifying the active AMS version.

Chapter 9, *Application Control Flow*, contains vital information for the Flash application developer. The AMS event-driven architecture is further explained, complete with a detailed list of commands an application can expect to receive. Information on keyboard events and menu processing can also be found in this chapter.

Chapter 10, *Error Handling*, describes the AMS implementation of error handling. It includes explanations of how an application can throw errors, catch errors, and clean up when an error occurs.

Chapter 11, *Creating the User Interface*, gives in-depth detail about the user interface components introduced in Chapter 3, *User Interface Overview*. A sample application which illustrates the use of these components is provided. The resource compiler is also discussed in this chapter.

Chapter 12, *Basic Text Editing Facility*, describes how an application can use text records to get information from the application user. Sample code illustrating the use of the text edit facility is provided.

Chapter 13, *Memory Management*, provides explanations of dynamic data storage, application data storage, and variable management.

Chapter 14, *Data Types*, defines the structure of the twelve data types supported by the AMS.

Chapter 15, *Expressions and the Expression Stack*, contains information important for applications that use the math engine for numerical or symbolic analysis. The internal representation of expressions is discussed in detail and augmented with examples.

Chapter 16, *Working with Numbers*, describes the two separate number subsystems that are built into the AMS operating system. It also discusses the use of the expression stack for performing numeric operations.

Chapter 17, *Graphing*, contains a thorough explanation of the Graph application as well as instructions on how graphing can be incorporated into a downloadable application. Two graph mode, graphing functions, and its usage of screen and memory are detailed.

Chapter 18, *TI **FLASH** Studio™ (IDE) Overview*, is the users manual for TI **FLASH** Studio. It provides information on PC requirements, the installation process, and the interface. It also contains an example that steps through the application development process.

1.3. Conventions Used in this Guide

Bold text indicates the name of a function, macro, or global variable that is described in the System Routines (Entry Points) section.

Italicized text indicates the name of an input parameter. It is usually associated with a function prototype.

The `Courier` font is used to distinguish Assembly or C program text.

2. The 68000 TI AMS Operating System Overview

The AMS calculator Operating System (OS) implements a classic cooperative event-driven architecture. The event manager interfaces with the device drivers to determine when something important has happened such as a keypress or a timer interrupt. This information is then packaged into an event message and sent to the application currently active in the calculator screen. An application reacts to event messages by performing some action such as moving its cursor or repainting its window. After responding to an event message, the application then returns to the event manager and awaits the next event. The event manager puts the calculator into low power mode until another event occurs. This process is illustrated in Figure 2.1.

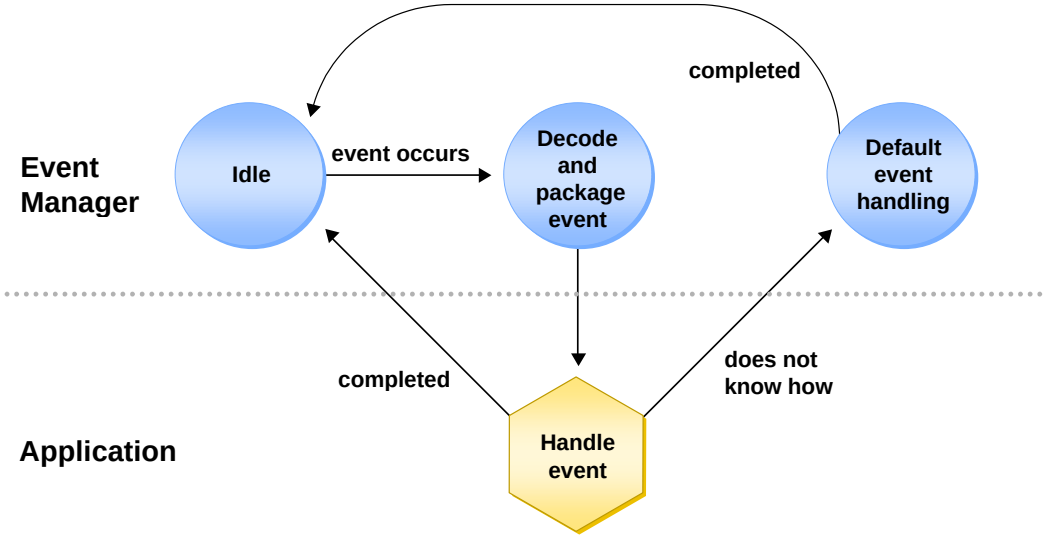


Figure 2.1: AMS Event Handler

Many operating system routines are available to the application as shown in Figure 2.2. The address of the jump table, a table of Operating System entry points and data structures, is stored in memory location 0xC8. Through the jump table, applications, and ASM programs can access low-level device drivers, the event manager, memory manager, symbol table manager, graphical user interface library, computer algebra system, math package, and utility routines.

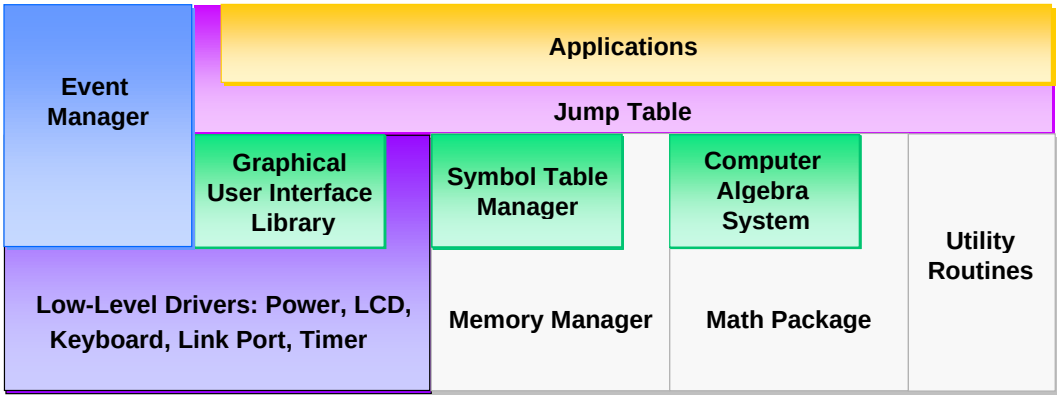


Figure 2.2: Application/OS Interface

TI‑BASIC programming language provides ease of programming at the expense of speed and control of every calculator feature. ASM programs are routines written in C or 68000 assembly language, both of which give the software developer much greater control over the calculator. ASM programs can be called from TI‑BASIC and executed from the Home screen author line.

ASM programs are generally small (≤ 8 K for AMS 2.03 and ≤ 24 K for AMS 2.04) and execute in RAM. They are intended to offer the same speed and efficient hardware access as Flash applications but as subroutines called from TI‑BASIC instead of fully integrated applications.

3. The TI-89 / TI-92 Plus Hardware Overview

3.1. Overview

The TI-89 and TI-92 Plus Graphing calculators provide a platform for writing interactive applications that utilize input, processing, storage, communication, and presentation. When creating applications, understanding the capabilities and limitations of the platform are important to creating a good interface to the user.

Pictured below is a block diagram of the TI‑89 / TI‑92 Plus from a programmer's perspective.

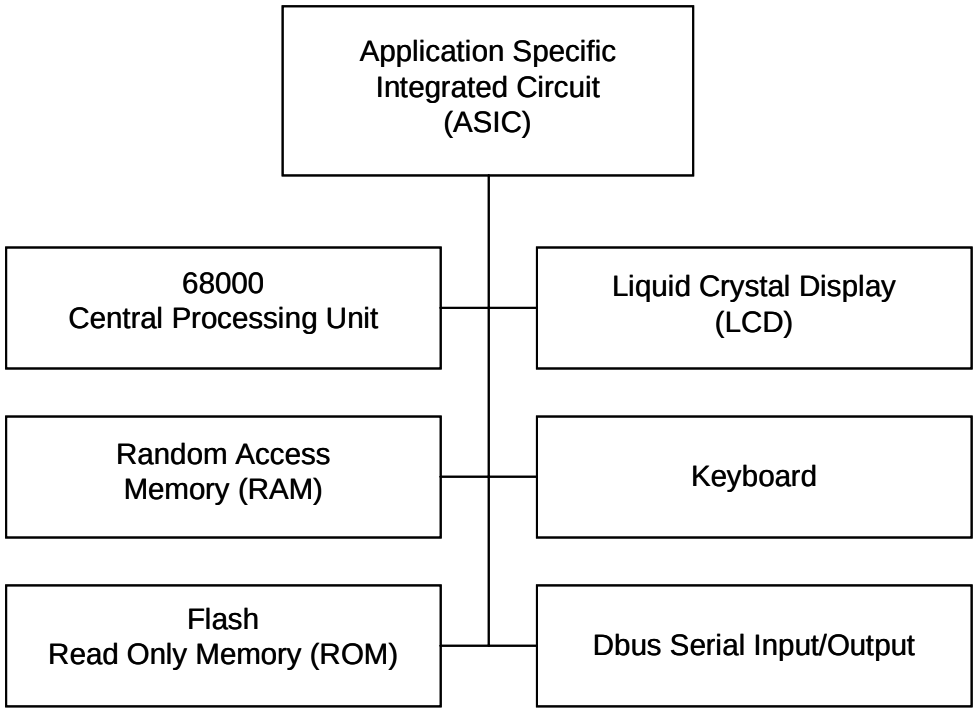


Figure 3.1: System Block Diagram

From the block diagram, the specific details of the components are:

- An ASIC, which contains all of the "glue logic" that allows the different components to communicate with each other, as well as specialized registers for system control.
- A Motorola 68000 CPU.
- 256 K bytes of RAM.
- 2 MB bytes of Flash ROM.

- A black and white LCD display (100 lines of 160 pixels for the TI-89, 128 lines of 240 pixels for the TI-92 Plus).
- A set-line / scan-line style matrix keyboard.
- A three line (D0, D1, and ground) serial IO interface.

This chapter describes in some detail the locations that an application may need to access in order to accomplish a task. For the most part, interaction with the hardware can be accomplished through the use of specific entry points. To remain compatible with other applications and future revisions of the OS, the developer should utilize entry points wherever possible.

3.2. Memory Map

Memory for the calculators consists of RAM memory, Flash memory, and address space within the ASIC. Memory is limited and optimization is important when developing for these platforms. Additionally, since Flash memory stores the OS, certificates, applications, and does not have infinite life, more restrictions are placed on its use.

The OS handles all memory allocation through the heap or file system. See chapter **13. Memory Management**. For reference, the memory map is shown in Table 3.1.

TI-89	Contents	TI-92 Plus
	RAM	
0x000000 0x0003FF	Vectors See section **3.2.1. Vector Table**	0x000000 0x0003FF
0x000400 0x0041FF	User Stack	0x000400 0x0041FF
0x004200 0x004203	0xDEADDEAD (Fence)	0x004200 0x004203
0x004204 0x004BFF	Supervisor Stack	0x004204 0x004BFF
0x004C00 0x005AFF	LCD Buffer	0x004C00 0x005AFF
0x005B00 (Border can vary)******	System bss and data segments	0x005B00 ****** (Border can vary)
(Border can vary)****** 0x3FFFF	Heap	****** (Border can vary) 0x3FFFF
	Flash ROM	
0x200000 0x20FFFF	Boot Sector	0x400000 0x40FFFF
0x210000 0x211FFF	Certificate Memory	0x410000 0x411FFF
0x212000 0x21FFFF	System Privileged	0x412000 0x41FFFF
0x220000 (Border can vary)******	Operating System	0x420000 ******(Border can vary)
(Border can vary)****** 0x3FFFFF	Archive Memory	******(Border can vary) 0x5FFFFF
	ASIC	
0x600000 0x7FFFFF	See section **3.2.1. Vector Table**	0x600000 0x7FFFFF

Table 3.1: Memory Map

3.2.1. Vector Table

In order to modify the interrupt vectors, it is necessary to first enable writing to this region by accessing the system configuration register at 0x600000 (see Table 3.1).

Address	68000 CPU Vector	TI-89 / TI-92 Plus Usage
0x000000	Initial supervisor stack pointer	Initial supervisor stack pointer
0x000004	Pointer to operating system entry point	Pointer to operating system entry point
0x000008	Bus error	Not used
0x00000C	Address error	Not used
0x000010	Illegal instruction	Not used
0x000014	Zero divide	Not used
0x000018	CHK instruction	Not used
0x00001C	TRAPV instruction	Not used
0x000020	Privilege violation	Not used
0x000024	Trace	Not used
0x000028	Line 1010 emulator	Error handler
0x00002C	Line 1111 emulator	System jump table call interface
0x000030 0x00005F	(Unassigned, reserved)	Not used
0x000060	Spurious interrupt	Not used
0x000064	Level 1 interrupt autovector	Heartbeat timer (keyboard scan)
0x000068	Level 2 interrupt autovector	Key press
0x00006C	Level 3 interrupt autovector	One second timer (not used)
0x000070	Level 4 interrupt autovector	DBus IO
0x000074	Level 5 interrupt autovector	System timer
0x000078	Level 6 interrupt autovector	On key
0x00007C	Level 7 interrupt autovector	Stack overflow
0x000080 0x000093	Trap 0 – 4	System reserved
0x000094 0x0000A3	Trap 5 – 8	Not used
0x0000A4 0x0000AC	Trap 9 – 11	System reserved
0x0000B0 0x0000BB	Trap 12 – 14	Not used
0x0000BC	Trap 15	System reserved
0x0000C0	Unassigned / reserved	Simple ROM detect constant — 0xFF0055AA
0x0000A8	Unassigned / reserved	Address of system call jump table
0x0000AC 0x0003FF		Not used

Table 3:2: Vector Table

3.3. ASIC registers

When accessing the ASIC registers, it is important to modify only the intended bits.

0x600000 System Configuration — Detect stack overflow.								
8	7	6	6	4	3	2	1	0
						Stack protect enable.		

Table 3.3: System Configuration Register

When Bit 2 is set, logic is enabled that triggers a level 7 interrupt on any write to addresses 0xE000000 through 0xFFFFFF and 0x000000 through 0x00003FF. This mechanism is used to detect stack overflow without the penalty of a software stack probe.

0x600004 System sleep / wake-up — Stop the system oscillator to preserve power. Specify wake-up condition. Interrupt level required for wake-up.								
8	7	6	5	4	3	2	1	0
				System timer	DBus IO	Not used	Key interrupt	Heartbeat timer

Table 3.4: System Sleep Register

Writing to this register will stop the system oscillator. The system oscillator is restarted by any interrupt whose corresponding bit is set to one. Interrupts level 6 and 7 always restart the oscillator.

0x60000C DBus configuration / status (IE = Interrupt Enable)															
Control								Status							
15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
AE	LD	LTO		CLE	CAIE	CTX	CRX	SLE	STX	SRX	SLI	SA			

Table 3.5: DBus Configuration Register

AE	Autostart enable
LD	Link disable
LTO	Link time-out disable
CLE	Control Link error IE
CAIE	Control Autostart IE
CTX	Control TX buffer empty IE
CRX	Control RX buffer full IE

SLE	Status Link error
STX	Status TX buffer empty
SRX	Status RX buffer full
SLI	Status Link interrupt
SA	Status Autostart

DBus is a three line serial IO interface; D0, D1 and ground. The following two registers are used to send and receive data through the DBus port. Reading the DBus status register resets that register.

0x60000E Link Data — Send / receive data through the link port.															
Low								High							
15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
				D1 In	D0 In	D1 Out	D0 Out	RX / TX buffer							

Table 3.6: Link Register

Autostart works in conjunction with the sleep/wake up register. If this bit is set and bit three of the sleep/wake-up register is set, the system will wake on DBus activity.

Link disable can be used to allow for direct monitoring of the DBus lines (see next register). When this bit is set, the state machine and barrel shifter that decode the DBus protocol and perform serial to parallel conversion are disabled.

The DBus protocol specifies a maximum bit time of two seconds. Link time-out occurs if D0 or D1 remains low for longer than this time. Bit 13, when set, disables the link time-out. If this bit is enabled, and D0 or D1 remain low for longer than two seconds after the state machine has started to decode a byte, a link interrupt is triggered and bit 7 is set to data error.

Bits 8–11 allow for enabling or disabling their corresponding interrupts. Bits 2–7 allow monitoring of the link port. These registers are modeled on RS232 control/status registers; programming serial IO on the TI‑89 / TI‑92 Plus is somewhat akin to writing an RS232 handler.

0x600014 Clock configuration — Clock / LCD control.								
8	7	6	5	4	3	2	1	0
						One second timer		LCD On

Table 3.7: Clock Configuration Register

Writing a 1 to Bit 2 of this register will trigger an autolevel 3 interrupt once per second. Writing a 1 to Bit 0 of this register blanks the LCD.

4. User Interface Overview

The TI‑92 Plus has a display of 240 by 128 pixels. The TI‑89 display size is 160 by 100 pixels. On both calculators, the display is divided into two regions: the window region and the status line. The bottom seven lines of the display are always used for the status line. The remaining lines constitute the window region which is available to the app. The window region is shared with the app's toolbar (if it has one) which is normally in the top 18 pixels of the display of the TI‑92 Plus and the top 16 pixels of the TI‑89. The window region will be different if an app is running in split screen mode (an app is given the size of its window region when it is started).

The user interface consists of windows, menus, dialog boxes, fonts, and the status line. An overview of these is presented in the following sections.

4.1. Windows

All characters, lines, figures, and images that appear on the display must be drawn to an open window. Note that menus and dialog boxes open and close their own windows and that dialog box windows may overlap the app's toolbar (dialog boxes are modal so the app's toolbar is inactive when a dialog box is active). Window routines exist to:

- Open, resize, and close windows
- Draw characters, strings, lines, ellipses, rectangles, and pixels
- Fill regions (rectangular or triangular)
- Store and recall bitmaps
- Scroll horizontally or vertically

If an app opens a window, that window must eventually be closed. All output to a window is clipped and will not exceed the window boundaries. An app's main window normally has no border but if it is in split screen mode, then it has a two-pixel border (one pixel thick if not active, two pixels thick if the active window). Note that by convention, windows that are overlapped on-top of the main window, such as dialog boxes, have single pixel borders and usually have rounded borders unlike pop-ups.

4.2. Menus

Menus allow the user to select an item from a hierarchical list of items. There are two formats: toolbars and pop-ups. Toolbars normally are placed at the top of the display and accessed with the function keys (even though they may be placed anywhere including within dialog boxes). Pop-ups “pop-up” over the display and are not attached to any toolbar. The MATH and CHAR keys bring up pop-ups as shown in Figures 4.1 and 4.2.

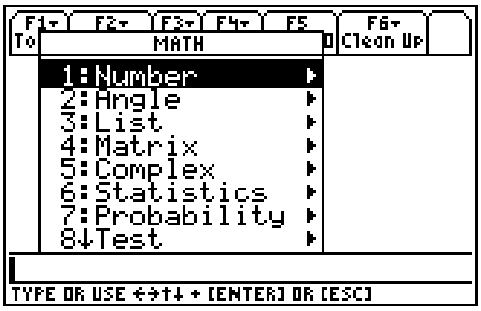


Figure 4.1: MATH Menu

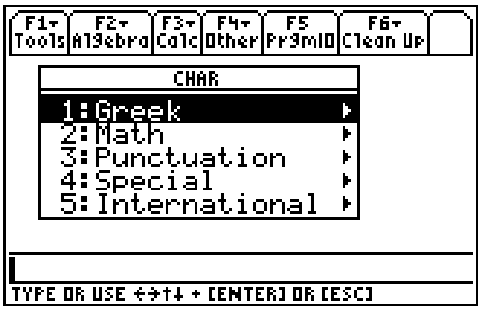


Figure 4.2: CHAR Menu

Menus can be defined with the resource compiler (static), **MenuNew** (dynamic) or both (the core menu is defined with the resource compiler which is then loaded into memory with the **MenuLoad** function so that it can be modified). The following example shows how a menu is defined by the resource compiler (see the **MenuLoad** function for an example using this menu as a core menu and then adding to it). This menu is shown in Figure 4.3.

```
TOOLBAR AddToMenu, RC_NO_IDS, 0, 240 {
   "TOP 1", 10 {
      "SUB 1", 11
      "SUB 2", 12
   }
   "TOP 2", 20 {
   }
}
```

In the above example, the numbers 10, 11, 12, and 20 are menu IDs. A menu ID is an integer in the range from 1 to 4095 (0xFFF) which can be explicitly assigned by the creator of the menu or generated by the resource compiler. In the following example, MID_1, MID_2 and MID_CORRECT will have the values 1, 2, and 3 respectively, as generated by the resource compiler. This menu is shown in Figure 4.4.

```
POPUP mPopupTest, 0, 0 {
   "POPUP 1", MID_1
   "POPUP 2", MID_2
   "SELECT THIS", MID_CORRECT
}
```

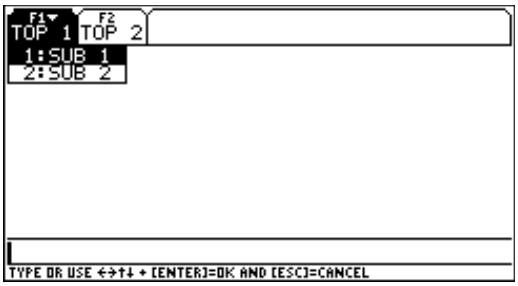


Figure 4.3: AddToMenu Screen Shot

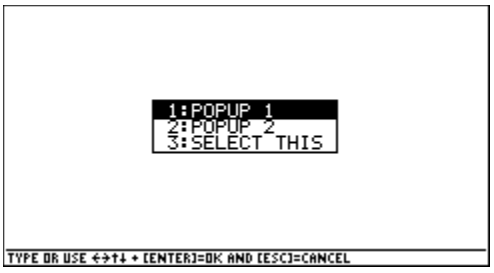


Figure 4.4: mPop-upTest Screen Shot

Menus are limited to three levels (level one being the toolbar or initial pop-up).

4.2.1. Toolbars

The function keys are normally used to select items from an app's toolbar. Toolbars are drawn with **MenuBegin**. **MenuBegin** also creates a heap-based structure to hold additional information about the menu — such as checkmark and enable/disable status. If the user presses a function key, it is passed to **MenuKey** to handle the entire menu selection process and the menu ID of the item selected is returned. Finally **MenuEnd** is called to close the menu.

There are several functions for dealing with dynamic menus. Briefly they are:

MenuNew — Create an empty dynamic menu.

DynMenuAdd, DynMenuChange — Add to or change a dynamic menu item.

MenuLoad — Load a static menu created with the resource compiler so that it can be modified with **DynMenuChange** and **DynMenuAdd**.

4.2.2. Pop-ups

There are three kinds of pop-ups which can be used depending on the features needed: static pop-ups, dynamic pop-ups, and dynamic pop-ups with menu features (checkmarks, grayed-out). Like toolbars, pop-ups are limited to three levels with the initial pop-up being the first level.

4.2.2.1. Static Pop-ups

The simplest use of pop-ups is to define a static pop-up with the resource compiler and then execute it with the **MenuPopup** function.

4.2.2.2. Dynamic Pop-ups

Dynamic pop-ups can be created using the following functions:

PopupNew	—	Create an empty dynamic pop-up.
DynMenuAdd, **DynMenuChange**	—	Add to or change a dynamic pop-up.
PopupAddText, **PopupChangeText**	—	Basically do the same thing as **DynMenuAdd** and **DynMenuChange** but for text only.
PopupClear	—	Empty out a dynamic pop-up so the handle can at least be reused in case there are other functions, like dialog boxes, that need to keep the same handle.
PopupDo	—	Execute a dynamic pop-up (do not use **MenuPopup**), returning the item selected by the user.

4.2.2.3. Dynamic Pop-ups with Menu Features

Static and dynamic pop-ups, as defined in the preceding two sections, do not have menu features like checking (adding/removing checkmarks from individual items) and the ability to gray-out individual items. In order to have those features, there are two additional functions. **PopupBegin** creates a structure similar to **MenuBegin** and returns an additional handle that can be passed to **MenuSubStat** and **MenuCheck** functions. This new handle is then passed to **PopupBeginDo** to actually execute the pop-up. If **MenuEnd** is called with this new handle, both it and the handle returned from **PopupNew** are freed.

4.3. Dialog Boxes

Dialog boxes provide a consistent method for inputting data from the user. A dialog box may consist of headers with buttons, text fields, pop-ups or edit fields. As with menus, dialog boxes can be built statically with the resource compiler or dynamically. Unlike dynamic menus, dynamic dialog boxes cannot be modified once they are created.

The routine to execute a dialog box and get back input from the user is called **Dialog**. It is passed two arrays that contain the initial and final input for the dialog box, one for the pop-ups and another for the edit fields. Dynamic dialogs are created with the **DialogNew** function and executed with the **DialogDo** function. The following example shows how a dialog is defined for the resource compiler.

```
DIALOG OverwriteDlg, 0, 0, OverwriteCallBack {
   TEXT,   {DF_OWNER_DRAW, 8, 15}
   POPUP,  {DF_TAB_ELLIPSES, DLG_DEF_X0, 28}, “Overwrite?”, OverwritePopup, 0
   EDIT,   {DF_TAB_ELLIPSES, DLG_DEF_X0, 41}, “New name”, 0, 17, 18
   HEADER, {0,0,0},  “Receive”, PDB_OK, PDB_CANCEL
}
```

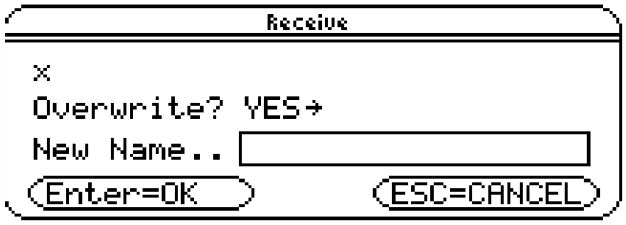


Figure 4.5: OverwriteDlg Dialog Box from Example

This example dialog shown in Figure 4.5 contains text drawn by the caller of the **Dialog** routine (*DF_OWNER_DRAW*), a pop-up defined elsewhere in the resource file (*OverwritePopup*), an edit field, and a header with two buttons (which are always placed at the bottom of the dialog box). The symbol *OverwriteCallBack* is a user supplied function that interfaces between the **Dialog** routine and the user code.

4.4. Fonts

There are three fonts used in the AMS: Small (F_4x6), Large (F_6x8), and Huge (F_8x10). The Small font is used in the status bar and dialog box headers. The Small font is also used in the dialog boxes and toolbars on the TI‑89. Some of the Small font characters, especially the international characters, are difficult to distinguish from each other and so care should be exercised when using the Small font. The Large font is used almost everywhere else, except in authoring lines on the TI‑92 Plus, which uses the Huge font. Figure 4.6 displays the three different fonts.

Figure 4.6: AMS Fonts

An app can control which of the three system fonts it uses in its own windows. Characters are drawn to the display based on the character attribute selected. The character attribute affects how the background and foreground pixels for a character are handled. This is shown in the following table.

Attribute	Background	Foreground
A_NORMAL	Unchanged	ON
A_REPLACE	OFF	ON
A_REVERSE	ON	OFF
A_SHADED	OFF	Every other pixel on.
A_XOR	Unchanged	XOR'd with destination.

Table 4.1: Available Character Attributes

Reverse mode is usually used to denote highlighted or selected items, such as in Figure 4.7.

Figure 4.7: Example Using the A_REVERSE Attribute

Shading is used to denote unselectable items. It is often used in menus, as shown in Figure 4.8.

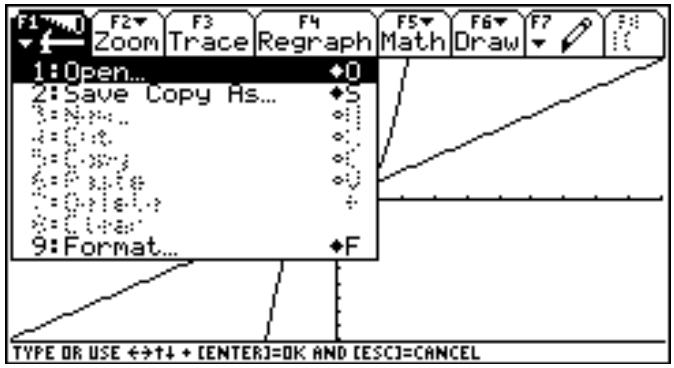


Figure 4.8: Example Using the A_SHADED Attribute

The character set is a modified ISO Latin set as shown in the following table.

		AMS		ASCII						AMS		AMS/ISO Latin Extensions					
	MSD	0	1	2	3	4	5	6	7	8	9	A	B	C	D	E	F
LSD	0	NULL	▪	SP	0	@	P	`	p	α	τ	…	°	À	Ð	à	ð
	1	SOH	◀	!	1	A	Q	a	q	β	ϕ	¡	±	Á	Ñ	á	ñ
	2	STX	▶	"	2	B	R	b	r	Γ	ψ	¢	²	Â	Ò	â	ò
	3	ETX	▲	#	3	C	S	c	s	γ	Ω	£	³	Ã	Ó	ã	ó
	4	EOT	▼	$	4	D	T	d	t	Δ	ω	¤	-1	Ä	Ô	ä	ô
	5	ENQ	←	%	5	E	U	e	u	δ	E	¥	µ	Å	Õ	å	õ
	6	ACK	→	&	6	F	V	f	v	ε	*e*	¦	¶	Æ	Ö	æ	ö
	7	BELL	↑	'	7	G	W	g	w	ζ	i	§	·	Ç	×	ç	÷
	8	BS	↓	(	8	H	X	h	x	θ	r	√	+	È	Ø	è	ø
	9	TAB	◀	)	9	I	Y	i	y	λ	T	©	¹	É	Ù	é	ù
	A	LF	▶	*	:	J	Z	j	z	ξ	$\overline{x}$	ª	º	Ê	Ú	ê	ú
	B	⇗	⬆	+	;	K	[	k	{	Π	$\overline{y}$	«	»	Ë	Û	ë	û
	C	FF	∪	,	<	L	\	l	|	π	≤	¬	*d*	Ì	Ü	ì	ü
	D	CR	∩	-	=	M	]	m	}	ρ	≠	-	∫	Í	Ý	í	ý
	E	🔒	⊂	.	>	N	^	n	~	Σ	≥	®	∞	Î	Þ	î	þ
	F	✓	∈	/	?	O	_	o	◆	σ	∠	¯	¿	Ï	ß	ï	ÿ

Table 4.2: Character Set

Note: For actual character representation of each font, see **Appendix D: TI-89 / TI-92 Plus "Small" Character Font**, **Appendix E: TI-89 / TI-92 Plus "Large" Character Font**, or **Appendix F: TI-89 / TI-92 Plus "Huge" Character Font**.

4.5. The Status Line

An app can write messages to the status bar; but when modifier keys (2nd, ↑, ◆, alpha, and ◁) are pressed or the battery level changes, the status bar is cleared of any messages and the modifier status is displayed. The status bar is also used to show things like the current directory, radian/degree mode, battery status, and other calculator information as shown in Figure 4.9.

Figure 4.9: Status Line

The following routines allow access to the status line:

ST_angle — Change the RAD/DEG indicator in the status line.

ST_busy — Turn on/off the BUSY indicator.

ST_eraseHelp — Clear the help status and restore the indicators.

ST_folder — Change current folder in the status line.

ST_helpMsg — Temporarily display a help message in the status line.

ST_readOnly — Turn on/off the lock symbol in the status line.

5. Flash Applications vs. ASM Programs

	Flash Application	ASM Program
Resides in	**protected Flash memory** — The Flash memory occupied by the OS and applications is protected from inadvertent or malicious changes.	**RAM or archive Flash memory** — ASM programs can be archived but must execute in RAM.
Executes in	**protected Flash memory** — Flash apps are executed in place, i.e., the app does not need to be moved to RAM before it can be executed.	**RAM** — ASM programs can only be executed in RAM. Calculator hardware does not allow 68000 instructions to execute in archive memory. The OS makes a temporary RAM copy of an archived ASM program before executing it.
Size	**≤ 4 MB** — Flash apps are limited by the amount of free Flash memory but can be no bigger than 4 MB.	**≤ 24 KB** — The current version of the heap manager cannot allocate any chunk of memory larger than 64 KB. The lower 24 KB limit is part of the antipiracy mechanism.
Data segment	**Yes** — The OS allocates a data segment for each application. Applications can define and reference global and static variables any of which may have an initial value.	**No** — ASM programs must allocate variables on the stack or within the code segment. This is not difficult in assembly language but C never allocates static/global variables in the code segment.
Copy protection	**Yes** — A Flash application can only be installed in calculators that have a license for the software. All calculators come with a freeware/shareware key which allow freeware/shareware applications to be installed without an additional license.	**None** — ASM programs can be freely copied between calculators.
[APPS] menu	**Yes** — Interactive Flash applications appear on the [APPS] menu.	**No** — ASM programs are not full-fledged applications. They can only be called from TI-BASIC as subprograms or from the Home screen author line.
User interaction	**Event driven** — Flash applications participate in cooperative multitasking through the OS. The OS provides default behavior for many of the special keys such as [APPS], [MODE] and function keys.	**Polled** — ASM programs must poll the keyboard to receive input from the user. No other applications can run until the ASM program returns to the OS. ASM programs do not get any automatic behavior for special keys on the keyboard.

	Flash Application	ASM Program
System overrides	**Yes** — Flash applications can override many system features.	**No** — ASM programs may move during heap garbage collect. Pointers to overriding system code would become invalid.
Localization	**Yes** — Calculator language localization is provided by Flash applications.	**No** — The OS only looks at Flash apps for language localizers.
TI-BASIC extensions	**≥ 0** — A single Flash application can provide many TI-BASIC extension functions and subprograms.	**1** — Each ASM program implements one TI-BASIC subprogram.
Shared code	**Yes** — Flash applications have a shared-code interface which can be used as a library for other Flash applications or ASM programs.	**No** — ASM programs have no OS-supported shared-code interface.
Object-oriented features	**Yes** — Each Flash application has an object frame which exposes its attributes (data and methods). Object frames can be arranged into a class hierarchy for method and attribute inheritance and class mix-ins.	**No**

6. Assembly Language Programming Overview

This chapter covers how to use assembly language to write programs for the TI-89 / TI-92 Plus calculator. You should already know how to write programs in assembly language and be familiar with Motorola 68000 architecture. See the TI Web site and the TI-89 / TI-92 Plus Guidebook.

6.1. What are ASM Programs?

ASM programs are subroutines written in 68000 assembly language. Because they appear as data type ASM in the VAR-LINK window, they are called ASM programs. They can be called from TI-BASIC programs or from the Home screen author line just like other TI-BASIC subroutines but with the advantage of speed and direct control of calculator resources that TI-BASIC as an interpreted language could never attain. ASM programs cannot, however, return function values on the estack to TI-BASIC.

ASM programs are small (≤ 8 K for AMS 2.03 and ≤ 24 K for AMS 2.04) and execute in RAM. They are easy to share with other calculators through the link port. You should consider developing a Flash application if your assembly language program is large. Because Flash applications are loaded into and execute from Flash ROM, they do not take up precious RAM. Additionally, your Flash applications enjoy a measure of copy protection that ASM programs do not provide.

6.2. Hardware Stack

The user hardware stack is 15.5 KB in size located from 0x0400 to 0x4BFF in memory. The stack serves four main purposes: it holds the return address from subroutine calls, subroutine parameters are passed on the stack, subroutine local variables are allocated on the stack, and register contents can be temporarily pushed onto and popped from the stack.

There is special circuitry in the calculator which detects stack overflow. An attempt to push a value or call a subroutine when the stack pointer is below 0x0400 causes level 7 auto-vector interrupt (address at memory location 0x007C) to occur. The level 7 auto-vector handler throws a protected memory error. See chapter **10. Error Handling** to learn how to catch errors.

6.3. Register Usage

Register A7 is the stack pointer. Do not use A7 for anything else. The stack contains the return address to the TI-BASIC interpreter when your ASM program is called.

Besides register A7, you can use the remaining registers as needed. But, if you intend to call OS-resident routines or mix subroutines written in assembly language and C, you should adhere to the Sierra C™ register usage conventions.

Registers D0, D1, D2, A0, and A1 are scratch registers. You do not need to save scratch register contents before using them. C uses register A6 as the subroutine parameter and stack-based variables frame pointer.

Save and restore D3 – D7 and A2 – A6 whenever you use them in subroutines.

The following example saves registers D3 – D5 and A2 – A3.

```
movem.l      d3-d5/a2-a3,-(sp)  ; the assembler recognizes SP as
                                ; an alternative to A7
```

Restore register values before returning from subroutine with:

```
movem.l      (sp)+,d3-d5/a2-a3
```

For memory and speed efficiency, your subroutines should only save and restore the registers you use. If you use only scratch registers, then you do not need to save any registers.

Use the `link` and `unlk` instructions with register A6 to access subroutine parameters and to allocate temporary local variables.

Sierra C expects function values to be returned in registers. Integer values are returned in D0. Pointer values are returned in A0.

6.4. Calling Flash-ROM-Resident Routines

Memory address 0xC8 contains a pointer to a table of OS routines and data structures. File tiams.inc contains jump table offsets which let you call OS routines from assembly language.

For example, to call OS routine **kbhit** to determine if a key has been pressed on the keyboard:

```
.include  "tiams.inc"
.
.
.
move.l    0xC8,a2          ; a2 -> jump table
.
.
.
move.l    kbhit(a2),a0     ; get address of kbhit routine
jsr       (a0)             ; call kbhit()
tst.w     d0               ; zero = no keypress, nonzero = key waiting
```

Arguments are passed to OS-resident routines on the hardware stack. Use the C declaration of an OS routine to determine the type and order of arguments required by the routine. Arguments are pushed onto the hardware stack in right-to-left order described by the routine's C prototype declaration.

For example, the program would call **memcmp** to compare two byte arrays. The C prototype for **memcmp** is:

```
int memcmp(const void *s1, const void *s2, size_t count);
```

The Assembly language call is:

```
; if (memcmp(ID, myid, sizeof(myid)) == 0)
              move.l      #5,-(sp)          ; push size of myid
              pea         myid(a6)          ; push address of myid
              pea         id(a6)            ; push address of ID
              move.l      memcmp(a2),a0     ; get address of memcmp
              jsr         (a0)              ; call memcmp
              add.w       #12,sp            ; pop arguments from stack
              tst.w       d0                ; test result from memcmp
              bne         notTheSame        ; not equal --->
```

When you look at OS routine C prototypes, keep in mind the size and range of C data types. In particular, note that int is two bytes in the AMS.

Type	Size (bytes)	Range
char	1	-128 ... 127
unsigned char	1	0 ... 255
short	2	-32768 ... 32767
unsigned short	2	0 ... 65535
int	2	-32768 ... 32767
unsigned int	2	0 ... 65535
long	4	-2147483648 ... 2147483647
unsigned long	4	0 ... 4294967295
pointer	4	0 ... 0xFFFFFFFF

Table 6.1: AMS C Data Types

6.5. Subroutine Linkage

Use `link` to allocate space from the hardware stack for local variables. Use `unlk` to free stack space before returning from the subroutine. Use the `movem.l` instruction to save and restore registers.

Example: subroutine linkage for subroutine mySubr with eight bytes of local variables and two parameters.

Its C prototype is:

```
short mySubr(short a, short b);
```

The Assembly language subroutine is:

```
;subroutine entry
mySubr:
             link        a6,#-8          ; allocate 8 bytes for local
                                         ; variables
             movem.l     d3-d4/a2,-(sp)  ; save registers
             .
             .
             .
             move.w      8(a6),d0        ; get parameter a from caller
             move.w      d0,-8(a6)       ; save in local variable storage
             .
             .
             .
;subroutine exit
             move.w      -2(a6),d0       ; return function result in D0
             movem.l     (sp)+,d3-d4/a2  ; restore registers
             unlk        a6              ; free stack space
             rts                         ; return from subroutine
```

Here is how stack memory looks in the above example after subroutine entry.

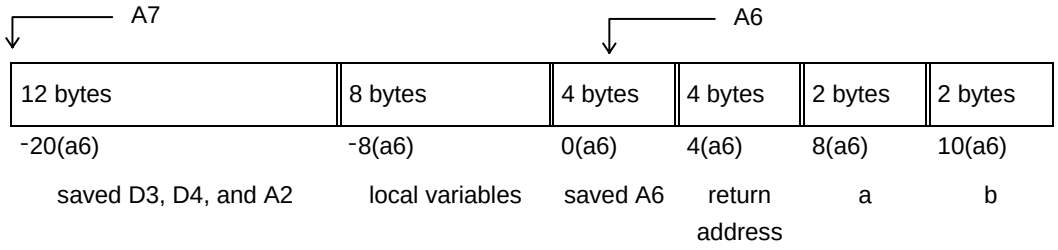


Figure 6.1: Example of ASM Stack Memory

6.6. Sample ASM Program

ASM programs do not have to be written in assembly language. Here is a sample ASM written in C. ASM program `waitkey` accepts a keypress from the user. It turns on the PAUSE indicator in the status line and puts the calculator in low power mode until a key is pressed. The key code for the pressed key is stored in a variable of the programmer's choosing.

```
/* ASM program to wait for a keypress. Go into idle mode until a
   key is pressed. */

#include "tiams.h"

/* Entry point must be called main */
void main(void)
{
   Access_AMS_Global_Variables;
   Event e;
   USHORT ch;
   EStackIndex varname;

   varname = top_estack;

   /* Argument must be string containing name of a variable */
   if (ESTACK(varname) != STR_DATA_TAG)
      ER_throw(ER_ARG_MUST_BE_STRING);

   /* Get pointer to beginning of variable name */
   varname = next_expression_index(varname-1) + 2;

   /* Make sure name is legal and not reserved for something else */
   if (TokenizeSymName(varname, TSF_PASS_ERRORS) == NULL)
      ER_throw(ER_INDIR_STRING_NOT_VARNAME);
   varname = top_estack;

   /* Get a keypress */
   while ((ch = EV_getc(ST_PAUSE, &e)) == 0)
      ;

   /* Push character number onto estack */
   push_ushort_to_integer(ch);

   /* Pop character number into variable */
   VarStore(varname, STOF_ESI, 0, top_estack);
}
```

To get a keypress code into, say, variable k, in your TI-BASIC program call *waitkey*("k").

7. Flash Application Layout

This chapter presents the physical layout of AMS Flash applications on disk and in calculator memory. It also discusses what needs to be in your source code to make a Flash app interface with the OS.

7.1. File Format

AMS application files are embedded within three layers of headers: the Flash header used by TI-GRAPH LINK™ software, the certificate header needed for license tracking, and the application header needed by the AMS OS.

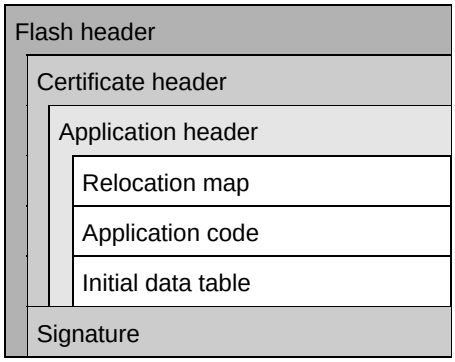


Figure 7.1: Flash Application File Format

7.1.1. Flash Header

The Flash header is used by TI-GRAPH LINK software. TI-GRAPH LINK removes this header when it sends software to a calculator.

The following table describes the fields of the Flash header used by AMS applications.

Offset (bytes)	Length (bytes)	Contents
0	8	"**TIFL**"
8	2	revision number ❶
10	1	flags (0 = binary data)
11	1	object type (0)
12	1	revision day ❶
13	1	revision month ❶
14	2	revision year ❶
16	1	length of internal application name
17	8	internal application name (zero-padded on the right end if length < 8 bytes)
25	23	filler (zeros)
48	1	device type (0x98 = TI-89, 0x88 = TI-92 Plus)
49	1	data type (0x24 = application)
50	24	filler (zeros)
74	4	length of data following header ❷

Table 7.1: Flash Header Format

❶ Binary-coded decimal, most significant digit first.

❷ Integer, least significant byte first.

7.1.2. Certificate Header

The certificate header is used by the license tracking software in the calculator. This header is kept with the application when it is downloaded into the calculator and when it is transmitted from one calculator to another or uploaded to a computer.

The certificate contains variable-length tagged fields. The application itself is actually a tagged field of the certificate. Other fields include the application's product ID (a number which cross-references the certificate with a license in the calculator's unit certificate), revision number, build number, internal name, and an embedded date certificate (the date the certificate was created).

7.1.3. Application Header

The OS keeps track of apps through the application header. The application header contains information about the application itself. The header contains the internal name of the application, flags, the length of the application's data segment, an offset to the beginning of application code, an offset to the beginning of initial data, and the length of initial data.

Lengths and offsets are stored most significant byte first — the convention used in the Motorola 68000 microprocessor.

This header is created by the MKAPPLET utility.

The following fields may be accessed through the AppHdr structure.

Offset (bytes)	Length (bytes)	Contents
0	4	magic number (0x167B533D)
4	8	internal application name (padded with trailing zeros to eight bytes)
12	24	reserved (fill with zeros)
36	2	flags
38	4	length of data segment
42	4	byte offset to code segment
46	4	byte offset to initial data table
50	4	length of initial data table
54	4	length of optional header — additional information can be stored just after the application header.
58	*n*	optional header — this information is ignored by the OS.

Table 7.2: Application Header Format

7.1.3.1. Magic Number

The magic number marks the beginning of the app header.

7.1.3.2. Internal Application Name

Every application has a unique internal name.

> **Note:** This field must match the internal application name in the Flash header. The internal application names in the app header and Flash header are case sensitive and must be identical.

Even built-in applications have internal names. You can call **EV_getAppID** with a built-in app's internal name to get its application ID. Your app can send messages to a built-in app (see **EV_sendEvent**) given its app ID.

Application Name	Internal Name
Home	TIHOME
Y= Editor	TIEQUED
Window Editor	TIWINDED
Graph	TIGRAPH
Table	TITABLED
Data/Matrix Editor	TIDMED
Program Editor	TIPRGMED
Text Editor	TITEXTED
Numeric Solver	TIINSLVR

Table 7.3: Internal Names of Built-in Applications

7.1.3.3. Flags

`0x0001` `APPHDR_LOCALIZER` Application provides language localization for the AMS Operating System (OS). The language setting pop-up menu on page 3 of the calculator's mode window is built by scanning all the app headers for applications with this flag set.

The remaining flag bits are reserved and should be zero.

7.1.3.4. Length of Data Segment

The data segment length is the amount of static RAM to allocate to the application when it is loaded into the calculator. It consists of the static initialized (.data) and uninitialized (.bss) RAM sections.

7.1.3.5. Byte Offset to Code Segment

The byte offset to code is a header-relative pointer to the beginning of the application code image.

7.1.3.6. Byte Offset to Initial Data Table

The contents of the application's initialized RAM (.data section) are initialized by copying the data from this table.

7.1.3.7. Length of Initial Data Table

Number of bytes in initial data table.

7.1.3.8. Optional Header

Additional header information may be included after the required header section. The optional header length specifies how many bytes are in the header extension.

7.1.4. Relocation Map

The OS uses the relocation map to calculate absolute addresses when the positions of the code and data segments are finally established. The relocation map consists of a six-byte entry for each location which needs to be updated.

Offset (bytes)	Length (bytes)	Contents
0	3	hole offset
3	3	base (2 bits) and relative value to place in hole (22 bits)

Table 7.4: Relocation Map Format

The hole offset specifies where in the application code or initial data table an absolute address needs to be updated.

The value to store in the hole is calculated from the base and relative value. The base is 00 for code-segment relative and 10 for data-segment relative. Base values 01 and 11 are reserved for future implementations of AMS.

7.1.5. Application Code

The code segment contains executable application code and constant data.

7.1.6. Initial Data Table

When the OS installs an application, it allocates a data segment in RAM to hold static and external variables. The initial data table provides their initial values. Static/external variables which are not explicitly initialized are set to zero.

The application's data segment is initialized when the application is installed and reinitialized every time the application is moved in Flash memory because of garbage collection.

7.1.7. Signature

The signature protects everything from the beginning of the certificate header to the end of the application from changes.

7.2. Layout in Memory

The OS creates an Application Control Block (ACB) for each app in the calculator whether it is one of the built-in apps that come preinstalled from the factory or installed later as a Flash app.

Routine **EV_getAppID** returns the ID of an app given its internal name. By design, the app ID is also the handle to the app's ACB. Dereference the handle to get a pointer to the app's ACB.

```
ACB * pacb = (ACB *)HeapDeref(EV_getAppID(name));
```

ACBs form a linked list in memory. Global OS variable OO_firstACB contains a handle to the first ACB. Routines **OO_NextACB** and **OO_PrevACB** are used to traverse the list of ACBs.

The Application Control Block contains information about the current state of the app. Here is a description of the ACB structure.

USHORT *flags* — Application control flags:

- 0x0001 — ACB_BUILTIN, the app is a built-in application.
- 0x0002 — ACB_INSTALLED, the OS sets this flag when app installation is complete.
- 0x0004 — ACB_LOCALIZER, the app is a language localizer, i. e. its name appears in the MODE screen as a language choice.
- 0x0008 — ACB_LOCK, reserved.
- 0x0010 — ACB_JT_VERSION, jump table version mismatch, do not show on the APPS pop-up menu.
- 0x0020 — ACB_SELECTED, app is selected in the VAR-LINK screen.
- 0x0800 — ACB_COLLAPSE, collapse view of TI_BASIC extension functions and commands in the VAR_LINK screen.

		0x1000 — ACB_BG, app receives background processing.
		0x4000 — ACB_COMPRESS, app is being moved during Flash memory compression.
		0x8000 — ACB_DELETE, app is about to be deleted.
AppID *myID*	—	ID of this app.
AppID *nextID*	—	ID of the next app in the linked list.
AppID *prevID*	—	ID of the previous app in the linked list.
ULONG *publicStorage*	—	Temporary storage space for the app.
AppHdr const * *appHeader*	—	Pointer to the AppHdr structure. An AppHdr resides with the app in Flash memory and cannot be changed.
BYTE const * *certhdr*	—	Pointer to the certificate header.
pFrame *appData*	—	Handle to the app's object frame. Use **OO_Deref** to convert it to a pointer.

The OS allocates memory for applications from two pools. The executable code and constant resources (.text and .const sections) are stored in Flash ROM. Static variables (.data and .bss sections) are stored in RAM.

Applications are stored in Flash memory beginning at the first sector boundary after OS code. The OS reapportions archive memory to application memory as needed to make room for additional Flash applications. When an application is deleted, applications after it in memory are moved up to fill the void. Flash sectors vacated during this process are returned to archive memory.

One pad byte of 0xFF is added between apps if needed to make sure each app begins on an even address boundary.

The OS allocates the application's static data in high RAM. The handle to this chunk of memory is locked to assure that it does not move during heap compaction. The data segment contains the app's Application Control Block, initialized data (.data), and uninitialized data (.bss). The OS frees the data segment when the application is deleted.

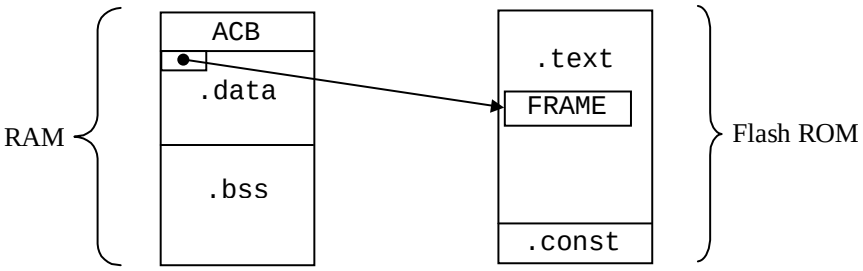


Figure 7.2: Application RAM and Flash Usage

The OS expects the first variable in the application's initialized data (.data section) to be a pointer to the app frame. The OS must be able to find the app frame in order to get the app's event processor entry point. To make this happen, you must declare a variable of type `pFrame` in your C source and initialize it with the address of your application's frame. Furthermore, you need to make sure it is the first initialized variable in your application.

7.3. Source Layout

An AMS application can serve several purposes. It can have an interactive user interface through windows and the keyboard — the most common usage. It can extend TI-BASIC with a library of functions and programs. It can implement a shared-code library — routines which can be called from other applications. And, it can override tables in the OS or another application to provide local language customization. An app can provide a mix of any of these features.

This section looks at the source requirements your application needs to implement each of these features.

7.3.1. Interactive Applications

Interactive applications need an object frame and an event handler. The object frame provides a directory of attributes and methods in the application and serves as an interface to the OS. It is in the object frame that the OS finds the address of the app's event handler.

This section covers the layout of the object frame with the help of `FRAME`, `ATTR`, and `ENDFRAME` macros, how the OS finds the app's frame, and details of predefined frame attributes and methods. Finally, a simple example pulls the pieces together into a complete application.

7.3.1.1. FRAME

Here is how a typical frame looks.

```
#define STRING1 (OO_FIRST_APP_STRING+0)
#define STRING2 (OO_FIRST_APP_STRING+1)

FRAME(frameName, parent, prototype, firstAttr, count)
   ATTR(attrSelector, value)
   ATTR( . . . )
   .
   .
   .
   ATTR(OO_FIRST_STRING+STRING1, "a string")
   ATTR(OO_FIRST_STRING+STRING2, "another string")
   .
   .
   .
ENDFRAME
```

The FRAME macro defines the header of an object frame, an OO_Hdr structure.

- *frameName* — name of the object frame. This becomes the name of the OO_Hdr structure.
- *parent* — pointer to another frame higher in the object hierarchy. This field should contain OO_SYSTEM_FRAME, a pointer to the root of the object hierarchy in the OS.
- *prototype* — pointer to another frame on the same level of the object hierarchy, often another frame within the application. This value can be zero (0) if the application has only one object frame. At any rate, this value must be zero in the last prototype frame of a linked list.
- *firstAttr* — number of the first attribute or method selector in the frame. This must be the same value as the *attrSelector* of the first ATTR macro in the frame.
- *count* — count of attributes and method selectors in the frame.

The FRAME header is followed by ATTR macros which define frame attributes (OO_Attr structures).

- *attrSelector* — the selector number of the attribute. Every selector within a frame must be unique and sorted into increasing order. The object frame accessor functions (OO_GetAttr and OO_SetAttr) look up frame attributes by their selector number.

> **Note:** The OS does not check that attribute selectors are in increasing order. You must make sure the attributes are in order when you create the frame. Attribute look-up will fail if they are not.

- *value* — the value of a frame attribute. This can be a number, a pointer to a string, a method entry point, or pointer to an application-defined structure — anything which can fit in 32 bits.

The frame ends with an `ENDFRAME` macro.

The `FRAME` and `ATTR` macros create `const` data structures which reside in Flash memory in the .text section of the application. For this reason, the `FRAME` macro also sets the read-only flag in the object frame header.

7.3.1.2. Pointer to FRAME

The OS must be able to find the object frame in the application. The OS expects the first initialized variable in an application to be a pointer to the application's object frame.

For example, say an application has an object frame named `myAppFrame`. Declare an initialized frame pointer:

```
pFrame pAppFrame = (pFrame)&myAppFrame;
```

By declaring `pAppFrame` before any other initialized static variables, the first variable in the .data section will be a pointer to the app frame — just where the OS expects to find it.

7.3.1.3. Object Frame Attributes

Application object frames can contain many attributes and method entry points. This section discusses the attributes and methods reserved by the OS.

Each attribute and method name is a C macro which defines a numeric selector number. In addition, there are macros in tiams.h which fetch and set application attributes and call application methods. The following slot descriptions state the attribute or method name (slot number) and accessor prototypes.

Many application attributes are hard-coded in the source and are not meant to be changed at run-time. The description of read-only attributes shows only the macros to fetch their values.

7.3.1.3.1. Attribute OO_APP_FLAGS (0x1)

```
APP_Flags GetAppFlags(AppID)
```

`0x0001`	`APP_INTERACTIVE`	Application has an interactive interface. Its name appears on the app's menu. This flag is zero for libraries and language localization apps.
`0x0002`	`APP_CON`	Attach Current/Open/New submenu to application name on the app's pop-up menu. This flag is ignored if the app's APP_INTERACTIVE flag is off.

0x0004	APP_ACCESS_SYSVARS	Application can store to column variables of the data matrix editor without causing a protected variable error message.
0x0008	APP_BACKGROUND	Application wants CM_BACKGROUND events. This allows applications, including noninteractive applications, to get execution time even when the application is not active.

The remaining flags are reserved and should be zero.

7.3.1.3.2. Attribute OO_APP_NAME (0x2)

```
UCHAR * GetAppName(AppID)
```

Pointer to the application's name. This name is displayed in the app's pop-up menu if the application has an interactive interface. The length of the name should be no more than 20 characters on the TI‑89 and no more than 32 characters on the TI‑92 Plus.

7.3.1.3.3. Attribute OO_APP_TOK_NAME (0x3)

```
UCHAR * GetAppTokName(AppID)
```

Pointer to application token name (≤ 8 characters). This is the name TI‑BASIC programs use to refer to functions and programs exported from the application. For example, if application "Linear Algebra" has a token name of linalg and exports its own implementation of the sin function, TI‑BASIC programs can call linalg.sin(. . .) which will not be confused with the built-in sin function.

This attribute is optional. If your application defines no TI‑BASIC extension functions or programs, this attribute is unnecessary.

7.3.1.3.4. Method OO_APP_PROCESS_EVENT (0x4)

```
void AppProcessEvent(pFrame self, Event * event)
```

Pointer to the application's event handler routine. The OS sends event messages to the application by calling its event handler.

This method is optional. If the application is a library with no user interface, then it can ignore event messages.

If the application needs to respond to any events, then this method must be implemented. A library application may need to know, for example, when it is being installed (CM_INSTALL), moved (CM_PACK/CM_UNPACK), or deleted (CM_UNINSTALL).

7.3.1.3.5. Attribute OO_APP_DEFAULT_MENU (0x5)

```
MENU * GetAppDefaultMenu(AppID)
void SetAppDefaultMenu(AppID, MENU *)
```

Pointer to the application's menu resource.

7.3.1.3.6. Attribute OO_APP_DEFAULT_MENU_HANDLE (0x6)

```
HANDLE GetAppDefaultMenuHandle(AppID)
void SetAppDefaultMenuHandle(AppID, HANDLE)
```

This attribute is managed automatically by the OS when your application uses a single static menu. See section **9.6. Menu Processing** on when and how to set this attribute.

7.3.1.3.7. Attribute OO_APP_EXT_COUNT (0x7)

```
long GetAppExtCount(AppID)
```

The number of TI-BASIC extension functions and commands exported by this application.

This attribute is optional. This attribute should be defined only if the application exports TI-BASIC extensions.

7.3.1.3.8. Attribute OO_APP_EXTENSIONS (0x8)

```
APP_EXTENSION const * GetAppExtensions(AppID)
```

Pointer to an array of `APP_EXTENSION` structures. There is one entry in the table for each exported TI-BASIC function or program. An entry contains the name string number, catalog help string number, and index of the function or program. Names in the APP_EXTENSION table must be sorted in ASCII order.

This attribute is optional. This attribute should be defined only if the application exports TI-BASIC extensions.

7.3.1.3.9. Attribute OO_APP_EXT_ENTRIES (0x9)

```
APP_EXT_ENTRY const * GetAppExtEntries(AppID)
```

Pointer to an array of `APP_EXT_ENTRY` structures. There is one entry in the table for each exported TI-BASIC function or program. An entry contains a pointer to the C routine which implements the extension, and a flag word which indicates whether the extension is a function or program.

This attribute is optional. This attribute should be defined only if the application exports TI-BASIC extensions.

7.3.1.3.10. Method OO_APP_LOCALIZE (0xA)

```
BOOL AppLocalize(AppID self, UCHAR const * language)
```

Pointer to the application's language localization routine. The OS calls this method in each application when the user chooses a new language in the mode window. This method returns TRUE if it switched the app to the given language.

How to localize an application for another language is covered in detail in section **7.3.4. Language Localization**.

This method is optional.

7.3.1.3.11. Method OO_APP_UNLOCALIZE (0xB)

```
void AppUnlocalize(AppID self)
```

Pointer to the application's routine to remove language localization.

This method is optional but should be implemented if the above OO_APP_LOCALIZE method is implemented.

7.3.1.3.12. Method OO_APP_CAN_DELETE (0xC)

```
BOOL AppCanDelete(AppID self)
```

Before the OS deletes an application, it calls this method to ask the application if it can be deleted. This method returns TRUE if the application can be deleted.

This method is optional. Implement it only if you have special requirements for when your app can be deleted.

7.3.1.3.13. Method OO_APP_CAN_MOVE (0xD)

```
BOOL AppCanMove(AppID self)
```

The application returns TRUE if it can be relocated to another address in Flash memory. The OS uses this method to query applications while it is preparing to garbage collect Flash memory.

This method is optional. Implement it only if you have special requirements for when your app can be moved.

7.3.1.3.14. Method OO_APP_VIEWER (0xE)

```
BOOL AppViewer(AppID self, BYTE * vartype, WINDOW *, HSYM symbol)
```

The [F6: Contents] menu command of the VAR-LINK window calls this method of each application in turn until one of them returns TRUE. If an application knows how to display variables of type *vartype*, it displays the contents of variable *symbol* in the given window and returns TRUE.

This method is optional. Applications which implement new data types may use this method to display a variable it understands in the VAR-LINK contents window. If no application returns TRUE, then the variable's contents are not displayed.

7.3.1.3.15. Attribute OO_APP_ICON (0xF)

```
BITMAP * GetAppIcon(AppID)
```

A pointer to a BITMAP which represents the application's icon.

This attribute is optional.

7.3.1.3.16. Method OO_APP_EXT_HELP (0x10)

```
void AppExtHelp(AppID self, USHORT strnum)
```

The catalog screen calls this method when the users presses [F1: Help] for a function or command extension implemented by the application.

This method is optional. It should only be implemented if the application exports TI-BASIC extensions. The system implementation of this method displays the extension's help string if this method is not implemented.

7.3.1.3.17. Method OO_APP_NOTICE_INSTALL (0x11)

```
void AppNoticeInstall(AppID self, ACB const *)
```

The OS calls this method in every application when a new application is installed. The app is passed a pointer to the new application's ACB.

This method is optional.

7.3.1.3.18. Method OO_APP_ABOUT (0x12)

```
char const * AppAbout(AppID self)
```

The VAR-LINK screen calls this method when a user presses [F6: Contents] for a Flash application. The application returns a pointer to a string containing version and copyright information.

This method is optional. The system implementation of this method displays the application's internal name, revision number, and date when its certificate was signed.

7.3.1.3.19. Attribute OO_APPSTRING (0x1000 and up)

```
char const *
```

Applications store pointers to their strings beginning with selector number 0x1000 defined by macro `OO_APPSTRING`. The menu system and language localizers expect application string numbers to be defined in the range 0x1000 – 0x17FF.

7.3.1.4. Example

C source file pipes.c:

```
#include "tiams.h"
#include "pipesr.h" ❶
#include "pipes.h"

/* Prototypes of functions in this source file */
void main(pFrame, PEvent);
static short sabs(short);
static short random(short, short);

FRAME(pipesFrame, OO_SYSTEM_FRAME, 0, OO_APP_FLAGS, 6)
   ATTR(OO_APP_FLAGS, APP_INTERACTIVE)   /* This is an interactive app */
   ATTR(OO_APP_NAME, "Pipes")   /* Name in [APPS] menu */
   ATTR(OO_APP_PROCESS_EVENT, &main)   /* Address of event handler */
   ATTR(OO_APP_DEFAULT_MENU, &pipesMenu)  /* Menu defined in pipesr.r */
   /* Strings used in menu */
   ATTR(OO_FIRST_STRING+P_Tools, "Tools") ❷
   ATTR(OO_FIRST_STRING+P_Clear, "Clear")
ENDFRAME

pFrame PipesFrame = (pFrame)&pipesFrame;/* Pointer to object frame */

#define BOX_DIMENSION (15)
#define MAX_BOX (25)

WINDOW w;
SCR_COORDS width, height;
WIN_RECT box;

/* Event handler - the OS calls this routine when an event has occurred
*/
void main(pFrame self, PEvent event)
{
   static short deltaX = 1, deltaY = 1;
   static short boxcount = -1;
```

```
switch (event->command)
{
   case CM_START:
      WinOpen(&w, event->info.startInfo.startRect, WF_DUP_SCR);
      DrawWinBorder(&w, &w.Window);
      WinClr(&w);

      width = w.Client.xy.x1 - w.Client.xy.x0;
      height = w.Client.xy.y1 - w.Client.xy.y0;

      /* Pick initial box location */
      if (box.x0 == 0 && box.y0 == 0)
      {
         box.x0 = random(1, width - BOX_DIMENSION - 1);
         box.x1 = box.x0 + BOX_DIMENSION;
         box.y0 = random(1, height - BOX_DIMENSION - 1);
         box.y1 = box.y0 + BOX_DIMENSION;
      }

      break;

   /* User pressed [F1][1:Clear] */
   case CM_CLEAR_ALL:
      WinClr(&w);
      break;

   case CM_QUIT:
      WinClose(&w);
      break;

   /* Draw pipes when system is not busy with anything else */
   case CM_NULL:
      if (boxcount <= 0)
      {
         // Choose new direction
         deltaX = random(2, 4);
         if (random(0,99) < 50)
            deltaX = -deltaX;

         deltaY = random(2, 4);
         if (random(0,99) < 50)
            deltaY = -deltaY;

         boxcount = MAX_BOX;
      }

      if (box.x0 < 0)
         deltaX = sabs(deltaX);

      if (box.x1 > width)
         deltaX = -sabs(deltaX);
```

```
            if (box.y0 < 0)
               deltaY = sabs(deltaY);
            if (box.y1 > height)
               deltaY = -sabs(deltaY);

            WinFill(&w, &box, A_REVERSE);
            WinRect(&w, &box, A_NORMAL);

            box.x0 += deltaX;
            box.x1 += deltaX;
            box.y0 += deltaY;
            box.y1 += deltaY;

            boxcount -= 1;

            break;

         case CM_ACTIVATE:
            DrawWinBorder(&w, &w.Window);
            EV_defaultHandler(event);
            break;

         case CM_WPAINT:
            WinBackupToScr(&w);
            break;

         default:
            EV_defaultHandler(event);
            break;

      }
   }

   static short sabs(short n)
   {
      return n < 0 ? -n : n;
   }


   static short random(short low, short high)
   {
      short range;
      static long seed = 29;

      if (seed == 0)
         seed = 1;

      seed *= 16807;

      if (seed < 0)
         seed = -seed;
```

```
   range = high - low + 1;

   return low + seed % range;
}

C header file pipes.h:
#ifndef _PIPES_H
#define _PIPES_H

#define P_Tools     OO_FIRST_APP_STRING ❷
#define P_Clear     OO_FIRST_APP_STRING+1

#endif
```

Resource file pipesr.r:

```
#include "tiams.h"
#include "pipes.h"

TOOLBOX pipesMenu, RC_NO_IDS, 0, 160 {
   P_Tools { ❷
   P_Clear,       CM_CLEAR_ALL
   }
}
```

❶ Pipesr.h is generated by the resource compiler.

❷ The numbering of menu commands and application strings needs some explanation. The menu system requires all string numbers that it references to be in the range 0x000 – 0xFFF. However, system string attributes begin at 0x800 and application string attributes begin at 0x1000. Consequently, when an app menu refers to string number 0x801, it fetches app attribute 0x1001. The example code shows how to use macros `OO_FIRST_APP_STRING` in the header file and `OO_FIRST_STRING` in the object frame to define menu string numbers and their corresponding text in the app frame.

7.3.2. TI-BASIC Extensions

Applications can extend TI-BASIC with functions and programs written in C or assembly language. The TI-BASIC interpreter interfaces with apps through the `OO_APP_EXT_COUNT`, `OO_APP_EXTENSIONS`, and `OO_APP_EXT_ENTRIES` attributes.

This example illustrates a simple application which implements a couple of TI-BASIC extensions. The app has no user interface, hence, no event handler entry point.

```
#include "tiams.h"

/* String numbers */
#define H_folders            0
#define H_vars               1
#define H_HELP             100
```

```
void folders(void);
void vars(void);

APP_EXTENSION const extensions[] = ❶
{
   /* function name #,       help string #,             function index */
   {OO_APPSTRING+H_folders,  OO_APPSTRING+H_HELP+H_folders,  H_folders },
   {OO_APPSTRING+H_vars,     OO_APPSTRING+H_HELP+H_vars,     H_vars    }
};

APP_EXT_ENTRY const extEntries[] =
{
   {folders,  APP_EXT_FUNCTION}, ❷
   {vars,     APP_EXT_FUNCTION}
};

FRAME(memutilFrame, OO_SYSTEM_FRAME, 0, OO_APP_FLAGS, 10)
   ATTR(OO_APP_FLAGS, APP_NONE)
   ATTR(OO_APP_NAME, "Memory Utilities")
   ATTR(OO_APP_TOK_NAME, "memutil") ❸
   ATTR(OO_APP_EXT_COUNT, 2)        /* export two extension functions */
   ATTR(OO_APP_EXTENSIONS, extensions)  /* address of extensions table */
   ATTR(OO_APP_EXT_ENTRIES, extEntries)  /* address of ext entries table
                                         */
   ATTR(OO_APPSTRING+H_folders,  "folders")
   ATTR(OO_APPSTRING+H_vars,     "vars")
   ATTR(OO_APPSTRING+H_HELP+H_folders,  "LIST OF FOLDERS")
   ATTR(OO_APPSTRING+H_HELP+H_vars,     "LIST OF VARIABLES IN FOLDER")
ENDFRAME

pFrame MemutilFrame = (pFrame)&memutilFrame; ❹

void folders(void)
/* Return a list of folders on the estack */
{
   SYM_ENTRY *pSym;
   static BYTE const HomeFolder[] = {0, 127, 0};

   push_quantum (END_TAG);

   pSym = SymFindFirst(&HomeFolder[2], FO_NONE);
   while (pSym != NULL)
   {
      push_zstr((char *)pSym->Name);
      pSym = SymFindNext();
   }

   push_quantum (LIST_TAG);
}
```

```
void vars(void)
/* Return on the estack a list of variables in current or given folder */
{
   Access_AMS_Global_Variables; ❺
   BYTE folder[SYM_LEN+1];
   BYTE tfolder[MAX_SYM_LEN];
   BYTE *fname;
   SYM_ENTRY *pSym;

   EStackIndex e = top_estack; ❺

   /* Get folder name */
   if (ESTACK(e) == END_TAG)
   {
      /* Use current folder */
      FolderGetCur(folder);
      fname = StrToTokN(folder, tfolder);
   }
   else
   {
      if (ESTACK(e) != STR_DATA_TAG)
         ER_throw(ER_DOMAIN);
      fname = e-1;
   }

   push_quantum (END_TAG);
   pSym = SymFindFirst(fname, FO_NONE);
   while (pSym != NULL)
   {
      push_zstr((char *)pSym->Name);
      pSym = SymFindNext();
   }

   push_quantum (LIST_TAG);
}
```

❶ The `extensions` table has an entry for each extension function or program. The first field of each entry is the string number of the name of the function. The second field is the string number of the function's help message. The third entry is an index into the following `extEntries` table. The entries in this table must be alphabetized by the function name.

❷ Each entry in the `extEntries` table cross-references a function number from the `extensions` table with the function's actual address. The C name of a function need not be the same as its TI-BASIC name — a language localizer may, in fact, override the exported name. The second field of each entry specifies whether the extension is a function (APP_EXT_FUNCTION) which returns a value on the estack, or a program (APP_EXT_PROGRAM) which does not return a value.

❸ The `OO_APP_TOK_NAME` attribute specifies the short name of the application to use when referencing its extension functions. In this example, TI-BASIC programs can call memutil.folders() and memutil.vars().

❹ The first initialized variable in your application must be a pointer to the app frame. Even though `extensions` and `extEntries` appear to be allocated before `MemutilFrame`, they are declared const and are not allocated with variables in the .data section. You cannot see it, but `memutilFrame` is also declared const by the `FRAME` macro.

❺ The `Access_AMS_Global_Variables` macro is required in every subroutine which needs to fetch or change AMS global variables. **top_estack** is a global variable in the computer algebra system.

7.3.3. Shared-Code Library

An application can make a library of its data structures and functions available as attribute slots in its object frame. The app library interface should use attribute slots 0x10000 (OO_FIRST_APP_ATTR) and up. Attribute slots 0x0000 – 0xFFFF are reserved for the OS.

7.3.3.1. Creating the Library Interface

Say, for example, your library implements the following functions and data structures:

```
int fileTableCount;
FILE fileTable[FILE_TABLE_SIZE];
int fileOpen(AppID self, char const *filename);
int fileRead(AppID self, int handle, char *buff, int size);
int fileWrite(AppID self, int handle, char const *buff, int size);
void fileClose(AppID self, int handle);
```

> **Note:** The first parameter of each exported function must be an AppID variable even though it will always be the ID of your library.

You can use Frame Description Language, FDL, to define an interface to these data and functions.

```
fileio.fdl . . .

appvar 0x10000 PlayerFileTableCount: int *;
appvar         PlayerFileTable: FILE *;
appfunc        PlayerFileOpen(AppID, char const *): int;
appfunc        PlayerFileRead(AppID, int, char *, int): int;
appfunc        PlayerFileWrite(AppID, int, char const *, int): int;
appfunc        PlayerFileClose(AppID, int): void;
```

Some things to note in the above example:

- The first variable, PlayerFileTableCount, is numbered 0x10000, the first attribute number available for applications. Subsequent attribute numbers are automatically incremented unless a new value is supplied.
- The variable type or function result is placed after the colon (:) much in the style of Pascal.
- Function parameters list only the order and type of arguments. Do *not* include parameter names.

Run the FDL compiler to produce a header file suitable for defining your API call macros and attribute names.

```
C>FDL fileio.fdl
```

Resulting header file fileio.h includes among other things:

```
/* Frame slot number assignments */
#define OO_PLAYER_FILE_TABLE_COUNT (65536) ❶
#define OO_PLAYER_FILE_TABLE (65537)
#define OO_PLAYER_FILE_OPEN (65538)
#define OO_PLAYER_FILE_READ (65539)
#define OO_PLAYER_FILE_WRITE (65540)
#define OO_PLAYER_FILE_CLOSE (65541)

/* Accessor/mutator macros */
GetPlayerFileTableCount(appid) ❷
SetPlayerFileTableCount(appid, int)
GetPlayerFileTable(appid)
SetPlayerFileTable(appid, FILE *)

/* Function call macros */
PlayerFileOpen(appid, name) ❸
PlayerFileRead(appid, handle, buff, size)
PlayerFileWrite(appid, handle, buff, size)
PlayerFileClose(appid, handle)
```

❶ FDL creates names for attribute slots by converting your variable and method names to upper case and adding “OO_” prefix.

❷ FDL creates a Set/Get pair of macros for each variable.

❸ These macros hide the details of method dispatch and better demonstrate in your source your intent to call a library function.

Include fileio.h in your app source so you can use the frame slot assignment macros in your ATTR declarations.

```
FRAME(playerObj, OO_SYSTEM_FRAME, 0, OO_APP_FLAGS, . . .)
   ATTR(OO_APP_FLAGS, . . .)
      .
      .
      .
   ATTR(OO_PLAYER_FILE_TABLE_COUNT, &fileTableCount)
   ATTR(OO_PLAYER_FILE_TABLE, fileTable)
   ATTR(OO_PLAYER_FILE_OPEN, fileOpen)
   ATTR(OO_PLAYER_FILE_READ, fileRead)
   ATTR(OO_PLAYER_FILE_WRITE, fileWrite)
   ATTR(OO_PLAYER_FILE_CLOSE, fileClose)
ENDFRAME
```

7.3.3.2. Accessing a Library

```
HANDLE EV_getAppID(UCHAR const * appname)
```

Get the ID of an application/shared-code library. *appname* is the app's internal name.

Once you have obtained the ID of an application, you can use the function call macros created by the FDL compiler to call routines in the app's library.

```
#include "tiams.h"
#include "fileio.h"
HANDLE libid;
   .
   .
   .
libid = EV_getAppID((UCHAR *)"TIPLAYER"); ❶
if (libid == H_NULL) ❷
{
   /* Could not find library */
}
   .
   .
   .
n = *GetPlayerFileTableCount(libid); ❸
fd = PlayerFileOpen(libid, "script");
```

❶ Get a handle to the shared-code library.

❷ **EV_getAppID** returns H_NULL if the requested library cannot be found, i.e., is not installed in the calculator.

❸ Macros defined in fileio.h can be used to access variables and call functions in the library.

7.3.3.3. Frame Description Language

Frame Description Language is provided as a tool to simplify the process of creating an interface to object frame variables and methods. FDL accepts an input file of statements in the following forms:

$\left\{ \begin{matrix} \text{var} \\ \text{appvar} \end{matrix} \right\}$ [*slot-number*] *var-name* : *type* ;

func [*slot-number*] *func-name*(pFrame [, *type* . . .]) : *type* ;

appfunc [*slot-number*] *func-name*(AppID [, *type* . . .]) : *type* ;

Comments begin with the pound sign (#) and extend to end of line.

The var and func keywords describe object frame variables and functions. FDL creates macros which let you retrieve and change the values of attribute slots and call functions in method slots. These created macros require the pointer to an object frame as their first parameter.

Every application has an object frame interface to the OS. The app's object frame also serves as a shared-code library interface. As a convenience, so you do not have to find the address of the app's object frame, the keywords **appvar** and **appfunc** create macros which access an app's object frame attributes given its app ID.

> **Note:** The first parameter of a func declaration must be type pFrame, the first parameter of an appfunc declaration must be type AppID.

The slot-number is an optional decimal (0 – 4294967295) or hexadecimal (0x0 – 0xFFFFFFFF) number which identifies the slot which the variable or method occupies. Subsequent slot-numbers are automatically incremented unless a new value is supplied.

Since attribute slots are 32-bit values, types should declare values which fit in 32-bits, integers and pointers.

> **Note:** Functions which return nothing may declare a return type of void.

The FDL compiler accepts one command line parameter, the name of the file to compile. It outputs a file of the same name with a .h extension. The output file contains C macros for accessing object frame attributes and calling frame methods. The resulting header file is suitable to include in your C source file.

Here is the actual header output file after running the FDL compiler on fileio.fdl in section **7.3.3.1. Creating the Library Interface**.

```
/*      FILE:  fileio.h
     CREATED:  2000.04.13 09:10
       INPUT:  fileio.fdl
   GENERATOR:  Frame Description Language compiler, version 2.000
*/

/* int * PlayerFileTableCount */
#define OO_PLAYER_FILE_TABLE_COUNT (65536) ❶
#define GetPlayerFileTableCount(obj) \ ❷
   (int *)OO_GetAppAttr(obj,65536)
#define SetPlayerFileTableCount(obj,value) \ ❸
   OO_SetAppAttr(obj,65536,(void *)value)
```

```
/* FILE * PlayerFileTable */
#define OO_PLAYER_FILE_TABLE (65537)
#define GetPlayerFileTable(obj) \
   (FILE *)OO_GetAppAttr(obj,65537)
#define SetPlayerFileTable(obj,value) \
   OO_SetAppAttr(obj,65537,(void *)value)

/* int PlayerFileOpen(AppID, char const *) */
#define OO_PLAYER_FILE_OPEN (65538)
#define PlayerFileOpen(obj,a) \ ❹
   ((int (* const)(AppID, char const *))OO_GetAppAttr(obj,65538))(obj,a)

/* int PlayerFileRead(AppID, int, char *, int) */
#define OO_PLAYER_FILE_READ (65539)
#define PlayerFileRead(obj,a,b,c) \
   ((int (* const)(AppID, int, char *,
int))OO_GetAppAttr(obj,65539))(obj,a,b,c)

/* int PlayerFileWrite(AppID, int, char const *, int) */
#define OO_PLAYER_FILE_WRITE (65540)
#define PlayerFileWrite(obj,a,b,c) \
   ((int (* const)(AppID, int, char const *,
     int))OO_GetAppAttr(obj,65540))(obj,a,b,c)

/* void PlayerFileClose(AppID, int) */
#define OO_PLAYER_FILE_CLOSE (65541)
#define PlayerFileClose(obj,a) \
   ((void (* const)(AppID, int))OO_GetAppAttr(obj,65541))(obj,a)
```

❶ A symbolic name is created for each frame slot number. The name consists of “OO_” prefixed to the name of each variable or function converted to upper case letters and underscores.

❷ A Get . . . macro is created for each variable attribute. It is used to get the value of an object attribute. It expands into a call to **OO_GetAppAttr** with the slot number of the attribute to retrieve. Note the return value is cast to the type of the variable.

❸ A Set . . . macro is created for each variable attribute. It is used to change the value of an object attribute. It expands into a call to **OO_SetAppAttr**.

❹ A method call macro is created for each function attribute. It expands into a call to **OO_GetAppAttr** to get the address of the routine to execute. The routine is called indirectly with the arguments specified in the parameter list of the macro. All the arguments and the routine return type are cast to the types in the original function definition so the C compiler will correctly type-check arguments and function return value. Imagine trying to create that macro manually!

7.3.4. Language Localization

Menu titles and item names, dialog text, help strings, error messages, TI-BASIC extension names, the name of the application as it appears on the app's pop-up menu, any string which appears in the app's object frame, can be overridden with a localizer app to add language customization. By placing all strings in the frame and using **XR_stringPtr** to cross-reference string numbers to string pointers, the job of localizing an app for a different language becomes much easier.

An application is typically localized by installing a small companion app which contains translations of all the app's strings for one or more languages. When the user chooses a different language on the MODE screen, all apps are notified of the new language. Each localizer app which contains a matching language, responds by overriding its target app's strings.

7.3.4.1. Localizer Template

The following sample application is a template localizer app. It can be used to customize a target app for a different language. There are places in the template to supply the language, internal name, app's menu name of the target application, and the translated strings.

Strings in the localizer app override strings with the same attribute number in the target app's object frame. Therefore, it is important that translated strings in the localizer use the same string numbers as the target app. A mismatch between the localizer string numbers and the target string numbers will lead to confusing menus and messages in the target when localization is applied.

```
/* Sample application localizer */

#include "tiams.h"

/*********************************************************************
 Set MyLang to the name of the language which this localizer implements.
 *********************************************************************/
char const MyLang[] = "Fran" RF_C_CEDILLA "ais";

/*********************************************************************
 Set TargetApp to the internal name of the app which this localizer
 will hook into. This is the same name described in section
 7.1.3.2. Internal Application Name.
 *********************************************************************/
unsigned char const TargetApp[] = "DEMOAPP";

/*********************************************************************
 Set APPSname to the name Target app should have on the [APPS] menu key.
 *********************************************************************/
char const APPSname[] = "Application de D" RF_E_ACUTE "monstration";
```

```
/*********************************************************************
 Set MY_APP_NAME to the name this localizer app should have in the
 VAR-LINK [F7:APPS] screen.
 *********************************************************************/
#define MY_APP_NAME “French Demo Localizer”

void main(pFrame self, PEvent e);
void observer(pFrame self, PEvent e);
BOOL localize(pFrame, char *lang);
void unlocalize(pFrame);
void noticeInstall(pFrame, ACB const *);
BOOL candelete(pFrame self);

static
FRAME(LocalizerFrame, OO_SYSTEM_FRAME, 0, OO_APP_FLAGS, 7)

   /* This app does not appear on the [APPS] menu */
   ATTR(OO_APP_FLAGS, APP_NONE)
   ATTR(OO_APP_NAME, MY_APP_NAME)

   /* It needs to respond to some events */
   ATTR(OO_APP_PROCESS_EVENT, &main)

   /* Export methods used in localization */
   ATTR(OO_APP_LOCALIZE, &localize)
   ATTR(OO_APP_UNLOCALIZE, &unlocalize)
   ATTR(OO_APP_CAN_DELETE, &candelete)
   ATTR(OO_APP_NOTICE_INSTALL, &noticeInstall)
ENDFRAME

pFrame appframe = (pFrame)&LocalizerFrame;

/* This little FRAME is hooked ahead of the Target app's frame. It
   accomplishes a couple of things:
   1) It renames the Target app on the [APPS] menu, and
   2) It redirects its event handler to this app so we can observe its
      pack/unpack and uninstall messages. All events are forwarded to
      their rightful owner.
*/
static
FRAME(TargetAppFrame, OO_SYSTEM_FRAME, NULL, OO_APP_NAME, 2)

   /* Override the Target app's name in the [APPS] menu */
   ATTR(OO_APP_NAME, APPSname)

   /* Redirect Target app's events to me */
   ATTR(OO_APP_PROCESS_EVENT, &observer)
ENDFRAME

/* Place the translated strings here. Keep the following OO_Hdr structure
   and target strings array together. The OO_Hdr structure is an object
   frame header for the following array of string attributes.
*/
```

```
static const OO_Hdr TargetStrings =
{
   OO_SYSTEM_FRAME,
   (pFrame)&TargetAppFrame,
   OO_RO | OO_SEQ,
   OO_FIRST_STRING + OO_FIRST_APP_STRING,
   174  // <----------------------------- number of strings
};
static char * const targetstrings[] =
{
   "first string",    // <---------------- local translation of strings
   "second string",
   /* etc. */
};

/* Keep track of when this localizer is hooked into the Target app */
pFrame hook = 0;

/*********************************************************************
 This app's event handler needs to unhook itself from the target app
 when it is deleted or moved. It can reconnect to the Target app when
 it has been reinstalled or has completed its move.
 *********************************************************************/
void main(pFrame self, PEvent e)
{
   switch (e->command)
   {
   case CM_UNINSTALL:
   case CM_PACK:
      unlocalize(self);
      break;

   case CM_INSTALL:
   case CM_UNPACK:
      localize(self, XR_stringPtr(XR_NativeLanguage));
      break;
   }
}

/*********************************************************************
 This routine observes all events sent to Target app. It localizes the
 Target after the app UNPACKs and unlocalizes the Target just before it
 PACKs or UNINSTALLs.
 *********************************************************************/
void observer(pFrame self, PEvent e)
{
   Access_AMS_Global_Variables;

   pFrame super = OO_SuperFrame;

   switch (e->command)
   {
```

```
   case CM_UNPACK:
      /* Pass the event on to the Target app before applying
         the localizer hook. */
      AppProcessEvent(super, e);
      localize(self, XR_stringPtr(XR_NativeLanguage));
      break;

   case CM_UNINSTALL:
   case CM_PACK:
      /* Unhook the localizer before passing the event on to the
         Target app */
      unlocalize(self);

   default:
      /* Forward all events to their rightful owner */
      AppProcessEvent(super, e);
   }
}

/**********************************************************************
 If requested language matches the language we know, hook over
 Target app.
 **********************************************************************/
BOOL localize(pFrame self, char *requestedLang)
{
   if (hook == 0 && strcmp(requestedLang, MyLang) == 0)
      return OO_InstallAppHookByName(TargetApp, (pFrame)&TargetStrings, &hook);

   return FALSE;
}

/**********************************************************************
 Unhook from Target app if we have anything to unhook.
 **********************************************************************/
void unlocalize(pFrame self)
{
   if (hook)
   {
      OO_UninstallAppHookByName(TargetApp, hook);
      hook = 0;
   }
}

/**********************************************************************
 We have just been notified that a new app was installed. Is it our
 Target application? If so, apply our localization to it.
 **********************************************************************/
void noticeInstall(pFrame self, ACB const *pacb)
{
   if (strcmp((char *)pacb->appHeader->name, (char *)TargetApp) == 0)
      localize(self, XR_stringPtr(XR_NativeLanguage));
}
```

```
/*********************************************************************
 This app can be deleted if Target app does not exist or Target app is
 not running.
 *********************************************************************/
BOOL candelete(pFrame self)
{
   Access_AMS_Global_Variables;

   AppID targetID = EV_getAppID(TargetApp);

   return targetID == H_NULL ||
         (targetID != EV_appA && targetID != EV_appB);
}
```

7.3.4.2. How Localization Works

Applications call **XR_stringPtr** to look up a string given a string number. When **XR_stringPtr** looks up a string, it starts with the frame pointed to by the ACB (Application Control Block) of the currently executing application (app identified by **EV_currentApp**). The prototype chain of the object frame is searched for the given string number. An unsuccessful search of the prototype chain tries again with the prototype chain of the frame's parent and ultimately to the system frame.

A language localizer app installs a new language by adding a new frame of strings at the head of the target app's object chain.

Here is how app frames are linked together before a localizer is installed.

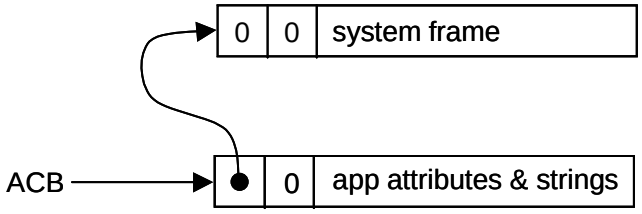


Figure 7.3: Linked App Frames

The above figure illustrates an app frame with a parent link but no prototype link. A string search would begin with the app frame then proceed to the system frame.

After installing a localizer, the ACB is redirected to point first to the localizer frame.

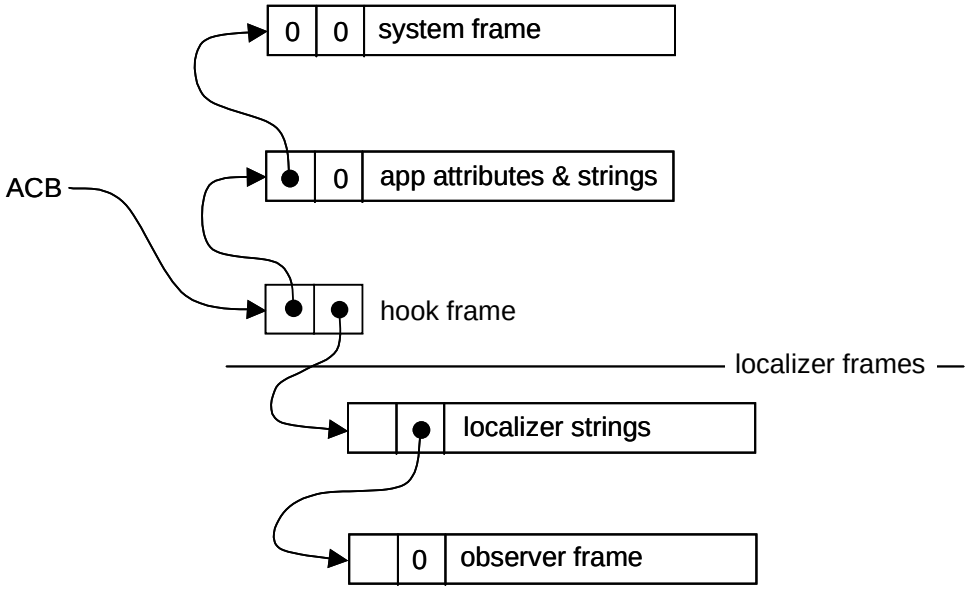


Figure 7.4: Redirected App Frame

The ACB now points to the hook frame installed in RAM by the localizer app with a call to **OO_InstallAppHookByName**. Now when **XR_stringPtr** looks up a string, it starts with the hook frame in RAM. Since the hook frame has no attributes, the search continues down the prototype chain to the localizer app's frames. The translated string is found in the localizer's string frame.

Not all strings need to be translated. If searching the localizer prototype chain proves unsuccessful, the search continues with the parent of the hook frame, picking up the search again in the app's prototype chain.

The search is quick — the header of each frame indicates the range of attributes in the frame body. No need to search a frame if the header says the sought attribute is not there. Disjoint ranges of string numbers can be placed in different frames and linked together through their prototype fields.

The observer frame at the bottom of the figure above lets the localizer app peek at every event sent to the target app. The localizer app uninstalls its language hook when it sees the target app is about to be deleted or moved. It can reinstall its hook when it sees the target app has finished a move.

8. Integrating a Flash Application

8.1. Mode Settings

A user can change the mode settings by either using the MODE screen or by executing the TI-BASIC setMode function. When any mode settings have changed, the array where they are stored is updated appropriately. In addition, all applications are sent a CM_MODE_CHANGE event message. The mode notification flags in the event message indicate which mode settings changed. Applications can ignore this message or test to see if a mode setting has changed that it needs to react to in some way. For instance, the application may need to set its window dirty flag (WF_DIRTY) if certain mode settings change that trigger a CM_WPAINT event message (requiring the application to update its windows). An example of testing for mode notification flags follows:

```
#include "tiams.h"
AP_myApp(pFrame self, Event *e)
{
   switch (e->command)
   {
   .
   .
   .
   case CM_MODE_CHANGE:
      if (!(( e->info.modeInfo.notifyFlags & MO_NOTIFY_SPLIT ) ||
            ( e->info.modeInfo.notifyFlags & MO_NOTIFY_VECTOR_FORMAT ) ||
            ( e->info.modeInfo.notifyFlags & MO_NOTIFY_PRETTY_PRINT )))
               wAppwindow.Flags |= WF_DIRTY;
         .
         .
         .
      break;
   .
   .
   .
default:
      EV_defaultHandler(e);
      break;
   }
}
```

8.1.1. Mode Notification Flags

MO_NOTIFY_FOLDER	—	Current folder has changed.
MO_NOTIFY_GRAPH_COUNT	—	2 graph mode or 1 graph mode.
MO_NOTIFY_GRAPH_TYPE_1	—	Graph mode change.

MO_NOTIFY_GRAPH_TYPE_2	—	Graph mode change for graph 2 if in 2 graph mode.
MO_NOTIFY_SPLIT	—	The screen size has changed.
MO_NOTIFY_ANGLE	—	Angle mode has changed.
MO_NOTIFY_PRECISION	—	Precision has changed between EXACT, APPROX, and AUTO.
MO_NOTIFY_FIX	—	Fix digits or float precision change.
MO_NOTIFY_NUMBER_FORMAT	—	Exponential format: Normal, scientific or engineering.
MO_NOTIFY_VECTOR_FORMAT	—	Rectangular, cylindrical or spherical.
MO_NOTIFY_COMPLEX_FORMAT	—	Real, rectangular or polar.
MO_NOTIFY_PRETTY_PRINT	—	Pretty Print on or off.
MO_NOTIFY_UNIT_SYSTEM	—	SI, ENG/US or CUSTOM unit system.
MO_NOTIFY_BASE	—	DEC, HEX or BIN base.
MO_NOTIFY_LANGUAGE	—	Language mode has changed.

8.1.1.1. Modifying Mode Settings Within an App

The mode settings can be modified within an application by calling **MO_currentOptions** to get the current mode settings into the mode option array, **MO_option**. After modifying the mode setting options, a call to **MO_digestOptions** will cause the new mode settings to take affect by sending out the appropriate mode notification messages.

A simple example of setting the split screen ratio to the 50/50 setting follows:

```
MO_currentOptions();
MO_option[MO_OPT_SPLIT_RATIO] = D_SPLIT_RATIO_1_1;
MO_digestOptions(H_NULL);
```

8.1.1.2. MO_option Array and Settings

Index	Setting	Description
0	MO_OPT_CURRENT_FOLDER	
1	MO_OPT_SPLIT_SCREEN	D_MODE_SPLIT_FULL=1, D_MODE_SPLIT_HORIZONTAL, D_MODE_SPLIT_VERTICAL
2	MO_OPT_NUMBER_OF_GRAPHS	D_MODE_GRAPHS_1=1, D_MODE_GRAPHS_2

Index	Setting	Description
3	MO_OPT_GRAPH_TYPE_1	D_GRAPH_TYPE_FUNCTION=1, D_GRAPH_TYPE_PARAMETRIC, D_GRAPH_TYPE_POLAR, D_GRAPH_TYPE_SEQUENCE, D_GRAPH_TYPE_3D, D_GRAPH_TYPE_DIFF_EQUATIONS
4	MO_OPT_GRAPH_TYPE_2	D_GRAPH_TYPE_FUNCTION=1, D_GRAPH_TYPE_PARAMETRIC, D_GRAPH_TYPE_POLAR, D_GRAPH_TYPE_SEQUENCE, D_GRAPH_TYPE_3D, D_GRAPH_TYPE_DIFF_EQUATIONS
5	MO_OPT_SPLIT_1	The app ID which is implemented as the memory handle of the first application's Application Control Block.
6	MO_OPT_SPLIT_2	The app ID which is implemented as the memory handle of the second application's Application Control Block.
7	MO_OPT_SPLIT_RATIO	D_SPLIT_RATIO_1_1=1, D_SPLIT_RATIO_1_2, D_SPLIT_RATIO_2_1
8	MO_OPT_ANGLE	D_ANGLE_RAD=1, D_ANGLE_DEG
9	MO_OPT_PRECISION	D_PREC_AUTO=1, D_PREC_RATIONAL, D_PREC_APPROX
10	MO_OPT_FIX	D_PREC_FIX_0=1, D_PREC_FIX_1, D_PREC_FIX_2, D_PREC_FIX_3, D_PREC_FIX_4, D_PREC_FIX_5, D_PREC_FIX_6, D_PREC_FIX_7, D_PREC_FIX_8, D_PREC_FIX_9, D_PREC_FIX_10, D_PREC_FIX_11, D_PREC_FIX_12, D_PREC_FLOAT, D_PREC_FLOAT_1, D_PREC_FLOAT_2, D_PREC_FLOAT_3, D_PREC_FLOAT_4, D_PREC_FLOAT_5, D_PREC_FLOAT_6, D_PREC_FLOAT_7, D_PREC_FLOAT_8, D_PREC_FLOAT_9, D_PREC_FLOAT_10, D_PREC_FLOAT_11, D_PREC_FLOAT_12
11	MO_OPT_NUMBER_FORMAT	D_EXP_FORMAT_NORMAL=1, D_EXP_FORMAT_SCI, D_EXP_FORMAT_ENG
12	MO_OPT_VECTOR_FORMAT	D_VECT_RECT=1, D_VECT_CYL, D_VECT_SPH
13	MO_OPT_COMPLEX_FORMAT	D_COMPLEX_OFF=1, D_COMPLEX_RECT, D_COMPLEX_POLAR

Index	Setting	Description
14	MO_OPT_PRETTY_PRINT	D_OFF=1, D_ON
15	MO_OPT_BASE	D_DEC=1, D_HEX, D_BIN
16	MO_OPT_UNIT_SYSTEM	D_UNIT_SI=1, D_UNIT_US, D_UNIT_CUSTOM
17	MO_OPT_UNIT_DEFAULTS	D_UNIT_DEFAULTS=1
18	MO_OPT_LANGUAGE	1 for English or AppId of language app.

8.2. Switching to the Home Screen

Under certain circumstances such as low memory, an application may need to quit and default back to the Home screen application. If the calculator is in full screen then simply exiting the application using **EV_quit** will cause the Home screen application to start. If the calculator is in split screen, then the mode setting for the split screen side the application is running on needs to be set to the Home screen application's ID. The following example demonstrates how to quit the current application and switch to the Home screen:

```
volatile HANDLE HomeID = H_NULL;
MO_currentOptions();

/* If in full screen, just quit to switch to home */

if (MO_option[MO_OPT_SPLIT_SCREEN] == D_MODE_SPLIT_FULL)
    EV_quit();
else  {

   /* If in split screen, set the appropriate side to home. */
   /* AMS behavior is such that if home was already on one side,
      it will become full screen. */

   HomeID = EV_getAppID( (const UCHAR *) "TIHOME" );
   if (MO_option[MO_OPT_SPLIT_1] == appID)
      MO_option[MO_OPT_SPLIT_1] = HomeID;
   if (MO_option[MO_OPT_SPLIT_2] == appID)
      MO_option[MO_OPT_SPLIT_2] = HomeID;
   MO_digestOptions(H_NULL);
}
```

8.3. Catalog

8.3.1. Built-in Functions and Commands

Pressing the CATALOG key displays the operating system's built-in functions and commands. As the cursor is moved through the list, help for the particular function or command is displayed in the status line as shown in Figure 8.1. F1: Help on the Catalog toolbar displays a catalog help dialog box that contains the help message from the status line (including text that may have been truncated due to the size limitations of the status line) as shown in Figure 8.2.

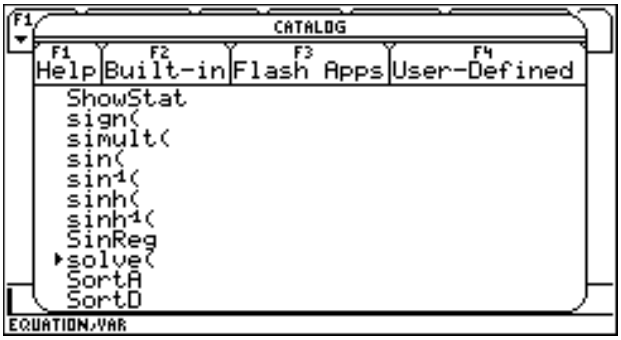


Figure 8.1: Catalog

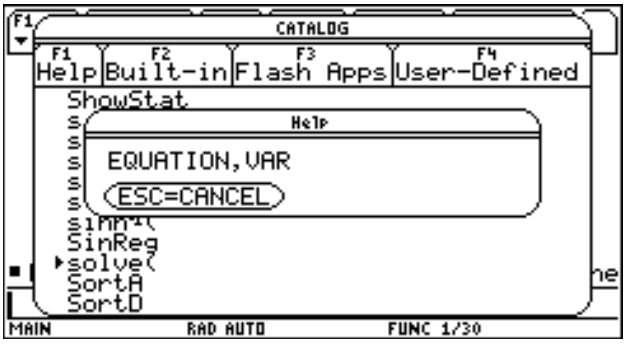


Figure 8.2: Catalog Help Dialog

8.3.2. User-Defined Functions and Programs

User-defined functions and programs can utilize the status line help and F1: Help by placing a comment as the first statement of the function or program as shown in Figure 8.3. The comment text will be the help message displayed in the status line and in the F1: Help dialog. Once in the catalog, if any user-defined functions or programs exist then the F4: User-Defined catalog toolbar selection item will be available. The F4 function key will display the user-defined functions and

programs. If the program or function has a comment as its first statement, then the comment text will be displayed on the status line as the cursor is moved through the list (see Figure 8.4). Pressing the F1: Help function key will display the comment text in a catalog help dialog box as shown in Figure 8.5. The list of user-defined functions and programs is displayed in alphabetical order by the program or function name. The folder name where the function or program is located is displayed to the right of the function or program name.

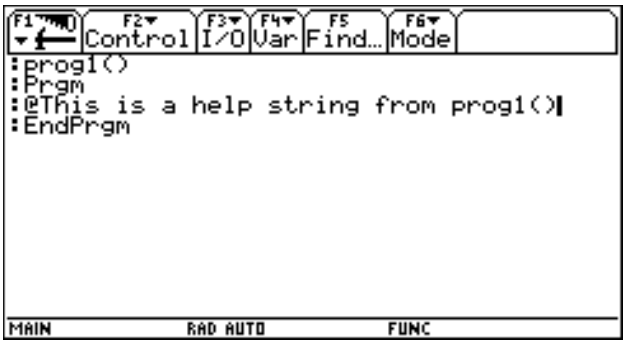


Figure 8.3: User Program

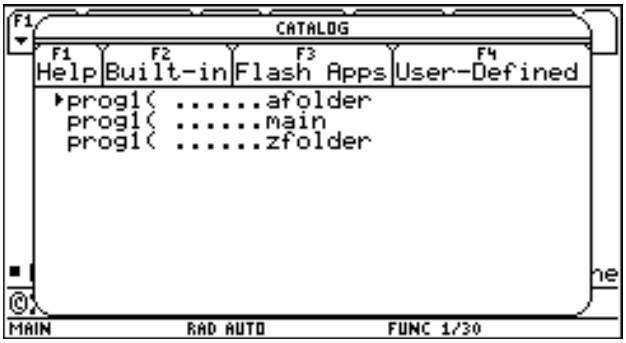


Figure 8.4: User-Defined Catalog

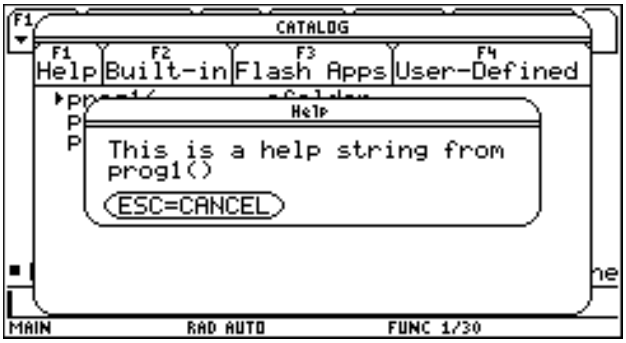


Figure 8.5: Help Dialog for User-Defined Catalog

8.3.3. Flash App Extensions

Flash applications can install functions and programs called App Extensions that are available to the rest of the system. The F3: Flash Apps catalog toolbar selection item will be available if any of the Flash applications loaded in the calculator installed App Extensions. Pressing the F3 function key will display the list of App Extensions in alphabetical order by function name with the name of the application that installed the App Extension to the right of the function or program name.

An example of defining App Extensions and App Help Strings within an application follows:

```
/* Applet strings */
#define STR_appfunc                0
#define STR_appprog                1
#define STR_HELP                 (100)

void main(pFrame, PEvent);

APP_EXTENSION const appExtensions[] =
{
   /* func name #,          help string #                    func index */
   {OO_APPSTRING+STR_appfunc, OO_APPSTRING+STR_HELP+STR_appfunc, STR_appfunc},
   {OO_APPSTRING+STR_appprog, OO_APPSTRING+STR_HELP+STR_appprog, STR_appprog},
};

APP_EXT_ENTRY const appExtEntries[] =
{
   {appfunc, APP_EXT_FUNCTION},
   {appprog, APP_EXT_PROGRAM},
};

FRAME(appObj, OO_SYSTEM_FRAME, 0, OO_APP_FLAGS, 11)
   ATTR(OO_APP_FLAGS, APP_INTERACTIVE)
   ATTR(OO_APP_NAME, "App")
   ATTR(OO_APP_TOK_NAME, "TIAPP")
   ATTR(OO_APP_PROCESS_EVENT, &main)
   ATTR(OO_APP_EXT_COUNT, 2 )
   ATTR(OO_APP_EXTENSIONS, appExtensions)
   ATTR(OO_APP_EXT_ENTRIES, appExtEntries)

   /*  The STR_appfunc string "appfunc" appears in the F2 Flash App
       catalog listing with the  OO_APP_TOK_NAME "TIAPP" -
       appfunc( . . . TIAPP   */

   ATTR(OO_APPSTRING+STR_appfunc, "appfunc")
   ATTR(OO_APPSTRING+STR_appprog, "appprog")

   /*  The STR_HELP+STR_appfunc string "app function help" appears in the
       Status Line when the cursor is on this function and also in the F1
       Help Dialog box  */

   ATTR(OO_APPSTRING+STR_HELP+STR_appfunc, "app function help")
   ATTR(OO_APPSTRING+STR_HELP+STR_appprog, "app program help")
ENDFRAME
```

8.4. Interfacing with TI-BASIC

An app can directly call many of the TI-BASIC commands. These commands begin with “cmd_” and are included in **Appendix A**. The C callable TI-BASIC functions or operators usually begin with “push_”. Examples include clearing the program I/O screen with the **cmd_clrio** routine and writing to the program I/O screen using the **cmd_disp** routine. An example function is **push_getfold** which is the TI-BASIC getFold function. TI-BASIC uses the expression stack (estack) to pass and return parameters. See chapter **15. Expressions and the Expression Stack** for a description of this stack and its contents. For a description of the data types used by TI-BASIC see chapter **14. Data Types**.

An app can also embed entire TI-BASIC programs or data. In fact, an app can consist of essentially nothing but TI-BASIC programs and data as shown in the following example. This example loads a TI-BASIC program using its StoProg function (which can store any TI-BASIC data item). The tokenized version of the program is in the prog1Data array. This array came from the TI-GRAPH LINK™ file containing the program. The program is run by using RunProg. The example program merely displays “Test” to the Program I/O screen. After the program runs, it is deleted. The TIBASIC_run function handles all errors including the ON key (to break the app). In this way, the user of the app does not even know a TI-BASIC program is running.

```
/*  BASIC -> APP Wrapper  */

#include "product.h"
#include "tiams.h"

static void AP_app(pFrame self, PEvent e);
#define ProgEnd(Prog)  (Prog+(sizeof(Prog) - 1))

FRAME(appObj, OO_SYSTEM_FRAME, 0, OO_APP_FLAGS, 3)
   ATTR(OO_APP_FLAGS, APP_INTERACTIVE)
   ATTR(OO_APP_NAME, "prog1" )
   ATTR(OO_APP_PROCESS_EVENT, &AP_app)
ENDFRAME

pFrame pAppObj = (pFrame)&appObj;
runningBASIC = FALSE;

const BYTE prog1Data[]={0X00,0X1D,0XE9,0X12,0XE4,0X00,0XE8,0XE5,0X51,
0XE4,0X02,0XE8,0XE5,0X00,0X54,0X65,0X73,0X74,0X00,0X2D,0X7A,0XE4,
0X02,0XE8,0X19,0XE4,0XE5,0X00,0X00,0X00,0XDC};

/* Run a BASIC program */
HANDLE TIBASIC_run(HANDLE hProgram, SINT *errNo)
{
   Access_AMS_Global_Variables;
   EStackIndex savetop = top_estack;
   USHORT errOffset;
   HANDLE hTokenized = H_NULL;
   HANDLE hResult = H_NULL;
```

```
   *errNo = ER_OKAY;
   TRY
      TRY
         runningBASIC = TRUE;
         if (NG_tokenize(hProgram, errNo, &errOffset))  {
            hTokenized = HS_popEStack();
            NG_execute(hTokenized, NG_DONT_APPROXIMATE);
            if (top_estack != savetop)
               hResult = HS_popEStack();
         }
      FINALLY
         runningBASIC = FALSE;
         HeapFreeIndir(&hTokenized);
         top_estack = savetop;
         reset_control_flags();
      ENDFINAL
   ONERR
      *errNo = errCode;
   ENDTRY
   return hResult;
}


/* Store Prog to symbol table, returning hVal of program if success. */
HANDLE StoProg( char *ProgName, const BYTE *Prog )
{
   BYTE nameBuf[MAX_SYM_LEN];
   HSYM hsym;
   SYM_ENTRY *SymPtr;
   HANDLE hVal = H_NULL;
   WORD ProgSize = (Prog[0] * 256 + Prog[1]) + 2;

   TRY
      if (FS_OK == TokenizeName( ProgName, nameBuf )) {
         cmd_delvar( TokNameRight(nameBuf) );
           if (hsym=VarStore(TokNameRight(nameBuf), STOF_NONE,
               ProgSize, ProgEnd(Prog)))
         {
            if (SymPtr = DerefSym(hsym)) {
               if (hVal = SymPtr->hVal)
                  memcpy( HeapDeref(hVal), Prog, ProgSize );
            }
         }
      }
   ONERR
      ERD_dialog( errCode, FALSE );
      return 0;
   ENDTRY
   return hVal;
}

BOOL RunProg( char *ProgName )
{  HANDLE hCommand, hResult;
   char *pCommand;
   SINT errCode;
```

```
   if (hCommand = HeapAlloc(80)) {
      strcpy(pCommand = HeapDeref(hCommand), ProgName );
      strcat(pCommand, "()" );
      if (hResult = TIBASIC_run(hCommand, &errCode))
         HeapFree(hResult);
      HeapFree(hCommand);
      if (errCode)  {
         ERD_dialog( errCode, FALSE );
         return(FALSE);
      }
   }
   return TRUE;
}

void DelVar( char *varName )
{
   BYTE nameBuf[MAX_SYM_LEN];

   TRY
      if (FS_OK == TokenizeName( varName, nameBuf ))
         cmd_delvar( TokNameRight(nameBuf) );
   ONERR
   ENDTRY
}

static void AP_app(pFrame self, PEvent e)
{
   HANDLE hVal;

   switch (e->command) {
      case CM_ACTIVATE:
         if (!runningBASIC) {
         push_quantum( END_TAG );
         EV_defaultHandler(e);
         if (StoProg( "prog1", prog1Data ))
            RunProg( "prog1" );
         DelVar( "prog1" );
         EV_quit();
         }
         break;
      case CM_QUIT:
         break;
      default:
         EV_defaultHandler(e);
         break;
   }
}
```

The previous example showed how to run a TI-BASIC program from an application. If an application just needs to execute a function or expression to return some value then the following example will do that. The function tCmdLineDriver below inputs a string from the user using a dialog box. It then calls the function CmdLine to execute that string and return a value. The return value is stored to a global variable, E1. If there is an error then an error dialog is displayed. The CmdLine function will evaluate the string and return a value on the estack. It will not evaluate anything that has side effects — that is the

expression to be evaluated may not store to any global variables, perform any kind of I/O (displaying values on the Home screen, graphing, and so on), or cause a switch to another application.

```
/* Evaluate the expression pointed to by pExpr (must be locked if in
   the heap).
*/
SINT CmdLine( char *pExpr )
{  Access_AMS_Global_Variables;
   SINT errNo;

   SET_SIDE_EFFECTS_FORBIDDEN; // No I/O or programs
   TRY
      push_quantum( END_OF_SEGMENT_TAG );
      push_parse_text( (BYTE *) pExpr );
      push_simplify_statements( top_estack );
      errNo = ER_OKAY;
   ONERR
      errNo = errCode;
   ENDTRY
   SET_SIDE_EFFECTS_PERMITTED;
   return errNo;
}

void tCmdLineDriver( void )
{  Access_AMS_Global_Variables;
   SINT errNo;
   EStackIndex saveTop;
   BYTE e1[] = {0,'e','1',0};
   char szBuf[256];

   memset( szBuf, 0, sizeof(szBuf) );
   strcpy( szBuf, "solve(x*x+3x-3=0,x)" );
   while (KB_ENTER == Dialog( &dGetStr,-1,-1, szBuf, NULL)) {
      saveTop = top_estack;
      if (errNo = CmdLine( szBuf ))
         ERD_dialog( errNo, FALSE );
      else {
         DlgNotice( "OK", "Answer stored in 'e1'" );
         VarStore( e1+3, STOF_ESI, 0, top_estack );
      }
      top_estack = saveTop;
   }
}

/*  Resource for tCmdLineDriver  */
DIALOG dGetStr, 0, 0, NoCallBack
{
   EDIT,   {0, 8, 15}, "", 0, 251, 34
   HEADER, {0, 0, 0}, "Enter expression, ESC to exit", PDB_OK, PDB_CANCEL
   XFLAGS, {0, 0, 0}, XF_ALLOW_VARLINK | XF_VARLINK_SELECT_ONLY, 0, 0, 0
}
```

In the CmdLine example above, the SET_SIDE_EFFECTS_FORBIDDEN and SET_SIDE_EFFECTS_PERMITTED macros can be removed in order to run TI-BASIC commands. This will cause no problems for an app if the user enters define or store commands. Or if the app itself issues the commands, it can know

ahead of time which commands it will execute. Commands that will cause a problem are those such as Graph or DspTbl. These commands, along with several others, cause a context switch which the app must handle. See the example app in section **8.4 Interfacing with TI-BASIC** (specifically the runningBASIC flag) for how to handle the context switch.

For an example on executing a command with side effects, see section **17.2 Working with the Graph Application**.

8.5. Verifying the OS Version

Version 2.04 of the AMS introduced F-Line instructions to call the API. This requires an app to run on Operating System Version 2.04 and later but can reduce every API call from six bytes to two bytes. This is the standard calling mechanism when using the tiams.h include file. Accessing AMS global variables requires the use of the **Access_AMS_Global_Variables** macro and each AMS global variable reference requires six bytes plus the overhead from the **Access_AMS_Global_Variables** macro.

In order to ensure your app is running in AMS 2.04 and later put a call to the **OS_NeedMinimumVersion** macro at the top of your app's event handler. It needs to be called before any F-Line instructions (an AMS 2.04 feature) are executed. Its format is:

OS_NeedMinimumVersion (*frame*, *major*, *minor*)

frame — The name of the variable which contains the address of your application frame.

major — Major version number of required OS level.

minor — Minor version number of required OS level.

For example, if you have a pointer to your app frame named pAppObj:

```
pFrame pAppObj = (pFrame)&appObj;
```

Then at the top of your event handler entry point call:

```
OS_NeedMinimumVersion(pAppObj, 2, 4);
```

This checks for OS release 2.04 or greater. If the OS is an earlier version, a flag is set in the app's ACB disabling the app so it will not appear in the app's menu. This macro returns to the OS without letting the rest of the event handler run.

Be aware that some F-Line calls may not be immediately apparent. If an app uses a library or API routine (say a long divide or any API call) in its local variable initialization, an F-Line may be inserted to call the library routine and is always used to call the API.

F-Lines can also be used to convert long calls or jumps to relative jumps or calls which reduces the app's relocation table and hence the app's size. See section **7.1 File Format** for a description of the relocation table.

8.6. Optimizing Code Space

One of the optimizations involves reducing the apps relocation table. All references to an app's global variables made by an app require a relocation entry to be stored with the app. If there are multiple references to a particular global variable in an app, the global references can be replaced with local pointers as shown in the example below.

```
static WINDOW appW;
static WIN_RECT appWRect;

void AP_EventHandler(pFrame self, PEvent e) {
   WINDOW *winPtr = &appW;

   switch (e->command) {
      case CM_START:
         appWRect = *(e->info.startInfo.startRect);
         if (WinOpen( winPtr, &appWRect, WF_TTY  | WF_DUP_SCR))
            WinClr( winPtr );
         else
            EV_quit();
         break;
      case CM_ACTIVATE:
         EV_defaultHandler(e);
         WinBeginPaint( winPtr );
         WinActivate( winPtr );
         WinStr( winPtr, "Just activated\n" );
         break;

   .
   .
   .

   }
}
```

In the preceding example, since there were several references to the global variable **appW** the pointer winPtr was initialized to the address of **appW** at the entry point to the **AP_EventHandler** routine and used instead of &appW. Since there was only one additional reference to **appWRect** an additional pointer to access that global was not created.

8.7. VAR-LINK

The VAR-LINK screen can be activated by pressing the [VAR-LINK] key or directly by using the **handleVarLinkKey** routine (**Appendix A**). By default, VAR-LINK is disabled within a dialog box. It can be enabled by using the XF_ALLOW_VARLINK flag. See section **11.4 Dialog Boxes** for further details.

VAR-LINK can also view files (3rd party data types) generated by an app. An app must have an OO_APP_VIEWER entry in its frame to specify a routine to view files (see chapter **7. Flash Application Layout** for a description of FRAME attributes). The prototype for the app viewer is as follows:

BOOL **LocAppViewer** (AppID *appID*, BYTE * *type*, WINDOW * *vlWin*, HSYM *hSym*)

appID — The app's ID.

type — A pointer to a one to four character string containing the file type of the file to be displayed.

vlWin — Pointer to a WINDOW structure that the app can draw to display the contents of the file.

hSym — The HSYM of the file to display.

If the app handles the specific view request it must return TRUE, if not it should return FALSE and the next app in the list of apps will be given a chance to view the file. See the **FOpen** routine for an example app viewer.

In terms of sample code see the **VarCreateFolderPopup** routine for the source to VAR-LINK's F2 (view) key.

9. Application Control Flow

9.1. Event-Driven Architecture

The TI AMS Operating System (OS) implements an event-driven architecture. After initializing the system, the operating system goes into a loop checking each hardware device for an event such as a keypress or clock tick. When a device indicates it needs processing, the OS packages information about the event into a message and sends it to the currently active application.

If none of the devices need attention, the window list is scanned for a dirty window, that is to say, a window which needs to be repainted. A message (see CM_WPAINT in section **9.3. Commands**) is sent to the window's owner application indicating it needs to repaint its window.

After all dirty windows have been redrawn, a null message (CM_NULL) is sent to the current application. Then the calculator is put into low power idle mode. Any hardware interrupt brings the calculator out of low power mode and starts the event scan at the top of the loop.

An application receives messages from the OS through its main entry point. A simplified overview of an application's main entry point follows:

```
#include "tiams.h"

TERecord terec;
AP_myApp(pFrame self, Event *event)
{
   switch (event->command)
   {
   case CM_START:
      .
      .
      .
      break;

   case CM_KEY_PRESS:
      .
      .
      .
      if (! TE_handleEvent(&terec, event))
         EV_defaultHandler(event);
      break;

   case CM_WPAINT:
      .
      .
      .
      break;
```

```
   default:
      EV_defaultHandler(event);
      break;
   }
}
```

The application's main entry point has two parameters — a pointer to the application's frame and a pointer to an event notification structure. The application typically utilizes a `switch` statement to decode what kind of event it received. The appropriate case label then acts on the event accordingly.

Events which are not picked out by any case label are given default handling by the routine **EV_defaultHandler**. The application may choose to modify the default behavior of some events by acting on them then passing them on to **EV_defaultHandler**.

The application relinquishes control to the operating system after handling an event by returning from its main entry point.

9.2. Event Structure Layout

```
typedef struct SEvent
{
   UINT command;
   UINT sourceID;
   UINT side;
   UINT status;
   union
   {
      EventInfo eventInfo;
      KeyInfo keyInfo;
      PasteInfo pasteInfo;
      PasteHandleInfo pasteHandleInfo;
      PaintInfo paintInfo;
      StartInfo startInfo;
      MenuInfo menuInfo;
      ModeInfo modeInfo;
   } info;
} Event, *PEvent;
```

command — A command number encoding which event occurred. This may be, among others, a keyboard event, a message from the window system, or an application-to-application message. See the next section (**9.3. Commands**) for details about each command type.

sourceID — The ID of the application which originated the event—usually the application which is currently running.

side — Which side of the screen the application's window occupies. An application sharing the screen with another application in split screen mode can tell which side of the screen it is on, either AP_SIDE_A for the top or left side, or AP_SIDE_B for the bottom or right side.

status — A copy of many of the status flags when the event occurred. This includes the state of the 2nd, ♦, and ⬆ keyboard modifier keys, angle mode setting, busy indicator, and so forth.

info — Event dependent information. It may specify keypress information, string paste pointer or handle, the address of which window to repaint, application start up information, or new mode settings.

9.3. Commands

The type of an event can be identified by the contents of the command field of the `Event` structure. Command numbers have symbolic names defined in tiams.h.

0x001 – 0x4FF Built-in strings

Character strings are kept together in a table for ease of language customization. The menu system uses command numbers in this range as an index into the string table. The default event handler converts these commands into CM_PASTE_STRING events containing the cross-referenced string pointer.

0x500 – 0x6FF Application-specific commands

Menu choices specific to each application.

0x700 – 0x7BF System commands

OS-generated commands.

0x700 CM_NULL

Sent to the active application when there are no other events to process. The application might use this event to take care of some background processing or update an animated display.

0x701 CM_INIT

Sent to each application once when the calculator is reset or batteries are inserted. This command is used principally by the built-in applications. The CM_INSTALL command is better suited for initializing Flash applications.

0x702 CM_START

Sent to an application when it is being started. Included in the start message is a rectangle with the requested window location and size chosen by the user from the mode screen settings. This rectangle is passed into **WinOpen** to create the application's initial window. Some applications display a Current/Open/New submenu on the APPS menu. The `startCode` field of the start message (*StartInfo*) tells the application which submenu command the user chose.

0x703 CM_ACTIVATE

More than one application may be visible on the screen, but only one at a time can interact with the user. The activate message is sent to an application to designate it as the interactive center of attention. Most event messages are directed to the active application until it receives a deactivate message.

The active application should highlight its window border to make it apparent to the user which window is active.

The default event handler displays the application's registered menu.

0x704 CM_FOCUS

Tells the application to turn on its cursor flash. This message is normally handled by the text editor event handler. See **TE_handleEvent**.

0x705 CM_UNFOCUS

Tells the application to turn off its cursor flash. This message is normally handled by the text editor event handler. See **TE_handleEvent.**

0x706 CM_DEACTIVATE

Sent to the active application to inform it that it no longer holds the interactive center of attention.

The default event handler calls **MenuEnd** to release registered menu memory.

0x707 CM_QUIT

Tells the application to prepare to quit. The application should save its data and release any memory handles it allocated.

0x708 CM_RESTART

Notifies the current application that the user has chosen to start the same application from the APPS menu. Applications may choose to ignore this command and there is no default handling.

0x709 Reserved

0x70A Reserved

0x70B CM_ON

Sent to each application when the calculator is turned on after being turned off with ([2ND] [OFF]). This command is not sent if the calculator is turned on after automatic power down.

0x70C CM_INSTALL

Sent to an application just after it has been installed in the calculator. This message is sent to applications which are already in Flash memory when batteries are inserted or when the calculator is reset. This message is also sent to an application just after it has been downloaded into Flash memory through the link port.

0x70D CM_UNINSTALL

Sent to an application just before it is deleted from Flash memory.

0x70E CM_PACK

Informs an application that it is about to be moved to another address in Flash memory. The OS sends this message before it begins garbage collection after deleting another Flash application. The application must save any state information it needs.

0x70F CM_UNPACK

Sent to an application after Flash memory garbage collection is complete. The application uses this opportunity to restore state information it saved when it received the CM_PACK message.

0x710 CM_KEY_PRESS

Sent when a key is pressed on the calculator keyboard. This message tells the application which key was pressed. Keypresses include ASCII characters (0x20 – 0x7E), extended ASCII characters (0x80 – 0xFF), control characters and special symbols (0x00 – 0x1F, and 0x7F), and extended key codes (>= 0x100) such as function keys and the cursor arrow keys. The application usually forwards extended key codes to the default event handler for further processing.

0x720 CM_CUT

Cut selected text to the clipboard. This and the following text editing commands are generated by menu choices or translated from CM_KEY_PRESS commands by the default event handler. Text edit commands are usually handled by the default text edit handler. See **TE_handleEvent**.

0x721 CM_COPY

Copy selected text to the clipboard.

0x722 CM_PASTE

Paste text in clipboard to edit buffer at cursor position. Any selected text is replaced.

0x723 CM_PASTE_STRING

Paste text from a string. The event message includes a pointer to the string to paste.

0x724 CM_PASTE_HANDLE

Paste text from a handle. The event message includes a handle to the string to paste. The handle is released back to the heap by the default text edit handler after the paste is complete.

0x725 CM_DELETE

Delete selected text. If no text is selected, delete one character to the left of the cursor.

0x726 CM_CLEAR

Clear selected text. If no text is selected, clear from cursor to end of edit buffer. If cursor is at end of edit buffer, clear all text from edit buffer.

0x727 CM_CLEAR_ALL

Clear everything. The application decides what it means to clear everything.

0x730 CM_TOGGLE_INSERT

Switch between text insert mode and overstrike mode. This command is implemented in the default text edit handler.

0x740 CM_CURSOR_FLASH

Show or hide the text cursor. This command is generated every half second by the timer.

0x750 CM_STO

Store key [STO▸] was pressed. The default event handler translates this command into the → character.

0x751 CM_RCL

Recall key [RCL] was pressed. The default event handler displays a dialog box for the user to enter the name of a variable to recall. The chosen variable's contents are pasted at the edit cursor.

0x760 CM_WPAINT

Sent to an application when one of its windows needs to be repainted. The event message includes a pointer to the window which needs updating.

0x770 CM_OPEN

Open variable. This and the following commands are sent when the user chooses one of the commands from the Tools menu. The application should save the variable it is working on and prompt the user for the name of another variable to open. See **VarOpen**.

0x771 CM_SAVE_AS

Save application data in a variable. The application prompts the user for the name of a new variable. See **VarSaveAs**.

0x772 CM_NEW

Create a new empty variable. The application should save the variable it is working on and prompt the user for the name of a new variable to create. See **VarNew**.

0x773 CM_FORMAT

Prompt the user for application preferences.

0x774 CM_ABOUT

Display information about the application.

0x780 CM_MODE_CHANGE

Sent to every application when mode settings have been changed on the MODE screen or by the TI-BASIC setMode function. Flags in the event message indicate which mode settings changed.

0x781 CM_SWITCH_GRAPH

Sent to every application when the user switches between graphs in two-graph mode.

0x782 CM_DEFAULTS

Sent to every application when the user selects
F1: Reset, 1: RAM, 2: Default from the MEMORY screen. Each application should reset its preferences back to its factory settings when it receives this message.

0x7C0 – 0x7FF Interapplication messages

Some of the built-in applications send messages to each other with this range of commands, but this means of communicating between applications is not recommended. Interapplication messaging is largely replaced with an object-oriented approach. Applications communicate with each other through their frame interface.

0x800 – 0xFFF Application string numbers

Applications should index the text of their menus, dialog boxes, and error messages in this range of command numbers.

9.4. Starting and Stopping an Application

The calculator OS starts an application by sending it the CM_START message. A field in the start message points to a window rectangle. This rectangle defines the window location and size previously established by the user with split window mode settings. The application should pass this rectangle to **WinOpen** to create its initial window. This is also a good time to initialize data structures.

The APPS menu displays a submenu of start up options for some applications. The start message tells the application which option the user chose: Current, Open . . . , or New . . . .

The OS then sends the application a CM_ACTIVATE message. The activate message tells the application that it is now the current active application. The application should build and display its dynamic menu or pass the event message to **EV_defaultHandler** to display its static menu. The application should also call **WinActivate** to highlight its window border.

Finally, the OS sends the CM_FOCUS message. The application usually lets text edit or default event handling process this event. If, however, the application cannot start for some reason, this is the time to deal with it. It is only after this third message is received that the application can force a quit and return to the Home screen if there is insufficient memory or some other condition is incorrect for the application to start normally.

The above three messages, CM_START, CM_ACTIVATE, and CM_FOCUS, are sent at the start of every application. The application now begins to receive a stream of events corresponding to user inputs.

When the user chooses another application from the APPS menu, the OS terminates the current application by sending it three messages, CM_UNFOCUS, CM_DEACTIVATE, and CM_QUIT in that order.

Text edit usually handles the CM_UNFOCUS message. The application may ignore this message if it does not have an open text edit field.

When the application receives the CM_DEACTIVATE message, it should call **WinDeactivate** to unhighlight its window, then forward the message to the default event handler which releases menu memory resources.

The CM_QUIT message tells the application to save the user's work and close its window (**WinClose**).

If the user has the calculator in split screen mode, he can switch the focus to the application on the other side of the screen. When this happens, the application receives CM_UNFOCUS and CM_DEACTIVATE messages in that order. When the user switches focus back to the application, it receives CM_ACTIVATE and CM_FOCUS messages.

The user may select the same application from the [APPS] key as the application already running. The application receives the CM_RESTART command. This is important for applications which have a Current/Open/New submenu on the APPS menu. The user's choice from the submenu is sent in a field of the restart event message.

9.5. Keyboard Events

The application receives a CM_KEY_PRESS message when the user presses a key on the calculator keyboard. The message includes which key was pressed.

The OS provides default behavior for most keypress messages. The application should check for and process significant keypresses. Keypresses the app does not understand should be passed to **EV_defaultHandler**, which implements system wide behavior for keys such as [MODE], [APPS], [CATALOG], etc.

The application can pass keypress messages to the text editor's event handler (**TE_handleEvent**) if it has an active text edit field in its window. **TE_handleEvent** returns TRUE if it acted on the event. The application should test the return value of **TE_handleEvent** and pass the message to **EV_defaultHandler** if the text editor did not act on the message.

9.6. Menu Processing

Each item in a menu has associated with it a command number. The OS uses the command number to communicate to the application which menu item the user chose. Menu processing proceeds as follows.

1. The user invokes menu processing by pressing one of the function keys [F1] . . . [F8].
2. The OS sends the function-key press ([F1] . . . [F8]) to the application as a CM_KEY_PRESS event.
3. The application forwards the event to the default event handler (**EV_defaultHandler**).
4. The default event handler looks up the application's current menu (attribute OO_APP_DEFAULT_MENU_HANDLE in the application's frame).
5. The default event handler calls **MenuKey** on the application's current menu to start user interaction with the menu.
6. **MenuKey** returns the command number of the user's chosen menu item.
7. Default event handler sends the command number as an event to the application.

> **Note:** The application's event handler entry point is called recursively by the above process, first with the CM_KEY_PRESS message then a nested call with the menu item command number. Applications must be re-entrant since default event handling often entails translating one type of command into another type. The application receives the translated message through its event handler entry point as a recursive call from the default event handler.

The OS automatically processes function-key presses ([F1] . . . [F8]) only if the application has placed a menu handle where the default event handler can find it. The OS looks for a handle to the current menu in the OO_APP_DEFAULT_MENU_HANDLE attribute of the application's frame. The software developer can construct static menus with the resource compiler and link them to the application when the application is created, or an application can build a dynamic menu at run time.

9.6.1. Static Menus

Static menus are easy to create and simple to use. If your application has very modest menu requirements, static menus are the better choice. Since static menus are handled transparently by the default event handler, the application needs no extra code to deal with function keypresses. Menus are automatically drawn when the application is activated, function keypresses are passed to the menu system, and menu memory is released when the application is deactivated. Incidentally, all the built-in applications employ static menus.

See section **11.5. Resource Compiler** on how to compose a menu source file and use the resource compiler to create an object file suitable for linking with your application.

The application should be compiled with the address of its menu in its OO_APP_DEFAULT_MENU frame attribute.

When an application receives a CM_ACTIVATE message, default event handling checks the application's OO_APP_DEFAULT_MENU attribute. If it finds a pointer to a menu there, it calls **MenuBegin** to draw the menu across the top of the calculator screen, and saves a handle to the menu in application frame attribute OO_APP_DEFAULT_MENU_HANDLE.

Default event handling of CM_DEACTIVATE calls **MenuEnd** on the application's current menu and frees the menu handle.

9.6.2. Dynamic Menus

An application may need to change menu contents based on its current state. In this case, the application needs to take a more active role in building menus, managing menu memory, and setting up the user interface.

Creating a new menu — an application uses **MenuNew** and **MenuAddText** to create dynamic menus. The application then calls **SetAppDefaultMenuHandle** to place the new menu's handle in the application's frame where default event handling can find it for function-key processing.

Disposing of a menu — an application disposes of a menu by calling **MenuEnd** to release its memory resources and `SetAppDefaultMenuHandle(MY_APP_ID(MyAppObj), H_NULL)` to unregister the menu with the default event handler.

The application must be able to display a dynamic menu when the application is activated and change menus while the application is active. Default event handling automatically releases menu resources when the application is deactivated.

An application is activated when it receives the CM_ACTIVATE message from the OS. The application should respond to this message by creating a new menu as described above.

An application changes menus by disposing of the current menu and creating a new menu.

Default handling of the CM_DEACTIVATE event automatically frees the application's menu handle and sets attribute OO_APP_DEFAULT_MENU_HANDLE to H_NULL.

9.7. Paint Events

The CM_WPAINT message tells the application to repaint its window. The address of which window to paint is included in the message in case the application has more than one window. The application should call **WinBeginPaint**, then any other window drawing routines, then **WinEndPaint**.

9.8. Background Events

Applications can arrange to receive time for background processing. This allows an application to execute on a time-available basis even when it is not the focus of interactive events. The OS sends CM_BACKGROUND messages only to applications which have the APP_BACKGROUND flag set in the OO_APP_FLAGS attribute of their object frames (see section **7.3.1.3. Object Frame Attributes**).

CM_BACKGROUND messages are very low priority. Only after all device events, dirty window repaint messages, and the null event have been sent to the current application are background events sent out. Every application which has its APP_BACKGROUND flag set is then sent a CM_BACKGROUND message. Background applications will continue to get CM_BACKGROUND messages until the OS determines a higher priority message must be sent to the current application.

> **Note:** Applications should keep background processing short so as not to degrade the response of interactive applications.

9.9. Default Event Handler

The simplest application for the AMS Operating System does nothing more than forward its events to the default event handler, **EV_defaultHandler**. Such a simple application does not display a window and does not respond in any visible way to keypresses from the user. While a truly useful application must respond to some events, it is helpful to know what happens to events which the app does not handle. The action provided by the default event handler is, in many cases, already sufficient and needs no further elaboration in the application.

Many events have no default action and, unless they are listed in this section, are discarded when they get to the default event handler.

9.9.1. CM_KEY_PRESS

The info.keyInfo.keyCode field of the CM_KEY_PRESS event contains a value indicating which key the user pressed. Many keypresses are translated into a

string of characters and sent to the current application as a CM_PASTE_STRING command. Those keypresses and their translated string are listed below.

Keypress	Translated String
KB_ANS	Ans(1)
KB_SIN	sin(
KB_ASIN	$\sin^{-1}$(
KB_COS	cos(
KB_ACOS	$\cos^{-1}$(
KB_TAN	tan(
KB_ATAN	$\tan^{-1}$(
KB_LN	ln(
KB_ALN	e^(
KB_INV_X (TI-92 Plus only)	^-1
KB_INFINITY (TI-89 only)	∞
KB_UNDER_SCORE (TI-89 only)	_
KB_THETA (TI-89 only)	θ
KB_AMPER (TI-89 only)	&
KB_ATSIGN (TI-89 only)	@
KB_EXCLAM (TI-89 only)	!
KB_COPYRIGHT (TI-89 only)	©
KB_SIGMA (TI-92 Plus only)	Σ(
KB_INTEGRAL	∫(
KB_DIFF	d(
KB_ROOT	√(
KB_OPTION + '0'	≤
KB_OPTION + '='	≠
KB_OPTION + '.'	≥

Table 9.1: Keypress Translations

Several keypresses initiate special handling.

Keypress	Action
KB_STO	Sends CM_STO command to the current app.
KB_RCL	Sends CM_RCL command to the current app.
KB_F1 through KB_F8	Initiates menu processing using the menu registered by the current app. The menu item chosen by the user is sent as a command to the current app.
KB_SWITCH	Switches focus between apps on each side of screen, or between current and previous app.
KB_VARLINK	Displays the VAR-LINK screen. If the user presses ENTER on a variable name, the name is sent in a CM_PASTE_HANDLE message to the current app.
KB_CHAR	Displays the CHAR pop-up menu. The chosen character is sent in a CM_PASTE_HANDLE message to the current app.
KB_CATLG	Displays the CATALOG. The chosen function or command name is sent in a CM_PASTE_HANDLE message to the current app.
KB_UNITS	Displays the UNITS dialog box. The chosen unit is sent in a CM_PASTE_HANDLE message to the current app.
KB_MATH	Displays the MATH pop-up menu. The chosen function is sent in a CM_PASTE_HANDLE message to the current app.
KB_CUSTOM	Toggles the custom menu on and off.
KB_MODE	Displays and processes the MODE screen. If any mode settings are changed, a CM_MODE_CHANGE message is sent to every application.
KB_MENU	Displays the APPLICATIONS pop-up menu. If an application is chosen from the menu, the current app is terminated and the chosen app is started.
KB_FLASH_APPS	Displays the FLASH APPLICATIONS pop-up menu. If an application is chosen from the menu, the current app is terminated and the chosen app is started.
KB_MEM	Displays the MEMORY dialog box.
KB_INSERT	Sends the CM_TOGGLE_INSERT command to the current application.

Table 9.2 Keypress Actions

Keypress	Action
KB_QUIT	Terminates the current app and switches to the Home screen.
KB_COPY	Sends the CM_COPY command to the current app.
KB_PASTE	Sends the CM_PASTE command to the current app.
KB_CUT	Sends the CM_CUT command to the current app.
KB_DELETE	Sends the CM_DELETE command to the current app.
KB_CLEAR	Sends the CM_CLEAR command to the current app.
KB_OPEN	Sends the CM_OPEN command to the current app.
KB_SAVE_AS	Sends the CM_SAVE_AS command to the current app.
KB_NEW	Sends the CM_NEW command to the current app.
KB_FORMAT	Sends the CM_FORMAT command to the current app.
KB_HELP_KEYS	Displays a map of additional keyboard character translations.
KB_HOME	Starts the Home screen.
KB_YEQ	Starts the [Y=] editor.
KB_RANGE	Starts the Window screen.
KB_GRAPH	Starts the grapher.
KB_TBLSET	Displays the TABLE SETUP dialog box.
KB_TABLE	Starts the Table app.
KB_OFF	Switches to the Home screen and turns the calculator off.
KB_OPTION + KB_OFF	Turns the calculator off.

Table 9.2 Keypress Actions *(continued)*

9.9.2. CM_PASTE_STRING

The application usually passes this command to a text edit field. However, if the CM_PASTE_STRING is not handled by the application, the default event handler breaks the paste string up and feeds each character back to the app in CM_KEY_PRESS events.

9.9.3. CM_PASTE_HANDLE

The application usually passes this command to a text edit field. If the app does not handle this event, the default event handler breaks up the string in the handle and feeds each character back to the app in CM_KEY_PRESS events. The handle is then automatically freed after the last character has been sent to the current app.

9.9.4. CM_STO

This command sends a store character → in a CM_KEY_PRESS event to the current app.

9.9.5. CM_RCL

This command displays and processes the RECALL dialog box to get the name of a variable. It then sends the contents of the variable to the current app in a CM_PASTE_HANDLE event.

9.9.6. CM_DEACTIVATE

This command turns off the custom menu (**CustomEnd**) or the running app's menu (**MenuEnd**), whichever is active. This is part of the automatic menu handling described in section **9.6. Menu Processing**.

9.9.7. CM_ACTIVATE

This command turns on (**MenuBegin**) the running app's menu. This is part of the automatic menu handling described in section **9.6. Menu Processing**.

9.10. Installing, Moving, and Deleting an Application

The OS sends CM_INSTALL to an application after it is downloaded into Flash memory, and when the calculator is reset. The OS allocates RAM for the application's data segment, zeros uninitialized static variables and sets the values of initialized static variables. Any additional initialization which the application needs to perform once when it is installed should be done at this time.

The OS calls the AppNoticeInstall method of every application in the calculator when a new application is installed. App localizers use this notice to watch for applications which need to be overridden with local language string tables. See **AppNoticeInstall** application method in section **7.3.1.3.17. Method OO_APP_NOTICE_INSTALL (0x11)**.

Sometimes an application needs to be moved to another place in Flash memory. This can happen when another application is deleted and the OS garbage collects to free up unused Flash memory. The application receives a CM_PACK command before garbage collect begins then CM_UNPACK after garbage collect ends. The application's static data is reinitialized when this happens, hence the application needs to perform much the same initialization as if it received the CM_INSTALL message.

Alternatively, there is a four-byte location (`publicstorage`) in the application's ACB (Application Control Block) where the app can store a value during CM_PACK processing which can be retrieved during CM_UNPACK processing. If the application has a lot of data to save, it can allocate memory from the heap and store its handle in `publicstorage`. When the application receives the unpack message, it retrieves the memory handle from `publicstorage`, reinitializes its data, and releases the handle. Use routines **OO_appSetPublicStorage** and **OO_appGetPublicStorage** to save and retrieve your application's `publicstorage`.

Note:	An application will never receive pack/unpack messages while it is active. The application will always have been terminated (that is, received the CM_QUIT message) before it is moved.

The OS sends CM_UNINSTALL to an application as final notification when it is about to be deleted. Any memory handles the application allocated when it was installed or while it was active should be deleted at this time to prevent memory leaks.

10. Error Handling

This chapter describes the Advanced Mathematics Software (AMS) implementation of error handling — how to throw errors, why you might want to throw an error, how to catch errors, and how to clean up when an error occurs.

10.1. Throwing an Error

Your application or ASM program can signal exceptional conditions by throwing an error. Calling **ER_throwVar** in your app diverts execution to an error handler, typically the system error handler. **ER_throwVar** accepts one argument, an integer in the range 0 to 0xEFF. System error numbers range from 0x000 to 0x7FF. Application-defined errors begin at OO_FIRST_APP_STRING (0x800). Look in tiams.h for macros beginning with ER_ for predefined error numbers. An error message is associated with each predefined error number.

Perhaps the condition most frequently needing special attention is the case when **HeapAlloc** cannot fulfill a request for memory. **HeapAlloc** returns H_NULL if it cannot allocate the requested amount of memory. Your app should always test the return value of **HeapAlloc**. Under most circumstances, if your app cannot allocate the memory it needs, it should throw an error.

```
h = HeapAlloc(BUF_SIZE);
if (h == H_NULL)
   ER_throwVar(ER_MEMORY); /* error number defined in tiams.h */
```

The system error handler catches this error and displays a dialog box indicating there was a memory error.

> **Note:** Think of **ER_throwVar** as a long jump rather than a subroutine call. Execution does not return from the **ER_throwVar** call.

Some AMS routines may throw an error instead of returning an error code. **HeapAllocThrow**, for example, tries to allocate memory but throws the `ER_MEMORY` error automatically if it fails.

Macro **ER_throw** works like **ER_throwVar** but accepts only integer constants. The Sierra C™ compiler generates more compact code for the **ER_throw** macro.

10.2. Delayed Error Messages

You should not throw an error while processing events CM_START, CM_ACTIVATE, CM_FOCUS, CM_UNFOCUS, CM_DEACTIVATE, CM_QUIT, CM_WPAINT, CM_INSTALL, CM_UNINSTALL, CM_PACK, CM_UNPACK,

CM_INIT, or CM_MODE_CHANGE. If your app hits an error condition while processing one of these events, it should store an explanatory error code number in **EV_errorCode**, then return to the OS. The OS will then display the error message dialog box.

It is best to steer clear of signaling error messages while processing these events. If, however, you cannot avoid exceptions, keep in mind that **EV_errorCode** can hold only a single error code. If your app stores to **EV_errorCode** multiple times before returning to the OS, only the last error code will be displayed in an error dialog box.

10.3. Throwing Your Own Errors

Your application may have exceptional conditions which are not properly described by any of the built-in error messages. Error numbers beginning at OO_FIRST_APP_STRING (0x800 – 0xEFF) are available for application-specific errors. Whenever your code throws an error number in this range, the system error handler looks for the text of the error message in the current application.

Place the text of your error messages in the frame of your application beginning with attribute OO_FIRST_STRING. Then, if your app throws error number OO_FIRST_APP_STRING + 1, for example, then the system error handler will display the text of string OO_FIRST_STRING + OO_FIRST_APP_STRING + 1. See section **7.3.1.1. FRAME** for a discussion of how to lay out an application frame and where to put your strings in the frame.

Sometimes your application is not the current app. If you use **ER_throwVar** to throw an application error when another application is the current app, the system error handler will look for the text of the error message in the other app. This can happen when another app calls routines in your shared-code library or when a TI-BASIC program calls an extension function defined in your app. How do you throw application errors if your app is not the current app? **ER_throwFrame** to the rescue!

ER_throwFrame takes two arguments — the error number and a pointer to your application frame. The system error handler looks in the given app frame for the text of your error message. Your shared-code routines and TI-BASIC extensions should always use **ER_throwFrame** to throw application errors.

> **Note:** The second parameter to **ER_throwFrame** should be the variable containing your app's pointer to frame described in section **7.3.1.2. Pointer to FRAME**. Installing a language localizer for your app links in a new frame ahead of your app frame by updating your pointer to frame. During a subsequent call to **ER_throwFrame**, the system error handler will look first for the text of your error message in the language localizer.

10.4. Catching Errors

Sometimes, you would like your application to catch error conditions rather than allowing the system error handler to display an error message. The TRY, ONERR, and ENDTRY macros are used together to give your app control over error conditions.

```
TRY
   /* code which can throw an error */
ONERR
   /* execution continues here only if an error was thrown above */
ENDTRY
```

Begin a block of code which can throw an error with the TRY macro. If **ER_throwVar** is called anywhere in the TRY block, even in a called subroutine, execution is immediately transferred to the ONERR block.

Within the ONERR block, the error number thrown in the TRY block is available in local int variable *errCode*. Code in the ONERR block can test *errCode* to determine what kind of error occurred and take appropriate action. Variable *errCode*, because it is local to the ONERR block, cannot be referenced outside the ONERR block.

Execution in the ONERR block flows through the end of the block to the **ENDTRY** macro. Alternatively, code in the ONERR block may execute the PASS macro to throw the error on up to the next higher enclosing TRY block or call **ER_throwVar** with a different error number to raise another exception.

TRY blocks can be nested.

10.5. Cleaning Up

Many times you want to catch errors so you can clean up after the code which threw the error. If, for example, your app needs to allocate several memory handles, but any of them could fail because of low memory conditions, your app should release the handles which were successfully allocated before passing the memory error on up to the system error handler. Otherwise your app will leak memory.

```
volatile HANDLE h1 = H_NULL, h2 = H_NULL, h3 = H_NULL;
TRY
   h1 = HeapAllocThrow(BUF1_SIZE);
   h2 = HeapAllocThrow(BUF2_SIZE);
   h3 = HeapAllocThrow(BUF3_SIZE);
ONERR
   HeapFreeIndir(&h1);
   HeapFreeIndir(&h2);
   PASS;
ENDTRY
```

Handle variables h1, h2, and h3 must be initialized to H_NULL so the error handling block can distinguish between allocated and unallocated handles. **HeapAllocThrow** returns a memory handle (a value other than H_NULL) if it succeeds. If **HeapAllocThrow** fails, it throws ER_MEMORY, thereby transferring execution to the ONERR block. **HeapFreeIndir** frees the handle if it is non-null and sets the variable back to H_NULL. The ONERR block is skipped if the entire TRY block is executed without throwing an error.

Sometimes your app needs to clean up after a section of code whether an error occurs or not. TRY . . . FINALLY . . . ENDFINAL blocks are helpful in this case.

```
volatile HANDLE h1 = H_NULL, h2 = H_NULL;
TRY
   h1 = HeapAllocThrow(BUF1_SIZE);
   h2 = HeapAllocThrow(BUF2_SIZE);
   .
   . /* do something with handles h1 and h2 */
   .
FINALLY
   /* free handles h1 and h2 */
   HeapFreeIndir(&h1);
   HeapFreeIndir(&h2);
ENDFINAL
```

If any error is thrown in the TRY block, execution transfers to the FINALLY block where handles h1 and h2 are released, if necessary, and the error is passed on up. If no error is thrown, the remainder of the TRY block is executed, memory handles h1 and h2 are released in the FINALLY block, and execution continues after the ENDFINAL macro.

The FINALLY block is always executed, thus guaranteeing that handles h1 and h2 will be freed.

TRY . . . ENDFINAL blocks and TRY . . . ENDTRY blocks can be nested within each other.

10.6. Caveats

You should be aware of some coding and design issues dealing with raising and catching exceptions.

10.6.1. Jumping Out of TRY Blocks

Do not do it. Jumping out of TRY blocks causes big trouble. TRY blocks maintain a stack of saved execution contexts in order to determine where execution should resume when an error is thrown. The TRY stack is automatically popped when an error is thrown or when an ONERR or FINALLY macro is reached. It is very important that the code in a TRY block does nothing to corrupt the TRY stack.

Specifically:

- Do not execute a return statement out of a TRY block.
- Do not execute a goto statement out of a TRY block.

Executing any code which bypasses popping the TRY stack will probably cause the calculator to crash the next time an error is thrown.

There is a way to return or goto out of a TRY block if you absolutely must. The trick is to pop the TRY stack yourself before leaving the TRY block. This can be accomplished by calling **ER_success** just before the goto or return statement.

10.6.2. Referencing Auto Variables in ONERR/FINALLY Blocks

The TRY macro saves many of the CPU registers on its execution context stack. Consequently, when an error is thrown, all variables which reside in CPU registers are reset to their contents before the TRY macro was called. This is only a problem with auto variables — global and static variables are never kept in CPU registers. If code in an ONERR or FINALLY block needs the value of a variable set in the TRY block, the code must arrange to make sure the C code optimizer does not put that variable in a CPU register. This can be accomplished by declaring such variables to be volatile. So, remember this rule:

Auto variables changed in a TRY block must be declared volatile if they are referenced in an ONERR or FINALLY block.

10.6.3. Where Not to Throw Errors

One last reminder:

- Do not throw errors while processing events CM_START, CM_ACTIVATE, CM_FOCUS, CM_UNFOCUS, CM_DEACTIVATE, CM_QUIT, CM_WPAINT, CM_INSTALL, CM_UNINSTALL, CM_PACK, CM_UNPACK, CM_INIT, or CM_MODE_CHANGE.

See section **10.2. Delayed Error Messages** for details on how to signal exceptions under these circumstances.

11. Creating the User Interface

As explained in chapter **4. User Interface Overview** , the user interface consists of windows, menus, dialog boxes, fonts, and the status line. This chapter will present more detail on windows, menus and dialog boxes along with common structures shared by all three of these components. The resource compiler, which converts resource files into object code, will also be discussed. Finally, a detailed example of an application that uses windows, menus, and dialog boxes will be presented.

11.1. Common Screen Components

Windows, menus, and dialog boxes all use several common components. These components include the screen region and coordinate typedefs — SCR_RECT, SCR_COORDS; the bitmap structure — BITMAP; and the icon structure — ICON.

11.1.1. Screen/Window Regions and Coordinates

Although windows are based on window coordinates (signed short values — WIN_COORDS), the screen is limited to only unsigned char coordinates — SCR_COORDS. Because of this, there are two separate structures that define a region on the screen: SCR_RECT and WIN_RECT. A region defines a rectangular area. The x0, y0 coordinates of a region specify the upper left coordinates; the x1, y1 coordinates specify the lower right coordinates. The SCR_RECT structure is based on SCR_COORDS whereas the WIN_RECT structure is based on WIN_COORDS. While the WINDOW structure uses SCR_RECT regions internally, all calls to window routines use window region and coordinates.

SCREEN	WINDOW
typedef unsigned char **SCR_COORDS**;	typedef signed short **WIN_COORDS**;

`typedef union {`
`  struct {`
`  SCR_COORDS x0, y0;`
`  SCR_COORDS x1, y1;`
`  } xy;`
`  unsigned long l;`
`} SCR_RECT;`
	`typedef struct {`
	`  WIN_COORDS x0, y0;`
	`  WIN_COORDS x1, y1;`
	`} WIN_RECT;`

Table 11.1: Screen vs. Window Coordinates

SCR_RECTs are defined as a union since the four bytes that define the coordinates can also be represented by an unsigned long value. There is a global SCR_RECT called **ScrRect** that defines the entire writeable region of the screen. This includes the area normally reserved for an app's menu but does NOT include the status line. This area can be changed by interrupt routines in response to keystrokes, and is not available to write to under normal circumstances.

11.1.2. BITMAP

BITMAPs are used to store or retrieve rectangular regions on a window. They can also be used for cursors (text or graphic), to do animation, to highlight areas on the screen, and as images in menus (along with ICONs). A BITMAP is defined as follows:

```
typedef struct {
   WORD NumRows;
   WORD NumCols;
   BYTE Data[1];
}  BITMAP;
```

A BITMAP must always have one or more rows and one or more columns so its size is always at least 5 bytes long. The macro BITMAP_HDR_SIZE defines the size of the BITMAP header (4 bytes). The **CalcBitmapSize** routine calculates the size given a pointer to a BITMAP structure.

11.1.3. ICON

An ICON can be thought of as a fixed 16x16 bitmap. Since ICONs do not have the BITMAP header, they cannot be used interchangeably with BITMAPs. They are stored as an array of 16 unsigned shorts (WORD). ICONs are normally only used in MENUs.

11.2. Windows

The Window routines provide a method to write to the screen of the calculator. Each active window must have an associated WINDOW structure. The Window routines use the WIN_RECT structure (defined in section **11.1.1. Screen/Window Regions and Coordinates**) to define regions. Coordinates are all specified by the WIN_COORDS type.

All screen IO must go through an opened window. Windows are opened with the **WinOpen** function. The windows in the system are linked together so that when a window is closed, the system can walk the list of windows to determine which windows are dirty and therefore need to be repainted. Hence, it is important for an app to close its window when done to remove it from the linked-list of

windows. When an application is started (gets the CM_START message), it gets a WIN_RECT that defines the region of its window. This region is based on whether the calculator is in full or split screen mode and the side of the screen split of the current app. An app can create additional windows which may overlap.

11.2.1. Window Regions and Coordinates

A window region is limited to a rectangular area defined by two coordinate pairs: an x, y pair that defines the upper left corner of the region and an x, y pair that defines the lower right corner of the region. All window region coordinates are represented as signed 16-bit numbers. The coordinate (0, 0) is the upper left corner of a region. Since coordinates may be negative, the coordinate (-1, -1) is up one pixel and over one pixel to the left of the coordinate (0, 0).

A window has three regions associated with it. The first region, the actual window, is the region that was defined when the window was created. The second region is the area of that window that may be drawn to, the client region. If the window is full screen (not counting the application's menu or the status bar which may not be overlapped), then the client region is equal to the window region. The client region is reduced by adding borders or a title to a window. Each window also has a clipping region which is a subset of the client region. Initially, the clipping region is equal to the client region but it may be changed by the app with the **SetWinClip** routine. The following graphic illustrates the three window regions.

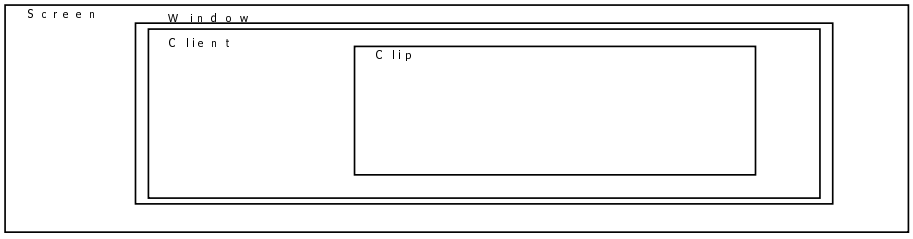


Figure 11.1: Window Regions

Real windows are limited to the size of the screen. Virtual windows are allocated a bitmap in memory and are limited to unsigned coordinates of the region (0, 0) . . . (255, 255). The size of the virtual window must not exceed the maximum allowable block of heap memory.

11.2.2. Window Routines

The following is a complete list of all of the Window routines. It is split into two sections: the main window routines and utility routines.

WinActivate — Make a window the current active window.

WinAttr — Set the attribute for the next write to a window.

WinBackground — Change the current default attribute for the background of a window.

WinBackupToScr — Copy a window's duplicate screen to the real screen.

WinBeginPaint — Save the current screen state and set up the screen to draw for the current window.

WinBitmapGet — Retrieve a BITMAP from a window.

WinBitmapPut — Store a BITMAP to a window.

WinBitmapSize — Calculate the size of a BITMAP in a window.

WinBitmapSizeExt — Calculate a BITMAP size including negative coordinates.

WinChar — Write a character at the current pen position.

WinCharXY — Write a character to a specific location.

WinClose — Close a window.

WinClr — Clear a window.

WinDeactivate — Deactivate a window, making the next window the active window.

WinDupStat — Enable/disable duplicate writes to a window (opened with WF_DUP_SCR).

WinEllipse — Draw an ellipse to a window.

WinEndPaint — Restore the current screen state that was saved with the corresponding **WinBeginPaint**.

WinFill — Fill a region of a window.

WinFillTriangle — Draw a filled triangle.

WinFont — Change the font for a window.

WinHeight — Return the height of a window's client region.

WinHide — Mark a window as not visible so that it is never activated by the system.

WinHome — Move the pen location for a window to the home position.

WinLine — Draw a line to a window.

WinLineExt — Draw a line using the slower but more accurate clipping code.

WinLineRel — Draw a line relative to the current pen location.

WinLineTo — Draw a line from the current pen location.
WinMoveRel — Move the current pen location relative to its current position.
WinMoveTo — Move the current pen location.
WinOpen — Open a window.
WinPixGet — Return the status of an individual pixel (on or off).
WinPixSet — Set a pixel.
WinRect — Draw a rectangle.
WinRemove — Close a window with the option to not update the screen.
WinReOpen — Reopen a window keeping the duplicate image if the window's size does not change.
WinScrollH — Scroll a region horizontally.
WinScrollV — Scroll a region vertically.
WinStr — Write a string to the current pen location.
WinStrXY — Write a string to a specific location.
WinWidth — Return the width of a window's client region.

Additionally, there are several utility routines for working with windows.

CalcBitmapSize — Calculate the size of a bitmap given a pointer to a BITMAP structure.
ClientToScr — Merge two SCR_RECTs.
DrawWinBorder — Redraw the border for a window.
MakeWinRect — Create a WIN_RECT.
MakeScrRect — Create a SCR_RECT.
ScrToWin — Convert a SCR_RECT structure to a WIN_RECT structure.
SetWinClip — Set the clipping region for a window.

11.3. Menus

There are two menu types: toolbars and pop-ups. Pop-ups are, for the most part, toolbars without a menu bar, however, they may have a title (for an example see Figure 4.2). Menus are limited to three levels (not counting the menu bar).

Menus are divided into two categories: static and dynamic. Examples of both are included at the end of this chapter. Like all resources, the text strings in a menu can be localized by using string reference numbers instead of actual text. Internally, menus are kept as a MENU structure that contain one or more MENU_ITEM structures at the end. Static menus are created with the resource

compiler and the data structures defining them reside in Flash. Static menus can be loaded into RAM with the **MenuLoad** function so that they can be modified just like dynamic menus. After dynamic menus are created, they must be locked and remain locked while they are in use so they do not move (since **MenuBegin** is passed a direct pointer to a MENU structure).

11.3.1. Menu-Draw Structure

Menus require a separate RAM based structure to be built which is created by **MenuBegin,** the menu-draw structure. This RAM based structure contains, among other things, flags for checkmarks and enabled items as well as other structures for maintaining the menu. Pop-ups only need a menu-draw structure if they need menu-like features: checkmarks or enable/disable features. In that case, use the **PopupBegin**/**PopupBeginDo** functions. Dynamic menus are created at run-time and reside entirely in RAM.

11.3.2. Menu IDs

Each menu item is assigned an identifier. By default these identifiers range from 1 up to the number of menu items in the menu. Menus created with the resource compiler can have symbolic names assigned to each menu item. These names are stored in a header file with the same base-name as the resource file only with a .h extension added. The default identifier numbers can be overridden if needed in the resource file. Dynamic menu items are also numbered sequentially starting at 1 by default, but the number may also be overridden. The example at the end of this chapter (see **11.6. Example**) uses overridden menu IDs. See the example in the **MenuTopStat** function entry point for an example that uses symbolic names.

Each item in a menu or pop-up contains either a text string, an ICON or a BITMAP. Use **DynMenuAdd** or **DynMenuChange** to add or change menu items in a dynamic menu. There are some older routines, **MenuAddIcon**, **MenuAddText**, and **PopupAddText**, for specifically adding icons or text but these routines do not need to be used. See section **11.5. Resource Compiler** for more information about using BITMAPs in a menu.

11.3.3. Menu Routines

DynMenuAdd — Add a new entry (text, icon, or bitmap) to a dynamic menu or pop-up.

DynMenuChange — Change an entry in a dynamic menu or pop-up.

FKeyI_H — For the given function key, return its index relative to KB_F1.

MenuAddIcon — Add an icon entry to a dynamic menu.

MenuAddText — Add a text entry to a dynamic menu.

MenuBegin — Begin the use of a menu by allocating a menu-draw structure and drawing the menu's top-level.

MenuCheck — Set, clear, flip, or return the status of a check mark for a menu item.

MenuEnd — End the use of a menu, freeing the menu-draw structure.

MenuFlags — Return the flag word for a dynamic menu/pop-up structure.

MenuGetTopRedef — Return the current value of a redefinable top-level menu item.

MenuItemDef — Given a menu ID, return a pointer to the text, ICON, or BITMAP defining it.

MenuKey — Handle a key for a menu returning the menu item selected.

MenuLoad — Begin a dynamically created menu, using a static menu as the starting point.

MenuNew — Begin a dynamically created menu.

MenuOff — Gray-out the top-level of a menu.

MenuOn — Draw the top-level of a menu.

MenuPopup — Execute a static pop-up as defined by the resource compiler, returning the item selected.

MenuSubStat — Enable or disable a sublevel menu item.

MenuTopRedef — Redefine a top-level menu item ICON for a menu that was started with the MBF_REDEF flag.

MenuTopSelect — Select a top-level menu item by drawing a thick box around the menu item.

MenuTopStat — Enable or disable a top-level menu item.

PopupAddText — Add a text entry to a dynamic pop-up.

PopupBegin — Allocate a menu-draw structure for a dynamic pop-up so that the pop-up items can have the enable / disable or checkmark features of menus.

PopupBeginDo — Execute a dynamically allocated pop-up using the handle returned by **PopupBegin**.

PopupClear — Clear all entries of a dynamically created pop-up.

PopupDo — Execute a dynamic pop-up created by **PopupNew**.

PopupNew — Begin a dynamically created pop-up, use **DynMenuAdd** or **DynMenuChange** to add to or change the pop-up.

PopupText — Return a pointer to the text of a dynamically created pop-up.

QMenuTopSelect — Return the currently selected top-level menu item as set by **MenuTopSelect**.

11.4. Dialog Boxes

Dialogs, like menus, can be either static or dynamic. The text strings in a dialog can be localized by using string reference numbers instead of actual text. Internally, dialogs are kept as a DIALOG structure that contain one or more DIALOG_ITEMS structures at the end. Static dialogs are created with the resource compiler and the data structures defining them reside in Flash. The only routine that handles static dialogs is the **Dialog** function. Dynamic dialogs are created with **DialogNew** and new fields can be added with **DialogAdd**. Dynamic dialogs are executed with the **DialogDo** function. Dialogs use a call-back routine to communicate with the caller as the user interacts with the dialog box. Call-backs are explained later in section **11.4.4. Dialog Call-Backs**.

11.4.1. Dialog Routines

Dialog — Open a dialog box and handle all keys pressed by the user until the dialog box is closed, returning any modified dialog box items.

DialogAdd — Add an item to a dynamic dialog box.

DialogDo — Works like **Dialog** only for dynamically created dialog boxes.

DialogNew — There are several utility routines for working with and creating standard dialog boxes.

DlgMessage — Execute a system created dialog with a title and a word-wrapped message.

DlgNotice — Macro: **DlgMessage** (Title, Msg, PDB_OK, 0).

DrawStaticButton — Utility routine to draw dialog box style buttons at the bottom of a window.

VarNew — Create a standard NEW dialog box.

VarOpen — Create a standard OPEN dialog box.

VarSaveAs — Create a standard SAVE COPY OF dialog box.

11.4.2. Dialog Fields

A dialog box consists of one or more fields. For a static dialog box these fields are defined in a structure which is compiled by the resource compiler into a DIALOG structure. There are some differences between the fields added with **DialogAdd** and those created with the resource compiler.

- Precede each of the field types listed below with “D_” to use as the dialog type to pass to **DialogAdd**. For an example, see the **DialogNew** function.
- HPOPUP can only be passed to **DialogAdd**, since resources cannot specify handles.
- DYNHEADER is only available on the resource compiler, since the header text can be specified at the time the header is created.

For a dynamic dialog box, each field is added to the dialog box by calling the **DialogAdd** function. The fields are defined in the following paragraphs. Every field has an associated x and y coordinate which is relative to the dialog box (not the screen) and a flag byte. The flag byte values are explained after all of the dialog fields.

11.4.2.1. Field Index

Each dialog field is assigned a field index starting with zero. For static dialogs this is based on the order they are defined in the resource file. For dynamic dialogs, it is the order they are added with the **DialogAdd** function. This field index is passed through the dialog’s call-back function as explained in this section and the call-back section.

11.4.2.2. DYNPOPUP

char * *TextPtr*, HANDLE (* *GetPopup*) (WORD),
WORD *OptionListIndex*

The *TextPtr* and *OptionListIndex* parameters are the same as for a normal POPUP field. Instead of using the name/address of a statically created pop-up, the address of a function that returns the address of a dynamically created pop-up is used. The pop-up may still be statically created but this allows for the possibility of passing one of several different pop-ups. The *GetPopup* routine is called with a single value which is the DYNPOPUP’s field index.

11.4.2.3. EDIT_FIELD

char * *TextPtr*, WORD *bOffset*, WORD *Flen*, WORD *Dlen*

An EDIT_FIELD is drawn as a box with an optional title. An empty string, `""`, is used to indicate no title. The field is defined by a string which labels the box, an offset into the *FieldBuf* array passed to **Dialog** (*bOffset*), the total length of the field in the Buffer array (*Flen*), and a display length in characters, not pixels (*Dlen*). The data in the Buffer array is copied to the display when **Dialog** is first

called. The data is assumed to be a zero-terminated string of characters and is returned as such. See **DialogNew** for an example. The call-back is called each time the edit field is modified. See section **11.4.4. Dialog Call-Backs** for a description of the events involved.

11.4.2.4. HEADER

char * *TextPtr*, WORD *lButton*, WORD *rButton*

A HEADER field is a static field which defines the title of a dialog box. The field is defined by a zero-terminated string. Two optional fields may follow: a left predefined button and a right predefined button. If you do not want predefined buttons when using **DialogAdd** then you must still pass two zero words or pass zero for the second button if only one button is needed. The buttons are placed in the lower left and right portion of the dialog box. The predefined buttons are: PDB_OK, PDB_SAVE, PDB_YES, PDB_CANCEL, PDB_NO, and PDB_GOTO.

11.4.2.5. HEDIT

char * *TextPtr*, WORD *Dlen*

HEDIT fields do not use the *FieldBuf* array passed to the **Dialog** function so they do not need the *bOffset* and *FLen* fields like normal edit fields. The call-back routine is called with the first parameter equal to DB_GET_EDIT_HANDLE and the second parameter equal to the field's index value. The call-back routine should then return the handle of an edit buffer of at least *DLen* bytes long.

11.4.2.6. HPOPUP

char * *TextPtr*, HANDLE *hPopup*, WORD *oIndex*

HPOPUPs work like POPUPs in a dialog box but instead of being passed a pointer to a MENU structure (which defines the POPUP), the handle to a dynamically created POPUP is passed to **DialogAdd**. This handle does not have to be locked since the dialog code will lock and unlock the handle as needed. As stated earlier, HPOPUPs cannot be used in the resource compiler.

11.4.2.7. MENU

MENU * *menuPtr*, WORD *MaxMenuWidth*

A MENU field defines a menu for a dialog box. Each dialog box can have at most one menu. A menu field is defined by a pointer to a MENU structure created statically with the resource compiler or dynamically (in which case the caller must insure the structure remains locked while in-use in the dialog box). The menu is drawn at the x, y coordinates specified using **MenuBegin**. When a menu key is pressed, the call-back is passed to the MENU's field index along with a DWORD value containing the menu-handle returned from **MenuBegin** in the high word and the key code in the low word. The call-back may return DB_EXIT to close the dialog box or a value greater than or equal to zero. This value is the field index of the item that will now be the top-most item in the dialog box. This scheme allows for multipage dialog boxes like the MODE screen on the calculator, see example

below. The numbers 0, 8, 16 represent the field indexes of the mode screen's dialog box.

```
BOOL ModeCallBack(WORD dlgId, DWORD dValue)
.
.
.
// Menu key press
if (dlgId == 20) {
   WORD fkey = LO_WORD(dValue);
   return fkey == KB_F1 ? 0 : fkey == KB_F2 ? 8 : fkey == KB_F3 ? 16 : -1;
}
```

11.4.2.8. POPUP

char * *TextPtr*, MENU * *Popup*, WORD *OptionIndex*

A POPUP field defines a pop-up menu. The field is defined by a pointer to an optional zero-terminated string used to label the pop-up, a pointer to a pop-up MENU structure (as created by the resource compiler, for dynamic pop-ups use HPOPUP), and the index (*OptionIndex*) into the *OptionList* passed to **Dialog**.

> **Note:** OptionList is a C structure, so indexes to it (like OptionIndex) are zero based. See the **Dialog** function entry point for an example.

The value stored in *OptionList* for this field is the value stored in the pop-up menu structure as the identifier for the currently selected menu item. Default identifiers for menus start from 1 and go up to the number of menu items in the menu. The user may redefine these identifiers if needed. The call-back is called each time the pop-up is modified. See section **11.4.4. Dialog Call-Backs** for a description of the events involved.

11.4.2.9. SCROLL_REGION

WORD *x1*, WORD *y1*, WORD *Index0*, WORD *Index1*, WORD *NumDspFields*, WORD *TotNumFields*, WORD *FieldHeight*

A scroll region defines a group of similar fields that will scroll as the user moves through the fields. The region (using dialog box coordinates) of the dialog box that scrolls is defined by the x and y parameters defined for all dialog fields to specify the upper left coordinate of the scroll region along with an *x1* and *y1* field to specify the lower right coordinate of the scroll region. The field index of the first field that will scroll (*Index0*) followed by the index of the last field that is scrollable (*Index1*) are specified next. These values are followed by the number of fields that are displayed at one time (*NumDspFields*), the total number of scrollable fields (*TotNumFields*), and finally the height in pixels of each field (*FieldHeight*). All of the fields that are scrollable must be defined contiguously and have the DF_SCROLLABLE bit set in their flag byte. The coordinates of the scrollable fields are relative to the dialog box except that they may extend beyond the

bottom coordinate of the dialog box. They are defined assuming a virtual scroll region.

Note:	If you use SCROLL_REGION, it must be the first item defined in the dialog box.

11.4.2.10. TEXT

char * TextPtr

A TEXT field is a static field that allows for stand-alone text to be placed anywhere in a dialog box. The field is defined by a zero-terminated string.

If the DF_OWNER_DRAW flag is set, then the call-back is passed the field index and a pointer to a OWNER_DRAW_STRUCT structure. The first item of this structure is a direct pointer to the DIALOG_ITEMS structure for the field to be drawn (this is not normally used). The second item is a pointer to the WINDOW structure for the dialog box. Using this pointer, the call-back can draw anything and anywhere to the dialog box (all clipped to the dialog boxes window). This may include both text and images.

```
typedef struct {
   DIALOG_ITEMS *Item;
   WINDOW *pW;
}  OWNER_DRAW_STRUCT;
```

An example from the VAR-LINK code's receive variable overwrite dialog box is listed below.

```
/* VAR-LINK RECEIVE: Overwrite callback  */
WORD VL_OverwriteCB( WORD DlgId, DWORD dValue )
{
   if (DlgId == 0) {
      WinStrXY( ((OWNER_DRAW_STRUCT *) dValue)->pW, 8, 15, (char *)
                  VL_VarName );
      return TRUE;
   }
   if (DlgId == DB_QACTIVE && dValue == 2)
      return (VL_OverwriteAns[0] == VLO_NO);
   if (DlgId == 1)
      return DB_REDRAW_AND_CONTINUE;
   return TRUE;
}
```

11.4.2.11. XFLAGS

WORD *xFlags1*, *xFlags2*, *xFlags3*, *xFlags4*

The XFLAGS field defines an array of four extended WORD flags. Currently only the first WORD is used and may contain the following flags. The remaining three WORDs should always be set to zero for future compatibility.

XF_ALLOW_VARLINK

Setting this extended flag allows all edit fields in the dialog box to allow the [VAR-LINK] key to be activated within the dialog box and to paste results to the edit field. If this flag or XF_VARLINK_SELECT_ONLY is not set, then when [VAR-LINK] is pressed in a dialog box, the dialog box will be closed and VAR-LINK will be activated.

XF_NO_ALPHA_LOCK

On the TI-89, Alpha-Lock is turned on for all dialog boxes with edit fields. Setting this extended flag disables this feature.

XF_VARLINK_SELECT_ONLY

This flag is similar to XF_ALLOW_VARLINK except that the user may not make any variable changes inside VAR-LINK (like deleting, copying, renaming, or locking variables).

11.4.3. Dialog Flags

Each dialog field has a flag byte that provides additional features explained in the table below.

Flag	Useable with these fields
DF_TAB_ELLIPSES	EDIT, POPUP — Draw '. . .' between label and pop-up/edit box.
DF_MAX_MENU_WIDTH	MENU — Pass MBF_MAX_MENU_WIDTH to **MenuBegin** instead of zero.
DF_SCROLLABLE	Any field — Used to denote scrollable fields in SCROLL_REGION.
DF_CLR_ON_REDRAW	SCROLL_REGION — Clear the entire visible scroll region when redrawn.
DF_TAB_SPACES	EDIT, POPUP — Draw spaces between label and pop-up/edit box.
DF_OWNER_DRAW	TEXT — Call-back is responsible for drawing this field (which may be text or an image).
DF_POPUP_RADIO	POPUP — Pop-ups act like the TI-83 radio buttons.
DF_SCREEN_SAVE	Any field (if first field) — The dialog code saves the area underneath the dialog box when it is started, DB_MEMFUL returned if it cannot.
DF_SKIP	Any field — This field is skipped since the system maintains this flag.

Table 11.2: Dialog Flags and Corresponding Fields

11.4.4. Dialog Call-Backs

The dialog code and the application can communicate changes to the status of a dialog box as the dialog box is changed by the user. This is done through the call-back function which is defined when the dialog box is created (either statically or dynamically). Even if no call-back is needed, a routine MUST be provided (the same do-nothing call-back can be provided for multiple dialogs). An example of a do-nothing call-back is shown below.

```
DWORD NoCallBack( WORD DlgId, DWORD dValue ) {
   return TRUE;
}
```

If the user provides a call-back function then it is called under the following cases. The call-back is passed with two parameters: *DlgId* (WORD) and *dValue* (DWORD). If *DlgId* is equal to DB_QACTIVE then the dialog code needs to know if the field whose field index is in *dValue* is active. Indexes range from 0 for the first item in the dialog box up to the number of fields in the dialog box less one. The call-back should return TRUE if it is active (not grayed-out) or FALSE if it is inactive (grayed-out). Inactive static fields (HEADER and TEXT) are not drawn instead of being grayed-out.

If *DlgId* is in the range zero through the number of fields in the dialog box less one, then it is the field index of a dialog item that has just been changed by the user. The application can take any necessary action (including adjusting values changed by the user). It must return one of the following values.

DB_REDRAW	Redraw the dialog box and ignore the key just pressed by the user.
DB_REDRAW_AND_CONTINUE	Redraw the dialog box and accept the key just pressed by the user.
TRUE	Do not redraw the dialog box and accept the key just pressed by the user.

Table 11.3: Call Back Function Return Values

dValue will vary depending on the type of the field changed:

POPUP — The value from the pop-up selected by the user (the identifier for the pop-up).

MENU — The low WORD contains the key pressed by the user to activate the menu. The high WORD contains the handle of the menu (the dialog box code calls **MenuBegin** initially on the menu field). The call-back may activate the menu if needed. See section **11.4.2.7. MENU** for more details.

EDIT — Address pointing to the data the user just entered.

DlgId can also have other special values. If it is DB_GET_TITLE, then *dValue* will be zero and the call-back must return the text for the header of the dialog box. This is only used if the static dialog box used the DYNHEADER field. If *DlgId* is DB_GET_EDIT_HANDLE then *dValue* will be the field index of an HEDIT field. See section **11.4.2.5. HEDIT** for more details.

11.5. Resource Compiler

The resource compiler (rc2.exe) is used to create dialog boxes, menus, and pop-ups. Normally, the TI **FLASH** Studio™ (IDE) will call the Resource Compiler. The resource compiler translates resource descriptions into internal structures usable by the AMS. To call the Resource Compiler manually, the following syntax is used.

rc2 [-mp] resource-file

The optional switch **m** forces all common strings within menus to be merged together, if possible, in order to save space. Due to the compacted nature of a menu, some strings cannot be merged. The optional switch **p** calls the C preprocessor (using com68.exe) on the resource file first. This allows a resource file to contain any C preprocessor commands (such as #define, #include, #ifdef, . . .).

The resource compiler translates the resource file into an assembly language file that defines the individual MENU or DIALOG box structures (POPUPs are special cases of MENUs). The resource file is a standard text file (blanks, tabs, and new-lines are skipped). A semicolon in the first column of a line denotes a comment. Each structure is denoted by a keyword: DIALOG, TOOLBOX or POPUP. The keyword is followed by additional fields as defined below.

Text fields in dialog boxes and menus are specified as either strings of text delimited by double quotation marks or resource string numbers. If resource string numbers are used, the app is responsible for supplying the text definitions in its application frame.

Menus and pop-ups can also have ICONs or BITMAPs in place of text strings. Icons and bitmaps can be defined in-line or in an icon/bitmap file. Icons are defined in-line with a single left bracket followed by 16 unsigned short values (0 . . . 0xFFFF) and a terminating right bracket. The hex values use the C syntax of numbers so 0xABCD is the same as the decimal number 43981. Bitmaps are defined in-line with double left brackets followed by the number of rows in the bitmap, the number of columns, the data as a sequence of hex bytes (0 . . . 0xFF), and finally double right brackets.

Icons and bitmaps can also be defined in an icon/bitmap file and referenced indirectly in the resource file. The icon/bitmap file is a standard text file with the icons and bitmaps defined as specified in the preceding paragraph. Each icon or

bitmap is followed by a comma and a unique identifying name. The icon or bitmap is referenced in the resource file by using a * character followed by the icon/bitmap filename, a comma, and then the identifying name specified in the icon/bitmap file. The following example creates a menu using both an icon and several bitmaps from the file appr2.ico which follows. This example also creates a pop-up using an embedded icon and an embedded bitmap as shown in Figure 11.2.

```
TOOLBOX TestMenu, 0, 0, 0 {
   "Text" {
      *appr2.ico, ICON_1, ID_2
      *appr2.ico, BITMAP_1, ID_3
   }
   *appr2.ico, BITMAP_2, ID_4
   *appr2.ico, BITMAP_3, ID_5
}

POPUP TestPopup, 0, 0 {
   [0x0000, 0xFFFE, 0x2FF9, 0x0BF9, 0x02F9, 0x00B9, 0x1029, 0x3006,
      0x6000, 0x6000, 0xFFFF, 0xFFFF, 0x6000, 0x6000, 0xF000,
      0x0000], ID_P1
   [[12, 14, 0x7F, 0xFC, 0x40, 0x04, 0x4F, 0xC4, 0x40, 0x44, 0x40,
      0x44, 0x4F, 0xC4, 0x48, 0x04, 0x48, 0x04, 0x48, 0x04, 0x4F, 0xC4,
      0x40, 0x04, 0x7F, 0xFC]], ID_P2
}

// APPR2.ICO
[0x0000, 0xFFFE, 0x2FF9, 0x0BF9, 0x02F9, 0x00B9, 0x1029, 0x3006, 0x6000,
   0x6000, 0xFFFF, 0xFFFF, 0x6000, 0x6000, 0xF000, 0x0000], ICON_1
[[12, 14, 0x7F, 0xFC, 0x40, 0x04, 0x4F, 0xC4, 0x40, 0x44, 0x40, 0x44,
   0x4F, 0xC4, 0x48, 0x04, 0x48, 0x04, 0x48, 0x04, 0x4F, 0xC4, 0x40,
   0x04, 0x7F, 0xFC]], BITMAP_1
[[15, 11, 0x00, 0x00, 0x08, 0x00, 0x0C, 0x00, 0x0E, 0x00, 0x0F, 0x00,
   0x0F, 0x80, 0x0F, 0xC0, 0x0F, 0xE0, 0x0F, 0x00, 0x0F, 0x80, 0x0D,
   0x80, 0x08, 0xC0, 0x00, 0xC0, 0x00, 0x60, 0x00, 0x60]], BITMAP_2
[[12, 12, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x10,
   0x00, 0x30, 0x00, 0x60, 0x00, 0xC0, 0x11, 0x80, 0x1B, 0x00, 0x0E,
   0x00, 0x04, 0x00]], BITMAP_3
```

Note that BITMAPs in menus are limited to a maximum of 16 rows and the columns are limited to the width of a menu item. The main reason for using BITMAPs in a menu is in the top-level toolbar. By using BITMAPs, the menu system can reduce the total width of the top-level toolbar as shown in the following image from the preceding example. Since the menu items associated with F2 and F3 are bitmaps, the width of the corresponding tabs is not as wide as if they had been defined as ICONs which are always assumed to be 16 by 16 in a menu.

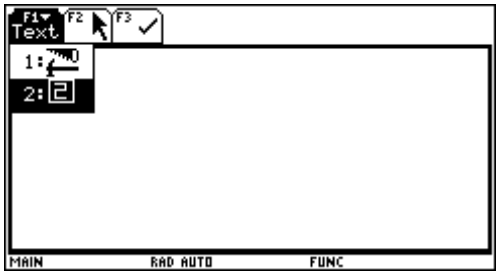


Figure 11.2: Screen Shot from Test Menu Example

11.5.1. DIALOG Boxes

DIALOG *DialogName, DialogWidth, DialogHeight, CallBackRoutine* {

DYNPOPUP {*Flags*, *x*, *y*}, *Text*, *GetDynPopupRoutine*, *OptionListIndex*

DYNHEADER {*Flags*, *x*, *y*}, *PDB1*, *PDB2*

EDIT {*Flags*, *x*, *y*}, *Text*, *BufferOffset*, *FieldLength*, *DisplayLength*

HEADER {*Flags*, *x*, *y*}, *Text*, *PDB1*, *PDB2*

HEDIT {*Flags*, *x*, *y*},*Text*, *0*, *0*, *DisplayLength*

MENU {*Flags*, *x*, *y*}, *MenuName*

POPUP {*Flags*, *x*, *y*}, *Text*, *PopupName*, *OptionListIndex*

SCROLL_REGION {*Flags*, *x*, *y*}, *x1*, *y1*, *Index0*, *Index1*, *NumDspFields*, *TotNumFields*, *FieldHeight*

TEXT {*Flags*, *x*, *y*}, *Text*

XFLAGS {*Flags*, *x*, *y*}, *FlagVal1*, *FlagVal2*, *FlagVal3*, *FlagVal4*

}

DialogName is the name given to the dialog box structure. The generated .h file is included to reference the dialog box from a C program. *DialogWidth* and *DialogHeight* define the width and height of the dialog box in pixels. If either or both is zero, then the system will set them to the maximum values used by the dialog box at the time the dialog box is executed. *CallBackRoutine* is the name of the call-back function used for the dialog box. See section **11.4.4. Dialog Call-Backs** for a description of call-backs.

Each item within the dialog structure has a *Flags* byte and an (x, y) coordinate of the upper left pixel of the field. All coordinates in a dialog box are relative to the upper left corner of the dialog box.

See section **11.4.2. Dialog Fields** for a description of the individual fields and their parameters. There are a few differences in defining a dialog box with the resource compiler as noted in the following paragraphs.

DYNHEADER does not have a field for the text of the header. Instead, the call-back is called with the *DlgId* parameter set to DB_GET_TITLE. The call-back routine must return a pointer to the text to use for the title. DYNHEADERs are only available in the resource compiler and cannot be added to dynamic menus with the **DialogAdd** function.

HEDIT entries do not use the *BufferOffset* or *FieldLength* fields and so these should be set to zero (*DisplayLength* must still be provided).

MENU (TOOLBOX can be used also) and POPUP require the name of a menu or pop-up that is defined elsewhere in the resource file.

11.5.2. MENUs

```
MENU MenuName, Flags, MaxHeight, MaxWidth {
   Text1, TOP_LEVEL_ID1
   Text2 {
      Text3, SUB_LEVEL_1_ID3
      Text4, SUB_LEVEL_1_ID4
      Text5 {
         Text6, SUB_LEVEL_2_ID6
         Text7 {
            Text8, SUB_LEVEL_3_ID8
            Text9, SUB_LEVEL_3_ID9
         }
      }
      .
      .
      .
   }
   Text10, TOP_LEVEL_ID10 {
   }
   *IconBitmapFile, IconOrBitmapName, TOP_LEVEL_ID11
   [HexValue1, . . ., HexValue16], TOP_LEVEL_ID12
   [[ NumRows, NumCols, BitmapData, ...]], TOP_LEVEL_ID13
   .
   .
   .
   }
```

MENU and TOOLBOX are synonymous. MENUs may have up to three levels of nesting (the top-level being the first level). Each successive level is enclosed in braces. Like dialog boxes, IDs are assigned sequential values (starting from one) by default. The names are stored in a header file (same base-name as the resource file with an extension of .h). The default numbers may be changed by following a name with an equals sign and then a number specifying the modified ID value.

As with dialog boxes, *MenuName* is the name used to refer to the menu both in the resource file and inside a C program. *Flags* is usually zero for a menu. If *Flags* is equal to RC_NO_IDS then the menu IDs are not symbolic values stored in the menu's header file but instead are absolute numbers that will be used as IDs for each menu item. If *MaxWidth* (in pixels) is set to zero then **MenuBegin** will draw the menu only as wide as necessary. *MaxHeight* should be set to 0 which will use the default height.

A menu item that includes other menu items is called a parent (the entry `Text2 {` above). Parent menu items do not normally have IDs associated with them since they cannot be selected by the user and so do not return an ID value. However, they may need an ID for two reasons. The first is if the menu is to be loaded using **MenuLoad** and then later modified; since an ID is needed to modify an entry. The second reason is if the menu item is to be disabled or checked. The entry `Text1, TOP_LEVEL_ID1` is not a parent since it is a stand-alone entry. It can be selected by pressing the menu key associated with that entry (F1 in this case). The entry `Text10, TOP_LEVEL_ID10 {` is a special case. It is a parent (cannot be selected) that has no children. This is only used for menus loaded with the **MenuLoad** function. This entry can then later have other menu entries (children) added to it. See the **MenuLoad** function for an example.

11.5.3. POPUPs

```
POPUP PopupName, Flags, MaxHeight, Title {
   . . . same format as a menu . . .
}
```

POPUPs are defined almost identically to MENUs. The differences are that POPUPs may have an optional title (a text string delimited by double quotes or an icon) as well as a *Flags* value of MF_NO_NUMS (32) which signifies that the individual items in the pop-up are not to be numbered. If *MaxHeight* is set to zero then the **MenuPopup** routine will try to fit as much of the pop-up on the screen as it can. This value may be overridden as long as the size of any title is taken into account.

11.6. Example

This section will discuss the following example in detail. It has all of the components described in this chapter: windows, menus (toolbars and pop-ups), and dialog boxes. It uses the resource compiler for a static menu and dialog box, and creates a dynamic pop-up.

```
1   // APP1.C
2   #include "tiams.h"
3   #include "app1.h"
4   #include "appr1.h"
5
6   static void AP_app(pFrame self, PEvent e);
7   FRAME(appObj, OO_SYSTEM_FRAME, 0, OO_APP_FLAGS, 4)
8      ATTR(OO_APP_FLAGS, APP_INTERACTIVE)
9      ATTR(OO_APP_NAME, "app1")
10     ATTR(OO_APP_PROCESS_EVENT, &AP_app)
11     ATTR(OO_APP_DEFAULT_MENU, &AppMenu )
12  ENDFRAME
13
14  pFrame pAppObj = (pFrame)&appObj;  /* Must be 1st! */
15  WINDOW appW;
16  char buf[22];
17
18  static void AP_app(pFrame self, PEvent e) {
19     Access_AMS_Global_Variables;
20     WIN_RECT appWR;
21     HANDLE hPopup;
22     short key, vSelect; WORD opts[3];
23     char outStr[256];
24
25     switch (e->command) {
26        case CM_START:
27           appWR = *(e->info.startInfo.startRect);
28           if (WinOpen( &appW, &appWR, WF_TTY  | WF_DUP_SCR))
29              WinClr( &appW );
30           else
31              EV_quit();
32           strcpy( buf, "FIRST" );
33           strcpy( buf+11, "SECOND" );
34           break;
35        case CM_ACTIVATE:
36           EV_defaultHandler(e);
37           EV_disableCmd(ACM_NOTHING);
38           WinBeginPaint( &appW );
39           WinActivate( &appW );
40           WinStr( &appW, "Just activated\n" );
41           break;
42        case CM_DEACTIVATE:
43           WinEndPaint( &appW );
44           break;
45        case CM_QUIT:
46           if (appW.Next) {
47              WinClose( &appW );
48              appW.Next = NULL;
```

```
            }
            break;
         case CM_KEY_PRESS:
            key = e->info.keyInfo.keyCode;
               if ((key <= 0xFF && isprint(key)) || KB_ENTER == key) {
               WinChar( &appW, key );
            }  else
               EV_defaultHandler(e);
            break;
         case CM_WPAINT:
            DrawWinBorder( &appW, &appW.Window );
            WinBackupToScr( &appW );
            break;
         case ACM_WINSTR:
            WinStr( &appW, "APP1 CMD1\n");
            break;
         case ACM_DIALOG:
            opts[0] = opts[1] = 1; /* default to 1st pop-up item */
            if (KB_ENTER == Dialog( &tDialog, -1, -1, buf, opts )) {
            sprintf( outStr, "Edit1: %s\nEdit 2: %s\nPopup1: %d\nPopup2: %d",
            buf, buf+11, opts[0], opts[1] );
            DlgNotice( "tDialog", outStr );
         }
         break;
      case ACM_POPUP:
         if (hPopup = PopupNew( "DYNAMIC POPUP", 0 )) {
            DynMenuAdd( hPopup, 0, buf, 1, DMF_TEXT | DMF_TOP );
            DynMenuAdd( hPopup, 0, buf+11, 2, DMF_TEXT | DMF_TOP );
            DynMenuAdd( hPopup, 0, "LAST ITEM", 3, DMF_TEXT | DMF_TOP );
            if (!(MenuFlags(hPopup) & MF_ERROR)) {
               vSelect = PopupDo( hPopup, -1, -1, 0 );
               sprintf(outStr, "Selected %s", PopupText(hPopup, vSelect));
               DlgNotice( "dPopup", outStr );
            }
            HeapFree( hPopup );
         }
         break;
      case ACM_HFONT:
         WinFont( &appW, F_8x10 );
         break;
      default:
         EV_defaultHandler(e);
         break;
      }
   }

   DWORD NoCallBack( WORD DlgId, DWORD Value ) { return TRUE; }

   // APP1.H
   #define ACM_WINSTR                              0x500
   #define ACM_DIALOG                              0x501
   #define ACM_POPUP                               0x502
   #define ACM_NOTHING                             0x503
   #define ACM_HFONT                               0x504

   // APPR1.R
   #include "app1.h"
   #include "tiams.h"
```

```
107
108   TOOLBOX AppMenu, RC_NO_IDS, 0, 240 {
109      "Actions" {
110         "WinStr",      ACM_WINSTR
111         "Dialog",      ACM_DIALOG
112         "Popup",       ACM_POPUP
113         "Grayed-out",  ACM_NOTHING
114      }
115      "HFONT",         ACM_HFONT
116   }
117
118   DIALOG tDialog, 180, 70, NoCallBack {
119      EDIT, {DF_TAB_SPACES, 12, 14}, "EDIT1", 0, 10, 11
120      EDIT, {DF_TAB_SPACES, 12, 24}, "EDIT2", 11, 10, 11
121      POPUP, {DF_TAB_ELLIPSES, 12, 34}, "FIRST POPUP", Popup1, 0
122      POPUP, {DF_TAB_ELLIPSES, 12, 44}, "2ND POPUP", Popup2, 1
123      HEADER, {0, 0, 0}, "DIALOG HEADER", PDB_OK, PDB_CANCEL
124   }
125
126   POPUP Popup1, RC_NO_IDS, 0 {
127      "Item 1-1", 1
128      "Item 1-2", 2
129   }
130   POPUP Popup2, RC_NO_IDS, 0 {
131      "Item 2-1", 1
132      "Item 2-2", 2
133      "Item 2-3", 3
134   }
```

11.6.1. Files in Example and Explanation of Details

app1.c	The C source file.
app1.h	The header file contains definitions used by app1.c and appr1.r.
appr1.r	The resource file contains the menus, dialogs, and pop-ups.

Lines 2 . . . 4	Specifies the standard header tiams.h, our header file app1.h, and the resource file appr1.h (generated by the resource compiler).
Lines 6 . . . 12	All apps must have a frame that defines the interface between it and the system. This app's menu is handled by the system because of line 11 (OO_APP_DEFAULT_MENU). See chapter **7. Flash Application Layout** for more details.
Line 14	The first data item must be a pFrame to the apps FRAME.
Lines 15, 16	The apps static global data.
Line 18	Every interactive app must have an event handler. See chapter **9. Application Control Flow** for a complete description of system events.
Line 19	Required if a function accesses any data in the AMS (the isprint macro accesses the global **CTypeTable**).
Lines 20 . . . 23	The event handler's local data.
Lines 26 . . . 34	When this app is started it finds the size of its current window from the start-up message; opens a window using **WinOpen** (note that the size and position of the window in the start-up message assumes the app has a menu); and then initializes any of its global data.
Line 35	The app is activated after it receives the start message and whenever it is switched to in split screen mode.
Line 36	Since we are letting the system handle our menu this call to **EV_defaultHandler** turns our menu on.
Line 37	Now that our menu is on we can disable/enable or check/uncheck any items, in this case the item with ID ACM_NOTHING is disabled.
Line 38	**WinBeginPaint** saves the current system draw status (current font, attribute, clipping, . . .) since another app may have switched states.
Line 39	Activates our app's window (the system puts our window at the topmost position of all windows) and highlights our window border (if present).

Lines 42, 43	When another app is switched to in split screen mode or our app is about to be closed a deactivate message is sent and we restore the saved screen state.
Lines 45 . . . 50	The quit message signals our app is about to be closed. We close our window if it was opened (all opened windows must eventually be closed) and mark it as closed.
Lines 51 . . . 57	Echo printable characters to our window, let the system respond to any other keys (if we did not the system would not work).
Lines 58 . . . 61	We receive a paint message when the system wants us to redraw our window. The WF_DUP_SCR in our call to **WinOpen** tells the system to make a backup of all of our writes to our window (at the cost of extra memory and time) and so all we have to do is redraw our border and call **WinBackupToScr** to copy the duplicate screen image to the display.
Lines 62 . . . 64	ACM_WINSTR is a menu ID defined in app1.h (line 98) and stored in our menu (line 110). Application menu IDs range from 0x500 . . . 0x6FF and are used to return values to an app. Since it is stored in our menu and the system is handling our menu we get the command automatically when the user selects it from our menu. In this case we just draw a string to our window.
Lines 65 . . . 67	ACM_DIALOG (lines 99, 111), another of our menu IDs is sent when it is selected from our menu to call our dialog (defined on lines 118 . . . 134). We then setup the parameters for our dialog. The edit field parameters were set on lines 32 and 33 in response to our start-up message and so their values will only change when the app is started. The array *opts* contains the values passed to and returned for the two dropdowns (lines 121, 122 and lines 126 . . . 134) in the dialog box.
Lines 68 . . . 70	If [ENTER] is pressed to close the dialog box ([ESC] is the other valid key) we display the values selected using **DlgNotice**. **DlgNotice** creates a dynamic dialog box and places the strings we pass along with [ENTER] and [ESC] buttons.
Lines 73 . . . 77	Here we create a dynamic pop-up using the values entered in the dialog box and one additional static item. Instead of using predefined menu IDs we pass 0 (the second parameter to **DynMenuAdd**) so that they are assigned sequentially (starting with 1).

Lines 78 . . . 82	If there were no errors adding the dynamic elements to the pop-up (memory full), we execute the pop-up and display the results as with the dialog box.
Lines 83	The memory for this pop-up was created dynamically so it must be freed when we are done with it.
Lines 86 . . . 88	One additional menu item lets the user change the font for our window.
Lines 89 . . . 91	Any messages we do not handle must be passed onto the system so they get handled by someone.
Line 95	All dialog boxes must have a call-back function.
Lines 97 . . . 102	Our predefined menu IDs used by both app1.c and appr1.r.
Lines 104 . . . 106	Our resource file, it includes our predefined menu IDs and the standard include file tiams.h also.
Lines 108 . . . 116	TOOLBOX defines a menu. The RC_NO_IDS flag says we will assign all menu IDs. A 0 tells the resource compiler to use the names we give as symbolic menu IDs and assign them sequential values.
Line 118	We define a 180 by 70 pixel dialog box with a call-back function named NoCallBack (defined on line 95).
Lines 119, 120	There are two text edit fields each 10 characters in length with space for 11 characters displayed (the extra space means the edit field will not scroll when the last character is hit). The first edit field will start at offset 0 of the buffer we pass to the **Dialog** function and the next edit field will start at offset 11 (must leave room for the zero byte terminator). Each edit field starts 12 pixels from the left edge of the dialog box and the first edit field starts 14 pixels from the top while the second edit field starts 24 pixels down.
Lines 121, 122	There are two pop-ups with coordinates spaced the same as the edit fields (12 pixels from the left edge and 10 pixels apart).
Lines 126 . . . 134	The pop-ups for the dialog box are defined here (they can also be used separately as static pop-ups).

This example can be built using the TI **FLASH** Studio™.

12. Basic Text Editing Facility

The basic text editing facility simplifies the entry and editing of text. The text editor supports cursor blink, text entry, and cut, copy, and paste through the clipboard.

12.1. How to Edit Text

Begin by creating a text edit record and attaching it to a window. The text edit record keeps track of the edit buffer contents, cursor location, selected text, and other details of presenting the text in a window. Routines **TE_open** and **TE_openFixed** initialize a text edit record and attach it to a window.

The edit buffer can be variable- or fixed-length. Use routine **TE_open** to create a variable-length edit buffer. The text editor maintains a handle to the edit buffer and expands or contracts the buffer automatically as text is added or deleted. The edit buffer always contains at least one byte for the null end-of-string terminator.

Use routine **TE_openFixed** to initialize a text edit record with a fixed-length edit buffer. Unlike **TE_open**, this routine cannot allocate a buffer if the app does not provide one. **TE_openFixed** must be given a pointer to a buffer big enough to contain the longest expected data plus room for one null end-of-string terminator.

The text editor handles keypresses through the **TE_handleEvent** routine. Pass to **TE_handleEvent** an event received by the app from the OS. The text editor processes the event and takes care of inserting or deleting text, highlighting selected text, cut, copy, paste, repainting, and cursor blink messages. **TE_handleEvent** returns TRUE if it handled the event. The application should pass the event to **EV_defaultHandler** if the text editor returns FALSE.

To do anything useful with the edited text, the application should look at CM_KEY_PRESS events before calling **TE_handleEvent**. The application should respond to keys such as [ENTER] to terminate the edit and act on the contents of the edit buffer.

Routine **TE_shrinkWrap** releases slack memory in a variable-length edit buffer, cancels selection highlight, turns off the cursor, and returns a handle to the edit text. The edit buffer memory is not freed. Use this routine to prepare the edit buffer for further processing in the app or to be stored as a variable.

Call **TE_reopen** to reopen a text edit record which has been closed by **TE_shrinkWrap**. This routine selects and highlights all the text in the edit buffer.

Call **TE_close** to close out a text edit record and release its memory.

12.2. Simple Text Edit Example

This simple application displays a single menu on the menu bar with choices for cut, copy, and paste. It responds to typing and cursor arrow keys.

File teappr.r:

```
#include "tiams.h"

TOOLBOX teappMenu, RC_NO_IDS, 0, 160 {
   XR_Tools {
      XR_CutO,                    CM_CUT
      XR_CopyO,                   CM_COPY
      XR_PasteO,                  CM_PASTE
   }
}
```

File teapp.c:

```
#include "tiams.h"
#include "teappr.h"

void main(pFrame, Event *);

FRAME(teappFrame, OO_SYSTEM_FRAME, 0, OO_APP_FLAGS, 4)
   ATTR(OO_APP_FLAGS, APP_INTERACTIVE)
   ATTR(OO_APP_NAME, "Text Edit Demo")
   ATTR(OO_APP_PROCESS_EVENT, &main)
   ATTR(OO_APP_DEFAULT_MENU, &teappMenu)
ENDFRAME

pFrame TeappFrame = (pFrame)&teappFrame;

WINDOW myWindow;
TERecord myTE;

void main(pFrame self, Event *event)

{
   WIN_RECT teRect;
   HANDLE hText;

   switch (event->command)
   {
      case CM_START:
         WinOpen(&myWindow, event->info.startInfo.startRect, 0);
         DrawWinBorder(&myWindow, &myWindow.Window);
         WinClr(&myWindow);

         /* Create an empty text edit field.

            The text field occupies one line at the top of the window.
            Text will scroll within the edit field if it is too long to
            be displayed. Arrows appear at either end to indicate some
            text has scrolled out of the edit field.
         */
```

```
            teRect.x0 = 0;
            teRect.y0 = 2;
            teRect.x1 = WinWidth(&myWindow);
            teRect.y1 = teRect.y0 + LF_HEIGHT - 1;

            TE_open(&myTE, &myWindow, &teRect, H_NULL,
               0, 0, TE_NOWRAP|TE_MORE_ARROWS);
            break;

         case CM_QUIT:
            /* Get text from edit record */
            hText = TE_shrinkWrap(&myTE);

            /* Do something with text
               :
            */

            /* Close text edit record */
            TE_close(&myTE);

            WinClose(&myWindow);
            break;

         case CM_ACTIVATE:
            DrawWinBorder(&myWindow, &myWindow.Window);
            EV_defaultHandler(event);
            break;

         default:
            /* Allow the text editor the first attempt at handling the
               event. If it does not know how to handle the event, then
               pass it on to the OS default event handler.
            */
            if (! TE_handleEvent(&myTE, event))
               EV_defaultHandler(event);
      }
   }
```

12.3. Clipboard

The text editor moves text between the clipboard and edit buffer in response to commands CM_CUT, CM_COPY, and CM_PASTE.

An application can manipulate the clipboard directly using routines **CB_replaceTEXT** and **CB_fetchTEXT**. Any text placed in the clipboard by an app with a call to **CB_replaceTEXT** can later be pasted into a text edit field. Likewise, text placed in the clipboard by the text editor can be retrieved by the app with a call to **CB_fetchTEXT**.

`

13. Memory Management

13.1. The Heap (Dynamic RAM Storage)

The heap is the place where all dynamic data is stored. The heap is organized around the use of handles. A handle is a WORD (unsigned 16 bit quantity) that references a block of heap memory. Handles are used so that the heap blocks can be compressed by garbage collection when the heap becomes fragmented. In order to use a handle it must first be dereferenced. The dereferenced handle points to the data in the heap. This pointer is valid as long as nothing is done that could cause the heap to be compressed. The following routines may cause the heap to be compressed (the caller must also be aware of other system routines that call these routines): **HeapAlloc**, **HeapAllocThrow, HeapAllocHigh**, **HeapAllocHighThrow, HeapCompress**, **HeapMax**, **HeapMoveHigh**, **HeapRealloc**, and **HeapShuffle**. After any of these routines are called (either directly or indirectly), any dereferenced handles must be dereferenced again; otherwise, the data pointed to by the dereferenced handle may have been moved. Most estack routines may cause heap compression. If necessary, a heap block can be locked. Once locked, a heap block will not be moved. This should only be done when necessary since it causes garbage collection to be inefficient, creating the possibility that insufficient RAM will be available when needed. Heap allocation routines accept a DWORD (double-word) length value but the AMS implementation limits heap allocations to 65,532 bytes.

For an application, all of its data is either stored in the heap (directly or using routines like the file system) or in the applications static data area. Data placed in the heap remains even after an app is terminated. All handles returned from calling any of the heap allocation routines must be freed before an app is terminated. Files and variables do not have to be deleted but remain until deleted by the app, deleted by the user or the system is reset.

HeapAlloc, HeapAllocThrow, HeapAllocHigh, HeapAllocHighThrow	—	Allocate memory from the heap.
HeapAvail	—	Return the total amount of free bytes in the heap.
HeapCompress	—	Coalesce all used heap blocks, deleting any free blocks from the heap if possible (garbage collection).
HeapDeref	—	Dereference a handle, returning a pointer to the data.
HeapFree	—	Free a heap block given its handle.
HeapFreeIndir	—	Free a heap block given the address of a handle.

HeapGetLock	—	Return TRUE if a block is locked, otherwise return FALSE.
HeapLock	—	Lock the block referenced by the given handle so that it is NOT moved during garbage collection.
HeapMax	—	Return the maximum block allowable in the heap (does garbage collection).
HeapMoveHigh	—	Try to reallocate a block as high in memory as possible.
HeapPtrToHandle	—	Find the handle to a given pointer.
HeapRealloc	—	Reallocate the size of a heap block (smaller or larger).
HeapShuffle	—	Move blocks of memory around (for debugging).
HeapSize	—	Return the number of bytes allocated for the given heap block.
HeapUnlock	—	Unlock the block referenced by the given handle so that it can be moved during garbage collection.
HLock	—	Lock and dereference a handle.

13.2. File System

The File routines provide a convenient way to store application data. Each active file must have an associated FILES structure. Files are stored in the symbol table (see section **13.3 Managing Variables**) and as such have a semipermanent life.

Filenames used as parameters are not tokenized variable names as required by the symbol table code, but are a string of characters. They must not be reserved names. If a filename does not have a folder name then it will be stored in the current folder. Internally, files are stored as third-party data types (GEN_DATA_TAG). This type is further defined by a file description which may be up to four letters. This will show up to the user in the VAR-LINK screen as the type specified when the file was opened.

When a file is opened with **FOpen** in FM_WRITE or FM_APPEND mode the associated variable is locked and inaccessible by any other routines in the system. It must be closed with **FClose** to return the variable to not in-use mode, to write the file type and the GEN_DATA_TAG, and to close the associated buffer. For files opened in FM_READ mode, the **FClose** will merely update the mode of the file in the FILES structure to closed and clear the associated error status.

There is no separate mode to open a file for both reading and writing. However, if a file is opened in FM_APPEND mode the contents of the file are not erased. All locations in the file can either be read from or written to (random access).

The file routines are:

FClose	—	Close a file, this is required for files opened in write mode.
FCreate	—	Create an empty file.
FDelete	—	Delete a file.
FEof	—	Return TRUE if a file is at the End of File mark.
FFindFirst, FFindNext	—	Used to search all files for a specific file type.
FGetC	—	Return a byte from an open file.
FGetPos	—	Return the current file position.
FGetSize	—	Return the number of bytes stored in an opened file.
FOpen	—	Open a file for reading, writing or both.
FPutC	—	Write a byte to a file opened in write mode.
FRead	—	Read multiple bytes from a file.
FSetBufSize	—	Set the size of the write buffer for an opened file.
FSetPos	—	Set the position of the next read or write for an opened file.
FSetSize	—	Truncate the size of a file opened in write mode.
FSetVer	—	Change the version number of an opened file.
FStatus	—	Return the status of an opened file.
FType	—	Return the file type (description field) of a file.
FWrite	—	Write multiple bytes to an opened file.

13.2.1. Opening Multiple Files for WRITE Mode

This section is only relevant if you plan on opening multiple files simultaneously in write mode. It requires some understanding of variables, specifically HSYMs which are explained in the next section. The FILES structure contains an HSYM of the opened file. HSYMs can become invalid whenever a new symbol table entry is added or a previous one is removed. In the FILE system the problem of HSYMs becoming invalid is only a problem if multiple files must be opened at the same time in WRITE or APPEND mode. Since the file system stores the HSYM of all opened variables in the FILES structure (this is needed for when the file is closed), creating a new file (which is just a symbol) with an existing file opened in WRITE mode may invalidate the previous file's HSYM. To get around this problem in the file system just use **FCreate** to create multiple files that must be opened simultaneously in WRITE mode before they are opened. This will insure

that the **FOpen** calls to open the files will not invalidate any previous HSYMs since the variables will have already been created.

13.3. Managing Variables

All variables are stored in the symbol table. The symbol table is setup as follows. There is one home folder which may contain variables or folders, although it only contains folders in the current system. It is a dynamically (in the heap) maintained array of SYM_ENTRYs. In the case of a folder the value pointer points to another dynamically allocated array of SYM_ENTRYs (one level) otherwise it points to the symbols value in the heap. Folder *main* always exists in this home folder and cannot be deleted. Variable and folder names are eight characters maximum. Names are arranged alphabetically within each array of SYM_ENTRYs. When a variable whose data is stored in Flash is executed or displayed in an application (that is, it is “in-use”), a copy of it is stored in RAM and a duplicate SYM_ENTRY (called a twin) is temporarily created immediately preceding the original SYM_ENTRY.

The SYM_ENTRY structure is defined as follows.

```
typedef struct {
   BYTE Name[8];
   BYTE MUST_BE_0;
   BYTE Version;
   FLAG_TYPE Flags;
   HANDLE hVal;
}  SYM_ENTRY;
```

- Name is a one to eight byte zero-terminated name.
- Version is set by the system to one of the values: TV_TI_92, TV_PARM , TV_INTERNAL, TV_NGIN, TV_SPAM, TV_CRAM, TV_3RDPARTYAPP, TV_SCRAM.
- Flags (type WORD) may be one of the values: GRAPH_REF1, GRAPH_REF2, SF_STAT_REF, SF_LOCK, SF_INUSE, SF_SELECTED, SF_RECEIVED, SF_FOLDER, SF_INVIEW, SF_EXTMEM, SF_EM_TWIN, SF_COLLAPSE, SF_PARM. The system normally handles the flags for a symbol. An app may set the SF_INUSE flag bit to signal it is using a variable and that it should not be used by another app.
- VAR_LINK will not display any variables with their SF_INUSE bit set or those that have an hVal of NULL. Setting the SF_LOCK bit will prevent **VarStore** from writing to the variable.
- The hVal is the handle of the data for the symbol.

In general, symbol table routines do not return direct pointers to SYM_ENTRYs (the exceptions being **SymFindFirst**, **SymFindNext**, and **SymFindPrev**).

Instead they return an HSYM value. An HSYM is a combination of a handle to the folder of a particular symbol along with the offset into the folder of that symbol. The symbol's HSYM value is valid as long as no other symbols are added to or removed from the symbol's folder. This is because symbols are kept alphabetically and adding a new symbol may affect the offset of any other symbols in the symbol's folder. To convert an HSYM value to a SYM_ENTRY pointer use the function **DerefSym**. The dereferenced HSYM value is a direct pointer into the heap and so it is valid only as long as garbage collection is not done.

Symbol and folder names are usually passed in token format. The exceptions are the low-level symbol and folder routines, which should be avoided. The symbol/folder pointer points to the tag (usually zero) at the high address with any additional bytes stored from high to low memory. So if the symbol *A23456* were tokenized it would be stored with a zero byte followed by the name followed by a second zero byte. Capital letters A-Z are always converted to lower case when tokenized in variable names. If this symbol were passed to a symbol routine, the address of the second zero byte would be passed. Note that, in order to save space, one byte variables in the range 'a' to 'z' have a single one-byte token value.

0	'a'	'2'	'3'	'4'	'5'	'6'	0

Figure 13.1: Token Representation of VarName A23456

Symbols are located in the symbol table by using the following strategy. First the system variables are searched. Then, if the symbol is not stand-alone (has an embedded folder name or implied current user folder or a folder name is passed as an argument — such as **AddSymToFolder**) the given folder is searched. If the symbol is stand-alone, then the folders are searched in the following order: the current temporary folder if one exists, and then the current user folder (default folder).

When a stand-alone symbol is added to the symbol table the current temporary folder is not searched for an existing name. The only way to add a variable to a temporary folder is with the **AddSymToFolder** function and specify the desired temporary folder name. Otherwise, stand-alone symbols are always added to the default folder. The symbol name structure is shown below:

[folder] [\ ] [name]

The system reserves certain names for itself. These reserved names include:

- System variables (xmin, xmax, medx1, . . .)
- Reserved function names (y1, y99, xt1, . . .)
- System commands (AndPic, BldData, Circle, . . .)
- System functions (abs, sin, cos, . . .)

The system variables and reserved function names is a finite list that is defined in Appendix B of the TI‑89 / TI‑92 Plus Guidebook and cannot be used for anything else. The system commands and functions are also reserved; but an application localizer can redefine their spelling for a particular language. Thus, the list of system commands and functions is open-ended. In order to allow for redefined system command and function names, an app can append a digit to each name it uses which will insure that the name does not conflict with any reserved names (as long as it does not conflict with a system variable or reserved function name).

As an example: In English, an app can create a variable called EXAKT and store any value to it. If the same app is run in German, storing to the variable EXAKT will cause an error because that is the redefined system function for “exact(”. But the app could store to EXAKT0 instead and not worry about a name conflict.

13.3.1. Normal Symbol Routines

In general, an app should use the file system to store any permanent data. The low-level symbol table routines assume the caller knows precisely which variables are being worked on. Thus, these routines do limited checking for things like locked, in-use or invalid variable names as well as no type checking. The TI‑BASIC variable routines are available to call and, unlike the low-level routines, they do extensive checking. Their parameters are passed on the estack and all errors cause exceptions. These routines are:

cmd_archive — Archive one or more variables.

cmd_copyvar — Copy one variable to another.

cmd_delfold — Delete one or more empty folders.

cmd_delvar — Delete one or more variables.

cmd_lock — Lock one or more variables.

cmd_movevar — Move a variable from one folder to another.

cmd_newfold — Create a new folder.

cmd_rename — Rename a variable or folder.

cmd_unarchiv — Unarchive one or more variables.

cmd_unlock — Unlock one or more variables.

DerefSym — Dereference an HSYM, returning a pointer to a SYM_ENTRY.

FolderCur — Set current default folder.

FolderGetCur — Get current default folder.

push_getfold — Get current default folder (TI‑BASIC version of **FolderGetCur**).

push_setfold — Set current default folder (TI-BASIC version of **FolderCur**).

TokenizeSymName — Tokenize a name and check for invalid or reserved names.

VarRecall — Look-up a variable (returning its HSYM).

VarStore — Store a value to a variable.

In order to use most of these routines, the variable names must be tokenized and pushed onto the estack. **TokenizeSymName** does this as well as checking for invalid names. There are only two routines for storing to and retrieving variable values: **VarStore** and **VarRecall**. This is because there are many system variables that are not in the symbol table (system variables are only accessible through **VarStore** and **VarRecall**). **VarStore** also does extensive type checking to protect certain variable types. For example, it is illegal to copy anything but a program to another program or a data variable to another data variable. **VarStore** must also insure that certain system variables are only stored to in the proper mode. As an example, *tmin* which is used in parametric graphing, can only be stored to in parametric graph mode and it has a limited range of values that can be stored to it. **VarStore** does all of the necessary type and value checking for all variables. To reiterate, the low-level routines do no type or value checking and should be used with extreme caution.

Note: Routines that take a variable number of arguments (**cmd_delvar**, **cmd_delfold**, **cmd_lock**, **cmd_unlock**, **cmd_archive**, **cmd_unarchiv**) require an END_TAG on the estack to mark the end of the parameter list of variable names passed to them.

13.3.2. Storing and Retrieving Variable Data

As noted earlier, the file system is the preferred method for an app to store or retrieve data. **VarStore** is the routine to store to a variable, and **VarRecall** is the routine to access a variable. These routines are complicated because of the built-in graphing application and the restrictions on the predefined system variables. When a variable is accessed while graphing or building a table, a flag in the symbol table entry for that variable is set. Whenever a store (or any change, including delete, rename, etc.) is done to a variable that already exists, the flags (one for each of the two possible graphs) must be checked to know if the graph and table are now incorrect because the variable changed. Other actions can also cause the graph or table to be incorrect — for example, changing the angle mode or folder.

It is even more important to use **VarRecall** and **VarStore** for system variables, some of which are not in the symbol table. In addition to the graph reference flags, the type or range of a value must be checked before storing to some of the system variables, and storing to some of them causes other system

variables to automatically change. Care must be taken when **VarRecall** or **VarStore** are bypassed by reading or writing directly to any variable using the low-level symbol table routines. The file system uses **VarStore** and **VarRecall**.

13.3.2.1. Store and Recall Look-up Paths

There are two types of folders: user folders and local (temporary) folders. There is always a current user folder which can be selected on the mode screen. The default current folder is the main folder and differs from any other user folder only because it cannot be deleted. A temporary folder is created and named by the system anytime a user function or program begins execution. Each new function/program creates another temporary folder. When the function/program is complete, the corresponding folder is deleted. The only variables in the temporary folder are the parameters of the function/program and any variables listed in a LOCAL command. System variables are considered to be outside the folders (even though some, such as the graph functions, are actually in the main symbol table). A specific user variable can be accessed from anywhere by including the path (user folder and backslash) with the name. For example, folder name\variable name. The following look-up paths are for the general case and do not include all the flags, conditions, etc. that must be checked once the variable has been found. See sections **13.3.2.2 Recall Look-up Path** and **13.3.2.3 Store Look-up Path** for details.

13.3.2.2. Recall Look-up Path

- Is the variable a system variable? These are special cases and cannot have a specified path.
- If a complete path is specified, return the HSYM handle for that variable, or if it does not exist, return H_NULL.
- If there is a backslash followed by the variable name, return the HSYM handle for that variable in the current user folder, or H_NULL if it does not exist.
- If there is not a path specified, check for that variable name in the current temporary folder. If it exists, return the HSYM handle. There may be no data associated with the variable yet if it was created by a LOCAL command but has not been initialized.
- If the variable does not exist in the current temporary folder, check the current user folder. If it exists, return the HSYM handle.
- Otherwise, the variable does not exist in the current path and H_NULL is returned.

13.3.2.3. Store Look-up Path

- Is the variable a system variable? These are special cases and cannot have a specified path. Many have restrictions on the domain and type of data allowed.
- If a complete path is specified and that variable already exists, replace the previous contents with the new contents (after verifying all flags, data type, etc.). Otherwise, create that variable with the given value.
- If there is a backslash followed by the variable name and that variable already exists in the current user folder, replace the previous contents with the new contents (after verifying all flags, data type, etc.). Otherwise, create that variable with the given value.
- If there is not a path specified, check for that variable name in the current temporary folder. If it exists, replace the previous contents with the new contents (after verifying all flags, data types, etc.).
- If the variable does not exist in the current temporary folder, check the current user folder. If it exists, replace the previous contents with the new contents (after verifying all flags, data types, etc.). Otherwise, create that variable in the current user folder.

13.3.2.4. HSYM VarRecall (BYTE **Var*, RECALL_FLAGS *Flags*)

VarRecall looks up a variable returning its HSYM or H_NULL if not found. **VarRecall** handles system variables even if they are not in the symbol table. *Var* is a pointer to the terminated zero of the tokenized variable name. *Flags* can have the following values although 0 and VR_NO_SYS_VARS are the two flags normally used.

Flags		
0	—	No restrictions.
VR_LINK	—	Used by link code only.
VR_FUNC_NAME	—	“y1(” entered, not “y1”.
VR_NO_SYS_VARS	—	Do not return system variables.

The recall routine returns the HSYM handle to the symbol table which remains valid until a variable in the same folder is added, deleted, or renamed. For the system variables that are not stored in the symbol table, there is one dummy symbol table entry. When a system variable not in the table is referenced, that entry will point to a copy of the desired system variable (with a tag added) and that handle will be returned. Since this one dummy entry is used for many system variables, the calling routine may have to copy the contents if it desires to have access to more than one value at a time. There is another function, **HToESI**, that will return a pointer of the type EStackIndex to the data type tag.

For example:

```
HSYM hsym;                    /* handle */
SYM_ENTRY *symp;              /* pointer */
EStackIndex estackIndex, nPtr;  /* pointer */
hsym = VarRecall( nPtr, 0 );  /* where nPtr is the pointer to the
                                 variable name in tokenized format */
if( hsym ) {
   symp = DerefSym( hsym ) ;  /* symp points to symbol table entry */
   if( symp->hVal ) {  /* hVal is the handle to the data part of the
                          variable which can also be null. */
      estackIndex = HToESI( symp->hVal );
      /* estackIndex points to the last byte of the data,
         which contains the type tag.
      */
.
.
.
```

VarRecall may throw the following errors:

INVALID_PATHNAME_ERROR	—	Invalid variable name.
ER_FOLDER	—	The variable is a folder.
ER_INVALID_VAR_REF	—	The variable cannot be referenced in the current mode. Some variables are only accessible by the application that created them, like C1 . . . C99, which can only be accessed by the data matrix editor. Other variables can only be accessed under certain conditions, like the stat variables, which can only be accessed if a stat calculation has been made.
ER_RESERVED	—	The VR_NO_SYS_VARS flag was set and the variable to be recalled was a system variable.
ER_UNDEFINED_VAR	—	Normally, H_NULL is returned if the variable is not found. This error is thrown if an attempt is made to execute an undefined function while graphing.

Most system variables cannot be used in a function that is being graphed, either because they change too often or because they are used by the graph routines themselves (for example, xmin or xc). When these variables are accessed while graphing, an error is reported by **VarRecall**. Locked variables may be looked up with **VarRecall**.

If a variable's in-use flag (SF_INUSE) is set, the variable is being used by an application. Depending on the application, the handle to the data may be null or the data may not be in a useable form. It is up to the caller to test the SF_INUSE flag and take appropriate action.

13.3.2.5. HSYM VarStore (BYTE **DestVar*, WORD *Flags*, WORD *SourceSize* [, *parm1*] [, *parm2*] [, *parm3*] . . .)

VarStore stores values, prepares a variable to be stored to, or stores individual values to elements of a list or matrix. The data type and domain of system variables are verified to be correct. System variables not in the symbol table will return H_NULL. Variables in the symbol table will return their HSYM handle. The *Flags* parameter determines the meaning of the remaining parameters.

Flags

STOF_ESI	—	*parm1* is an EStackIndex pointing to a locked block of memory, most likely the estack.
STOF_HESI	—	*parm1* is a handle to a block of memory containing the data to store (will be locked initially).
STOF_ELEMENT	—	*parm1* is an EStackIndex pointing to the element to store. For a list, *parm2* is a WORD indexing the element to store to and *parm3* (also a WORD) must be zero. For a matrix, *parm2* indexes the column of the matrix to store to and *parm3* indexes the row. The indexes for both lists and matrices start at one.
STOF_NONE	—	Nothing is assumed about the source, no copy is done (left to caller). It creates the symbol table entry, verifies the name and flags, and other conditions.
USER_FUN_TAG	—	*parm1* points to the USER_FUN_TAG of the function to store.

Otherwise *Flags* must equal TEXT_VAR_TAG, GDB_VAR_TAG, PIC_VAR_TAG, DATA_VAR_TAG, or GEN_DATA_TAG. The destination is verified to have the same type as the source. The copy operation is not done.

SourceSize is the size of the source data including the tag but not the size word stored in the heap. If it is zero then the size of the estack expression pointed to by *parm1* will be used for STOF_ESI, STOF_HESI, and STOF_ELEMENT. Otherwise, the new value is not allocated (return value->hSym may still not be H_NULL if previous data existed in that variable). For STOF_ESI, STOF_HESI, and STOF_ELEMENT the source is copied to the new destination.

For TEXT_VAR_TAG, GDB_VAR_TAG, PIC_VAR_TAG, DATA_VAR_TAG, or GEN_DATA_TAG the source is not copied but the destination is verified to have the same data type (TEXT, GDB, PIC, DATA, or OTH). If *SourceSize* is not zero then the destination is allocated to the given size; it is up to the caller to do the actual copying of data.

VarStore may throw the following errors:

ER_DATATYPE, ER_DOMAIN	—	The value stored is in the wrong domain or of the wrong type for the variable being stored to.
ER_DIMENSION	—	Illegal index when storing to a list or matrix.
ER_FOLDER	—	The variable is a folder, which cannot be stored to.
ER_ILLEGAL_IN_FUNC	—	Functions can only store to local variables.
ER_INVALID_VAR_REF	—	The variable cannot be stored to in the current mode, see **VarRecall** also.
ER_LOCKED, ER_VAR_IN_USE	—	The variable is locked or in-use and cannot be stored to.
ER_MEMORY	—	Not enough memory to do the store operation.
ER_PROTECTED	—	The variable cannot be stored to by the current app in the current mode.
ER_RESERVED	—	The given variable is a system reserved variable and the value to be stored is invalid for this variable.
INVALID_PATHNAME_ERROR	—	Invalid variable name.

13.3.2.6. General Data Storage

Most of the data that will be stored in a variable will be on the estack or in a buffer in the heap. To store floating-point data contained in a C variable, use **push_Float** (var) and use **top_estack** as the EStackIndex for **VarStore**. This automatically rounds the mantissa to 14 digits and adds the float tag. The caller must remember to restore **top_estack** to its original value.

Example:

```
BCD16 flt;
const BYTE Name[] = { '\0', 't', 'e', 'm', 'p', '\0' };
EStackIndex old_top = top_estack;

push_Float( flt );  /* round to 14 digits, add float tag */
VarStore(  Name+5, STOF_ESI, 0, top_estack );
top_estack = old_top;  /* restore original top_estack */
```

Some variables are system protected. These include programs and functions (TI-BASIC or ASM), data variables, graph databases, third-party data-types

(includes files), pictures, and text variables. Only variables of the same type can be copied to a system-protected variable.

If the variable name already exists, the lock flag is checked — a variable that is locked or archived cannot be overwritten. **VarStore** throws an error in this case. System variables cannot be locked and some system variables cannot be changed by the user. **VarStore** also checks the in-use flag. Any variable being used by an application cannot be overwritten except by that application. It cannot be deleted, renamed, or linked either. The in-use count is verified to be 0 for functions and programs.

If the variable is not a system variable, locked, system protected, or in use, a value can be stored to it no matter what it contained before. However, there are type restrictions on individual elements of lists or matrices. Usually, **VarStore** makes sure there is enough memory left to store the new contents to the variable before deleting the current contents of the variable. When storing to an existing variable, both graph reference flags are tested. If one or both is set, the corresponding dirty graph and dirty table flag(s) are set to indicate the graph and table are no longer valid.

Storing to an individual element or submatrix of a matrix or list does the same checks as for an existing variable and updates the length. In addition, the data type of the element needs to be correct (expression, relation, string).

13.3.2.7. System Functions

Only functions with the correct function argument can be stored in the variables reserved as system functions. These are y1(x) – y99(x), xt1(t) – xt99(t), yt1(t) – yt99(t), r1(θ) – r99(θ), u1(n) – u99(n), y1’(t) – y99’(t), and z1(x, y) – z99(x, y). They can be single line or multiline functions but an error is returned if the user tries to create any other data type with those names or if the number of arguments is incorrect. Since they are system variables, they cannot be locked and must be in the main folder. Empty functions are not valid and should not be added to the symbol table. This is true for user functions also.

13.3.3. Low-Level Routines

Low-level routines allow direct access to the symbol table with little data type, status checking, or regard to reserved names. There are general purpose utility routines, routines to directly manipulate folders (including temporary folders) and variables. As stated earlier, they do NOT use tokenized names but deal with names in C string format (the pointer to the first letter in the name is passed, not to the zero byte terminator as with tokenized names).

13.3.3.1. Utilities

HSymDel — Delete a variable given its HSYM.

HSYMtoName — Create a fully qualified (name and folder) symbol name from an HSYM.

MakeHsym — Create an HSYM given a SYM_ENTRY pointer and the handle of its folder.

ResetSymFlags — Apply a mask to the flag byte of all variables in the system.

SetOK — Set the global system variable OK to one or zero.

StrToTokN — Convert a zero terminated symbol name (ASCIIZ) into a tokenized format (does NOT handle reserved names).

13.3.3.2. Low-Level Folder Routines

AddSymToFolder — Add a symbol to a specific folder.

FindSymInFolder — Search for a symbol in a specific folder.

FolderAdd — Directly add a folder to the home folder.

FolderCount — Return the number of symbols in a folder.

FolderDel — Delete a folder (even if not empty).

FolderFind — Look for a folder.

FolderFlags — Set or clear flags in all folders in the system.

FolderOp — Lock or unlock a folder or all folders in the system with **HeapLock** or **HeapUnlock** so they will not move.

FolderRename — Rename a folder.

13.3.3.3. Low-Level Symbol Routines

SymAdd — Add a symbol to the symbol table.

SymDel — Delete a symbol from the symbol table.

SymFind — Look for a symbol in the symbol table.

SymFindFirst — Find the first symbol in a folder (or all folders).

SymFindFoldername — Return name of folder for SymFindFirst/Next.

SymFindHome — Find a symbol in the home folder.

SymFindMain — Find a symbol in the main folder.

SymFindNext — Find next symbol after calling SymFindFirst.

SymFindPrev — Find previous symbol.

14. Data Types

The TI AMS Operating System (OS) supports the following data types:

- Expressions
- Lists
- Matrices
- Data Variables
- Text Variables
- Strings
- Graph Databases
- Pictures
- Programs
- Functions
- Assembly Programs
- Third Party Data Types (FILEs)

Note that there are two additional data types: figures and macros. They are only supported by the, at one time, built-in geometry application which is now a separate application.

This chapter defines the structure of the data objects that may appear in each of the above data types. These data objects all have an embedded tag value which is always the last value stored in a data object. The first word (Most Significant Byte first) of every data object is the length of the object. This length does not include itself, so to find the tag associated with any object, add the length of the object to the starting address of the object plus one. There is a routine, **HToESI**, that given the handle of a data object will return a pointer to the tag byte for that object. Shown below is a list of the data tag values along with other associated tags (END_TAG, COMMAND_TAG, END_OF_SEGMENT).

Tags	Values
FUNC_BEGIN_TAG	23
PRGM_TAG	25
NONNEGATIVE_INTEGER_TAG	31
NEGATIVE_INTEGER_TAG	32
POSITIVE_FRACTION_TAG	33
NEGATIVE_FRACTION_TAG	34
FLOAT_TAG	35
STR_DATA_TAG	45
LIST_TAG	217
USER_DEF_TAG	220
DATA_VAR_TAG	221
GDB_VAR_TAG	222
PIC_VAR_TAG	223
TEXT_VAR_TAG	224
COMMAND_TAG	228
END_TAG	229
END_OF_SEGMENT	233
ASM_PRGM_TAG	243
GEN_DATA_TAG	248

Table 14.1: Data Tag Values

14.1. Expression

There are several different data objects that can appear as an expression.

14.1.1. Non-Negative or Negative Integers

Description	Bytes
Data length of integer	2
Non-negative/negative integer (bignum first) binary data	var. max 255
Length of data field	1
NONNEGATIVE/NEGATIVE_INTEGER_TAG	1

Table 14.2: Data Object for a Non-Negative or Negative Integer

14.1.2. Positive or Negative Fractions

Description	Bytes
Data length of fraction	2
Positive/negative denominator (bignum first) binary data	var. max 255
Length of denominator	1
Positive/negative numerator (bignum first) binary data	var. max 255
Length of numerator	1
POSITIVE/NEGATIVE_FRACTION_TAG	1

Table 14.3: Data Object for a Positive or Negative Fraction

14.1.3. Floating-Point Numbers

Description	Bytes
Data length of FP number	2
Exponent/sign	2 ❶
BCD mantissa value (MSD at lowest address)	7 (MSD->LSD)
FLOAT_TAG	1

Table 14.4: Data Object for a Floating-Point Number

❶ Bit 7 of the most significant (lower address) Exponent/Sign byte is the mantissa sign. The remaining 15 bits represent the exponent and exponent sign (0x4000 = exponent of 0, 0x3FFF = exponent of -1, 0x4001 = exponent of +1).

14.1.4. All Other Tags Not Listed Here

In general, an expression is any statement that starts with a tag not listed below. That is anything that is not a list, matrix, function, program, picture, string, text, graph database, assembly language program, or a third party data type (FILE). The preceding three types (integers, fractions, and floating-point numbers) are only special cases of expressions.

14.2. List

A list is a collection of expressions. A list may only contain expressions and nothing else.

Description	Bytes
Data length of list	2
END_TAG	1
Expressions (1 per element of list)	variable
LIST_TAG	1

Table 14.5: Data Object for a List

14.3. Matrix

A matrix is stored as a list of lists, guaranteed to have scalar elements resulting in a rectangular matrix. Each list represents one row of the matrix, surrounded by a LIST_TAG/END_TAG pair as shown below.

Description	Bytes
Data length of list	2
END_TAG	1
END_TAG (one for each row)	1
Expressions (1 per element of list)	variable
LIST_TAG (one for each row)	1
LIST_TAG	1

Table 14.6: Data Object for a Matrix

14.4. Data Variable

Description	Bytes
Length of Data Variable	2
Column Width (# chars -2)	1
Number of Columns (0-99, 0 = no column data)	1
Column Number	1
Length of Column List (MSB, then LSB)	2
List Contents	variable
LIST_TAG	1
. . . repeat above section for each column	variable
Number of Formulas (0-99, 0 = no formulas)	1
Column Number	1
Data Length (MSB, then LSB)	2
Formula Contents	variable
Expression (any) Tag	1
. . . repeat above section for each formula	variable
Number of Titles (0-99, 0 = no titles)	1
Column Number	1
Data Length (MSB, then LSB)	2
Title Contents (STRING)	variable
STR Tag	1
. . . repeat above section for each title	variable
DATA_VAR_TAG	1

Table 14.7: Data Object for a Data Variable

These blocks are repeated from zero to 99 times, depending upon the corresponding Number field immediately preceding the block.

14.5. Text Variable

Description	Bytes
Data length of text	2
Position of edit cursor	2
Text data for line with 0x0D terminator (text starts at the byte after the edit cursor position and ends at the byte before the zero byte terminator)	variable
0 (end of text)	1
TEXT_VAR_TAG	1

Table 14.8: Data Object for a Text Variable

Note that the first character of each line can be one of the following:

0Ch	Page Break character
'C'	Executable Calculator Command follows
'P'	Print Object (as in a Lab Report)

Table 14.9: Valid first characters for a Text Variable Data Object

14.6. String Variable

Description	Bytes
Data length of string.	2
0	1
String, stored left to right.	variable
0 (end of string)	1
STR_DATA_TAG	1

Table 14.10: Data Object for a String Variable

14.7. Graph Database

There are several differences between TI-92 graph databases and TI-89 / TI-92 Plus graph databases which are noted below. TI-92 graph databases can be sent to the TI-89 / TI-92 Plus but TI-89 / TI-92 Plus graph databases cannot be sent to a TI-92. Once it is received by the TI-89 / TI-92 Plus, a TI-92 graph database still has the TI-92 version number (version=TV_TI_92 in the SYM_ENTRY structure for that variable in the symbol table, see section **13.3 Managing Variables**), which allows it to be sent to another TI-92 from the TI-89 / TI-92 Plus.

Description		**Bytes**
Data length of Graph Database		2
Number of Graphs (1 = one graph mode, 2 = two graph mode)		1
Angle Mode (Radian = 1, Degree = 2)		1
Real/Complex Mode		1
Mode	**Value**	
Real	1	
Rectangular	2	
Polar	3	
Graph Mode — Graph 1 (if in two graph mode this is the graph on AP_SIDE_A)		1
Mode	**Value**	
Function	1	
Parametric	2	
Polar	3	
Sequence	4	
3D	5	
Differential Equations	6	
Active Side — two graph mode only (0 = **gr_active** pointing to graph 1, 1 = **gr_active** pointing to graph 2)		1 ❶
Graph Mode of Graph 2 — two graph mode only (see above graph modes for values)		1 ❶

Table 14.11: Data Object for a Graph Database

Description							Bytes
Split Setting — two graph mode only							1 ❶
	Description	**Setting**					
	Full	1					
	Horizontal	2					
	Vertical	3					
Split Ratio — two graph mode only							1 ❶
	Description	**Value**					
	Split Ratio 1:1	1					
	Split Ratio 1:2	2					
	Split Ratio 2:1	3					
Graph 1 Range Settings (based on Graph 1 Mode)							variable
	Function	**Parametric**	**Polar**	**Sequence**	**3D ❷**	**DifEq ❸**	
	xmin	xmin	xmin	xmin	xmin	xmin	
	xmax	xmax	xmax	xmax	xmax	xmax	
	xscl	xscl	xscl	xscl	xgrid	xscl	
	ymin	ymin	ymin	ymin	ymin	ymin	
	ymax	ymax	ymax	ymax	ymax	ymax	
	yscl	yscl	yscl	yscl	ygrid	yscl	
	Δx	Δx	Δx	Δx	Δx	Δx	
	Δy	Δy	Δy	Δy	Δy	Δy	
	xres	tmin	θmin	nmin	zmin	t0	
		tmax	θmax	nmax	zmax	tmax	
		tstep	θstep	plotStrt	zscl	tstep	
				plotStep	eyeθ	tplot	
					eyeϕ	diftol	
					eyeΨ	Estep	
					ncontour	fldres	
					xscale	ncurves	
					yscale	dtime	
					zscale		

Table 14.11: Data Object for a Graph Database *(continued)*

Description		Bytes
Graph 1 Formats		8 ❹
Description	**Value**	
2 bytes: Flags		
Seq Mode Axes: Time = 1, Custom or Web = 0	0x8000	
Seq Mode Axes: Web = 1, Custom or Time = 0	0x4000	
Seq Mode Web: Trace = 0, Auto = 1	0x2000	
3D Expanded View: Off = 0, On = 1	ox0800	
TI-92 3D Mode Style: WireFrame = 1, Hidden Surface = 0	0x0100	
Coordinates: Off = 1, On = 0	0x0080	
Graph Order: Sequential = 0, Simul = 1	0x0040	
Grid: Off = 0, On = 1	0x0020	
Axes: Off = 1, On = 0	0x0010	
3D Mode Axes: Normal = 0, Box = 1	0x0008	
Labels: Off = 0, On = 1	0x0004	
Leading Cursor: Off = 0, On = 1	0x0002	
Coordinates: Rect = 0, Polar = 1	0x0001	
1 byte: x axis for custom axes in Seq or DifEq Modes	see below	
1 byte: y axis for custom axes in Seq or DifEq Modes	see below	
Seq Mode: n = (-1), u = 0, specific u# function = 1-99		
DifEq Mode: t = 0, y = 100, specific y# function = 1-99, y' = -100, specific y#' = (-1) – (-99)		
2 bytes: Flags		
DifEq Mode Axes: Time = 0, Custom = 1	0x0010	
DifEq Fields: FLDOFF = 0, SLPFLD or DIRFLD = 1	0x0004	
DifEq Fields: SLPFLD = 0, DIRFLD = 1	0x0002	
DifEq Solution Method: RK = 0, Euler = 1	0x0001	
1 byte: TI-89 / TI-92 Plus 3D Mode Style		
Wire Frame	0	
Hidden Surface	1	
Contour Levels	2	
Wire and Contour	3	
Implicit Plot	4	
1 byte: Unused		

Table 14.11: Data Object for a Graph Database *(continued)*

Description		Bytes
Number of Graph 1 Functions (may be zero)		1
Function Number (set MSB for yt in parametric mode)		1
Graph 1 Function (copy from symbol table including length)		variable
Repeat above two lines for each additional function		. . .
Number of initial conditions (= 0 if not Seq or DifEq mode, or if none exist in Seq or DifEq mode)		1
Initial Condition Number		1
Initial Condition Expression (including length)		variable
Repeat above two lines for each additional initial condition		. . .
Graph 1 Table Flags (not included if 3D mode)		1
Description	**Value**	
Table connected to trace (on = 1, off = 0)	0x80	
Table Independent Ask (on = 1, off = 0)	0x40	
Graph 1 tblStart (not included if 3D mode)		10
Graph 1 Δtbl (not included if 3D mode)		10
Graph 1 tblInput (including length [2 bytes] — may be zero) (not included if 3D mode)		variable
Graph 2 Range Setting (if two graph mode) (see Graph 1 Range Settings for contents)		variable ❶
Graph 2 Formats (if two graph mode) (see Graph 1 Formats for contents)		8 ❶
Number of Graph 2 Functions (if two graph mode) (may be zero if same type as graph 1)		1 ❶
Function number (set MSB for yt in parametric mode)		1 ❶
Graph 2 Function (copy from symbol table including length)		variable ❶
Repeat above two lines for each additional function		❶
Number of initial conditions (= 0 if not Seq or DifEq mode, or if none exist in Seq or DifEq mode, or if graph 1 = same mode)		1 ❶
Initial Condition Number		1 ❶
Initial Condition Expression (including length)		variable ❶
Repeat above two lines for each additional initial condition		. . . ❶

Table 14.11: Data Object for a Graph Database *(continued)*

Description		Bytes
Graph 2 Table Flags (not included if 3D mode)		1 ❶
Description	**Value**	
Table connected to trace (on = 1, off = 0)	0x80	
Table Independent Ask (on = 1, off = 0)	0x40	
Graph 2 tblStart (not included if 3D mode)		10 ❶
Graph 2 Δtbl (not included if 3D mode)		10 ❶
Graph 2 tblInput (including length [2 bytes] — may be zero) (not included if 3D mode)		variable ❶
GDB Tag		1

Table 14.11: Data Object for a Graph Database *(continued)*

❶ This field is only present in a two graph mode graph database (i.e., Number of Graphs is equal to 2).

❷ eyeΨ and ncontour are not present on the original TI-92. xscale, yscale, and zscale are not system variables and are for internal use only (zscale was not present on the TI-92). The system variable zscl is no longer used on the TI-89 / TI-92 Plus although it still exists for compatibility with the TI-92.

❸ Differential Equation mode was not available on the original TI-92.

❹ TI-92 graph databases only contain the first four bytes of graph format data. The TI-92 3D mode style flag is not used by the TI-89 / TI-92 Plus, which uses the seventh byte for 3D style information. TI-92 graph databases do not use the 3D Expanded View flag since that was not available on the original TI-92.

14.8. Bitmap PIC Images

Description	Bytes
Data Length of PIC	2
Number of Rows in image	2
Number of Columns in image	2
Bitmap data (8 pixels per byte with the data bits going from most significant to least significant which correspond to left to right pixels on the screen)	variable
PIC_VAR_TAG	1

Table 14.12: Data Object for a PIC

14.9. Tokenized Programs and Functions

Programs and functions are stored similarly and both use the same tag (USER_DEF_TAG). There are two formats depending on whether the program/function is tokenized or in text format. The tokenized format is listed below. There is a routine, **GetFuncPrgmBodyPtr**, that given a pointer to a USER_DEF_TAG returns the pointer to the function or program body — that is it skips all of the parameters and flags.

Description	Bytes
Data Length of program/function	2
END_OF_SEGMENT	1
Tokenized statements	variable
PRGM_TAG or FUNC_BEGIN_TAG ❶	1
END_TAG (terminates parameter list)	1
Parameter List	variable
In-Use Counter	1
Flag 2 (reserved) ❷	1
Flag 1 ❸	1
USER_DEF_TAG	1

Table 14.13: Data Object for a Tokenized Program or Function

❶ This byte is PRGM_TAG for programs and FUNC_BEGIN_TAG for functions.

❷ This flag is reserved for future use, except for Bit 0, the LOCK flag. If the LOCK flag is set, the program will be locked on transmit, or is locked on receive.

❸ This flag byte contains the flags listed below. For tokenized programs/functions the FF_PARSE flag bit will be set to zero.

Description	# of Bits	Value
Graph Style (Least Significant Bits)	3	
Line		0
Dot		1
Thick		2
Animate		3
Path		4
Above		5
Below		6
Square		7
FF_PARSE	1	0/1
FF_ADD_TO_RECENT	1	
FF_ADD_TO_PRIOR	1	
FF_RECENT	1	
FF_PRIOR	1	

Table 14.14: Flag 1 Values

14.10. Programs and Functions in Text Format

Programs and functions stored in text format have a different format than the corresponding tokenized format as listed below.

Description	Bytes
Data Length of program/function	2
Text of program/function	variable
Zero byte	1
Cursor position for editing	2
PRGM_TAG or FUNC_BEGIN_TAG	1
COMMAND_TAG	1
END_TAG	1
Flag 3 (not used)	1
Flag 2 (reserved)	1
Flag 1	1
USER_DEF_TAG	1

Table 14.15: Data Object for a Program or Function Stored in Text

Note that the Flag 3 byte is not used. The Flag 1 and Flag 2 bytes are the same as for tokenized programs/functions (listed above) except that the FF_PARSE bit in Flag 1 is set to one.

14.11. Third Party Data

The Third Party Data object is the format used by the FILE system and may be used by the apps for their own data types.

Description	Bytes
Length	2
Contents (format is application dependent)	variable
Zero byte	1
Identifier (1-4 ASCII characters)	1-4
Zero byte	1
GEN_DATA_TAG	1

Table 14.16: Data Object for Third Party Data

14.12. Assembly Program

Assembly programs are stored in binary format.

Description	Bytes
Length	2
Contents (assembly)	variable
ASM_PRGM_TAG	1

Table 14.17: Data Object for an Assembly Program

15. Expressions and the Expression Stack

This chapter explains the internal data structures used to represent expressions and how the expression stack (estack) is used to do numeric and symbolic operations.

15.1. Overview

The AMS Operating System (OS) evaluates both numeric and symbolic expressions. Expressions are represented in a tagged internal representation called tokenized form. The tokenized form explicitly represents the hierarchical ordering of operations and their operands.

The system provides a tokenizer that uses a lexical scanner and a parser to translate text strings into tokenized form. Tags are used to delimit each element of this form. Numbers, variables, operations, and functions all have associated tag values that identify them. However, some symbols that appear in the text representation do not appear in the tokenized form. For example, delimiters such as commas, parentheses, braces, and brackets are implied by the structure of the tokenized form.

The system also provides a simplifier, which performs evaluation and simplification. The simplifier attempts to reduce an expression to its simplest form. It calls upon a variety of subsystems to perform the operations that are specified by the expression. The work of the tokenizer and simplifier are performed primarily on a stack structure called the expression stack.

Finally, the system provides a detokenizer. As the name implies, the detokenizer translates the tokenized form of an expression into the corresponding text string. The system also provides the means to convert the tokenized form into a pretty printed form that can be displayed.

15.2. Contiguous Tokenized Polish Representation

The tokenizer produces a form called contiguous tokenized Polish representation. In this representation an expression occupies one contiguous block of memory allocated as an array of Quantums. A Quantum is defined in the system by the C declaration:

```
typedef unsigned char Quantum;
```

This representation has two primary advantages — space efficiency and relocatability. No internal pointers are necessary to manipulate or maintain the structure. Since the hierarchical ordering of operations is implicit in the representation, delimiters such as parentheses are not needed to enforce the ordering.

Tokenized Polish form places the operands deepest in the representation and the operator higher or on top of the operands. For example, the simple sum *a + b* would produce the form:

+	(highest address)
b	
a	(lowest address)

This representation is also written

a b +

with the lowest address on the left and highest address on the right. It is important to remember that this form is always interpreted from high address to low address. Evaluation always encounters the operator before its operands. This method is different from reverse Polish form, which encounters the operands before the operator.

Since each operand can also be an expression, any level of complexity can be represented. Here are a few more examples of expressions and their Polish representations. Remember that the tokenizer produces the Polish representation by reading the text expression from left to right, but thereafter, the system interprets the Polish representation from right to left (or high address to low).

Expression	Polish representation
a*b+c	a b * c +
a*(b+c)	a b c + *
a*b+c/d	a b * c d / +
a*(b+c)/d	a b c + * d /
x*y^n-z	x n y ^ * z -

Table 15.1: Examples of Polish Representations

15.2.1. Tags

Tags are single Quantum values that are used in the tokenized form to represent most elements of the structure and also are used to delimit those elements whose representation requires more than a single Quantum. For example, the single letter variables a through z, the symbolic constants π and *e*, the Boolean

constants true and false, and most built-in mathematics functions and operators are represented using a single tag. Floating-point numbers, rational numbers, and integer numbers each require an identifier tag on top of the standard representation of the number. All of the tag values are defined in tiams.h. Each of the tag names ends with the characters "_TAG." The following sections describe the various tags and what they identify or represent.

15.2.2. Numbers

The Operating System includes two separate number systems — the rational number system which contains tagged integers and tagged fractions, and the floating-point number system, which uses Binary Coded Decimal (BCD) floating-point numbers. A primary difference between these number systems is that the rational system is by definition exact and the floating-point system is assumed always to be an approximation.

In the rational system the number of digits is limited but not fixed. If an arithmetic operation on two rational numbers completes successfully, then the result is exact. No loss of precision occurs. In the floating-point system the number of digits is fixed, and therefore, loss of precision is always a possibility. Thus, the result of a floating-point operation is considered to be an approximation.

The rational numbers include tagged integers and tagged fractions. The term tagged integer is used to distinguish these numbers from the C programming types — int, short, long, etc. A tagged integer has three elements — a tag at the highest address, a length, and a magnitude.

An integer magnitude is represented as a sequence of adjacent quantums, with the least significant quantum deepest (lowest address) and the most significant quantum nonzero. For example, the 16 bit integer 65534 (0xFFFE) would appear as 254 255 (0xFE 0xFF) with the least significant quantum deepest.

A sized integer magnitude is a one-quantum length field on top of an integer magnitude. With the quantum size of one byte, the length can be 0 through 255 quantums, and the maximum possible sized integer magnitude is $256^{255} - 1 \approx 10^{614}$. Thus, the sized integer magnitude for the integer 65534 would appear as 254 255 2 (0xFE 0xFF 0x2).

A non-negative integer is represented as a NONNEGATIVE_INTEGER_TAG on top of a sized integer magnitude. Thus, the tagged integer representation of the integer 65534 is 254 255 2 NONNEGATIVE_INTEGER_TAG.

A negative integer is represented as a NEGATIVE_INTEGER_TAG on top of a sized integer magnitude. So, the tagged integer representation of the negative integer -65534 is 254 255 2 NEGATIVE_INTEGER_TAG.

Integer value	Tagged integer representation
-5	5 1 NEGATIVE_INTEGER_TAG
256	0 1 2 NONNEGATIVE_INTEGER_TAG
65538	2 0 1 3 NONNEGATIVE_INTEGER_TAG
-1000000	64 66 15 3 NEGATIVE_INTEGER_TAG

Table 15.2: Tagged Integer Examples

The integer zero is a special case in this representation. Zero has no integer magnitude but is represented simply by a NONNEGATIVE_INTEGER_TAG on top of a zero length field as follows, 0 NONNEGATIVE_INTEGER_TAG. Note that this is the only valid representation of a simple tagged integer zero. The system never generates nor expects a NEGATIVE_INTEGER_TAG on top of a 0 length field nor any tagged integer with a nonzero length field and a zero magnitude. These invalid representations will cause unexpected system behavior.

Fractions include two sized integer magnitudes — one for the numerator and one for the denominator. A positive fraction is identified by a POSITIVE_FRACTION_TAG. A negative fraction is identified by a NEGATIVE_FRACTION_TAG. The denominator is placed deepest in the representation, then the numerator, then the tag on top. Fractions are always fully reduced, that is, the greatest common divisor of the numerator and denominator is 1.

Fraction value	Tagged fraction representation
1/2	2 1 1 1 POSTIVE_FRACTION_TAG
-2/3	3 1 2 1 NEGATIVE_FRACTION_TAG
5/256	0 1 2 5 1 POSITIVE_FRACTION_TAG
-999999/1000000	64 66 15 3 63 66 15 3 NEGATIVE_FRACTION_TAG

Table 15.3: Tagged Fraction Examples

The fraction representation includes two special cases. They are called signed zeros. Signed zeros occur when the system performs symbolic operations such as computing limits or simplifying expressions involving infinity. They are represented by a fraction whose numerator is 0 and whose denominator is 1. Thus, +0 is 1 1 0 POSITIVE_FRACTION_TAG, and -0 is 1 1 0 NEGATIVE_FRACTION_TAG. These are the only valid fractions with a zero numerator, and the denominator must be 1. The system does not generate nor expect any other fraction whose numerator or denominator is zero. Invalid fractions will cause unexpected behavior.

The representation of Binary Coded Decimal (BCD) floating-point numbers is described in detail in the TI-89 / TI-92 Plus Sierra C™ Reference Manual, chapter **2. Compiler**, section **2.9.4 Floating-Point Representations**. In simplest terms they consist of a two byte quantity which represents the algebraic sign of the number and the exponent or power of 10 by which the mantissa is multiplied. This quantity is followed by a mantissa, which represents the fixed number of significant digits in the number. Each nibble or hexadecimal digit of each byte represents a single decimal digit.

Tagged floating-point numbers are represented by a FLOAT_TAG on top of a 14 digit floating-point number. The float number is placed as it would normally appear in memory with the sign/exponent at the lowest address, and then, the mantissa with the most significant digits at the lower address and the least significant digits at the higher address. For example, the tagged floating-point representation for the float approximation of π is:

0x40 0x00 0x31 0x41 0x59 0x26 0x53 0x58 0x98 FLOAT_TAG.

15.2.3. Variables, Units and Physical Constants

Variables are represented in two ways. Since single alphabetic characters (a–z) are most often used to represent variables, each of them is identified by a unique tag value. Thus, the variable a is represented by A_VAR_TAG, the variable b is represented by B_VAR_TAG, and so on, through the variable z represented by Z_VAR_TAG.

Multicharacter variable names and all single nonalphabetic character names are identified by a VAR_TAG at both ends of the sequence of characters. The name characters are placed between the VAR_TAG's with the first character deepest. The Operating System uses an extended ASCII character set described in Table 4.2: Character Set. Valid name characters are specified in the TI-89 / TI-92 Plus Guidebook. Names are case insensitive, so x and X are both tokenized as X_VAR_TAG, and abc, Abc, AbC, and so on, are all tokenized as VAR_TAG a b c VAR_TAG.

Variable Name	Representation
x	X_VAR_TAG
baz	VAR_TAG b a z VAR_TAG
θ	VAR_TAG θ VAR_TAG
Theta	VAR_TAG t h e t a VAR_TAG
å_295	VAR_TAG å _ 2 9 5 VAR_TAG

Table 15.4: Variable Name Examples

The Operating System has system variables that are reserved for special purposes. These reserved variables are used in a variety of ways in graphing, plotting, table generation, and statistical computations. They are numbered and are represented as a SYSVAR_TAG on top of the corresponding system variable number. For example, the function graphing variable xmin is represented by SV_XMIN SYSVAR_TAG, and the statistics variable Σx is represented by SV_SIGMA_X SYSVAR_TAG.

Two special naming conventions are associated with the underscore character '_'. Variable names that end with an underscore are assumed to be complex variables. Thus, the variable z is assumed to be real, but the variable z_ is assumed to be complex. Variable names that begin with an underscore are assumed to be unit names or the names of physical constants, which include a unit expression. For example, the unit meter is named _m, and the unit kilogram is named _kg. The physical constant for the speed of light is named _c and evaluates to the unit expression 299792458.0 _m / _s.

15.2.4. Other Constants

Arbitrary real constants @1, @2, . . . , and arbitrary integer constants @n1, @n2, . . . behave somewhat like variables and somewhat like constants. The number following the @ symbol (1, 2, etc.) is referred to as the "suffix." The representation uses ARB_REAL_TAG or ARB_INT_TAG on top of one quantum containing the identifying suffix. Thus, @25 is 25 ARB_REAL_TAG, and @n10 is 10 ARB_INT_TAG.

The system also implements the following symbolic constants.

Constant Type	Value	Tag
Boolean	TRUE	TRUE_TAG
Boolean	FALSE	FALSE_TAG
Finite	π	PI_TAG
Finite	*e* (base of the natural ln)	E_TAG
Finite	i (√(-1))	I_TAG
Transfinite	-∞	MINUS_INFINITY_TAG
Transfinite	∞	PLUS_INFINITY_TAG
Transfinite	±∞	PLUS_OR_MINUS_INFINITY_TAG
Transfinite	0/0 (any real value)	UNDEFINED_TAG

Table 15.5: Symbolic Constants

Strings are represented as a STR_DATA_TAG on top of the string data, which is delimited by a null character (0) on both ends. Thus, the string “hello” is represented as 0 h e l l o 0 STR_DATA_TAG. Since zero is also the value of VAR_TAG, this representation looks much like a variable name. However, variable names have a maximum length of eight characters and can contain only valid name characters. Strings are not limited in length and can contain any character except a null (0).

15.2.5. One-argument Tags

Tokenized Polish representation makes no distinction between functional expressions and operator expressions. Both are represented as an identifying tag value on top of its argument(s). Many built-in functions require exactly one argument, for example, sin(x), ln(x), abs(x), etc. A few of the operators also operate on exactly one operand, for example, -x, x!, x%, etc.

Expression	Representation
-x	X_VAR_TAG CHS_TAG (change sign)
n!	N_VAR_TAG FACTORIAL_TAG
20%	20 1 NONNEGATIVE_INTEGER_TAG PERCENT_TAG
sin(x)	X_VAR_TAG SIN_TAG
ln(abc)	VAR_TAG a b c VAR_TAG LN_TAG
abs(ln(x))	X_VAR_TAG LN_TAG ABS_TAG

Table 15.6: Examples of Single Argument Functions and Operators

15.2.6. Two-argument Tags

Many built-in functions require exactly two arguments, for example, zeros(ln(x),x), mod(a, b), nCr(m, n). Functions of two arguments are represented as the corresponding function tag on top of the first argument on top of the second argument.

Expression	Representation
zeros(ln(x),x)	X_VAR_TAG X_VAR_TAG LN_TAG ZEROS_TAG
mod(a, b)	B_VAR_TAG A_VAR_TAG MOD_TAG
nCr(m, n)	N_VAR_TAG M_VAR_TAG COMB_TAG (combinations)

Table 15.7: Examples of Functions of Two Arguments

Many built-in operators also require exactly two operands, for example, arithmetic operators, power operators, relational operators, logical operators, the store operator and the with operator. The arithmetic operators +, −, *, /, .+, .−, .*, ./, and the store operator →, all place the first operand deepest, then the second operand, and finally the corresponding tag on top.

Expression	Representation
a + b	A_VAR_TAG B_VAR_TAG ADD_TAG
x .* y	X_VAR_TAG Y_VAR_TAG DOT_MULT_TAG
π → z	PI_TAG Z_VAR_TAG STORE_TAG

Table 15.8: Examples of Arithmetic Operations and the Store Operation

The remaining binary operators, the power operators ^ and .^, the relational operators =, /=, <, <=, >, and >=, the logical operators and, or, and xor, and the with operator |, all place the tag on top of the first operand on top of the second operand, just as the functions do.

Expression	Representation
x ^ y	Y_VAR_TAG X_VAR_TAG EXPONENTIATION_TAG
r > s	S_VAR_TAG R_VAR_TAG GT_TAG
a or b	B_VAR_TAG A_VAR_TAG OR_TAG
c | d	D_VAR_TAG C_VAR_TAG SUCH_THAT_TAG (with)

Table 15.9: Examples of Other Binary Operations

15.2.7. Tags That Take More Than Two or a Variable Number of Arguments

The tokenized Polish representation of functions that take more than two arguments is the function tag on top of a tail of arguments. A tail is a sequence of expressions on top of an END_TAG. The first argument is at the top of the sequence just below the function tag. The last argument is deepest in the sequence just above the END_TAG. Thus, Σ(m, m, 1, n) is represented as END_TAG N_VAR_TAG 1 1 NONNEGATIVE_INTEGER_TAG M_VAR_TAG M_VAR_TAG SUMMATION_TAG.

A tail is also used for functions that accept a variable number of arguments. For example, the ∫ function will accept 2, 3, or 4 arguments. Therefore, ∫(ln(x), x) is represented by END_TAG X_VAR_TAG X_VAR_TAG LN_TAG INTEGRAL_TAG. ∫(sin(x), x, 0, π) is represented by END_TAG PI_TAG 0 NONNEGATIVE_INTEGER_TAG X_VAR_TAG X_VAR_TAG SIN_TAG INTEGRAL_TAG.

An empty tail is represented by simply an END_TAG. Thus, rand( ) is represented by END_TAG RAND_TAG.

15.2.8. Lists and Matrices

Most of the functions and operators in the system will operate on lists of expressions and matrices containing expressions. A list is represented as a LIST_TAG on top of a tail of expressions. Thus, the empty list { } is represented as END_TAG LIST_TAG. The list {a, π, tan(x)} is represented by END_TAG X_VAR_TAG TAN_TAG PI_TAG A_VAR_TAG LIST_TAG. None of the elements of a list can be a list unless they all are equal length lists thus forming a matrix.

A matrix is represented as a list of lists. Each of the inner lists represents a row of the matrix. For example, the matrix [a, b; c, d] is represented by END_TAG END_TAG D_VAR_TAG C_VAR_TAG LIST_TAG END_TAG B_VAR_TAG A_VAR_TAG LIST_TAG LIST_TAG. The lengths of the rows must be equal. None of the elements of a row can be a list. The system neither generates nor expects invalid list or matrix structures. They will cause unexpected system behavior.

15.2.9. Primary, Secondary, and Command Tags

The tags discussed in the preceding sections are called primary tags. In each case the single tag on top of the representation is all that is required to identify that element. However, the Operating System provides more than 256 built-in functions, operators, commands, programming constructs, and so on. Therefore, some of the primary tags are used with additional one Quantum tag values to provide additional identifiers.

The tag value SECONDARY_TAG is used with secondary tag values to provide representation for additional functions and operators. For example, ORD_TAG happens to have the same Quantum value as V_VAR_TAG, but a SECONDARY_TAG on top of that Quantum value represents the ord() function rather than the variable v.

Expression	Representation
getKey( )	END_TAG GETKEY_TAG SECONDARY_TAG
#s	S_VAR_TAG INDIRECTION_TAG SECONDARY_TAG
v ▸ POL	V_VAR_TAG TO_POLAR_TAG SECONDARY_TAG

Table 15.10: Secondary Tag Examples

Similarly, the primary tag value COMMAND_TAG is used on top of command tag values to provide representation for elements of the TI-BASIC programming language.

Expression	Representation
ClrHome	CLRHOME_TAG COMMAND_TAG
DelVar x	X_VAR_TAG DELVAR_TAG COMMAND_TAG
Disp a, b	END_TAG B_VAR_TAG A_VAR_TAG DISP_TAG COMMAND_TAG

Table 15.11: Command Tag Examples

15.2.10. User and Application Defined Functions and Programs

Both calculator users and application developers can provide new functions and programs. References to them are all represented in the same way. The topmost identifying tag is USER_FUN_TAG. Next comes a variable name representation that specifies the name of the function or program. Finally comes a tail of arguments. Thus, if a user or application defines a function f(x), then the function reference f(0) is represented by END_TAG 0 NONNEGATIVE_INTEGER_TAG F_VAR_TAG USER_FUN_TAG. If a user or application defines a program pa(x, y, z), then the program reference pa(c, “dd”, -1) is represented by END_TAG 1 1 NEGATIVE_INTEGER_TAG 0 d d 0 STR_DATA_TAG C_VAR_TAG VAR_TAG p a VAR_TAG USER_FUN_TAG.

15.3. External Versus Internal Tokenized Polish

The system actually uses two slightly different tokenized forms. The tokenizer produces a form called external tokenized form, which has been described in the previous sections. This form has individual tags for representing all of the operators, functions, and commands provided by the system. This form allows for all the different ways that expressions may be represented including multiple representations of the same expression. For example, a / b and a * (b ^ -1) are different representations of the same expression.

The simplifier produces a form called internal tokenized form. The two primary reasons for this second form are expression recognition and efficiency of implementation. The simplifier must be able to recognize when expressions are similar or the same. Thus, expressions are translated into a standard form using fewer tags, allowing the simplifier to more easily recognize expressions that combine or cancel. Fewer tags and a standardized form allow the implementation of the simplifier to be smaller and faster.

The following tags only occur in external tokenized form. Simplification replaces these tags with a standard internal form.

- CHS_TAG (change sign or negation) is replaced by a multiplication by minus one. Thus, -x is replaced by -1 * x.
- SUBTRACT_TAG is replaced by addition of a negative operand. Thus, x − 2 is replaced by x + (-2). The expression x − y is replaced by x + (-1 ∗ y).
- DIVIDE_TAG is replaced by multiplication by the denominator raised to the minus one power. Thus, x / y is replaced by x * (y ^ -1).
- E_TAG, which represents the base *e* of the natural logarithms, is replaced by the exponential function exp( ) represented by EXP_TAG. EXP_TAG is an internal only tag and never occurs in the external tokenized form. Thus, *e*^x is replaced by exp(x). When the symbol *e* occurs other than as a base for exponentiation, it is replaced by exp(1). Thus, *e* + x is replaced by exp(1) + x.
- Hyperbolic function tags (SINH_TAG, COSH_TAG, TANH_TAG) are replaced by the equivalent exponential expressions.
  Sinh(x) is replaced by exp(x) * (1 / 2) + exp(x) ^ (-1) * (-1 / 2).
  Cosh(x) is replaced by exp(x) * (1 / 2) + exp(x) ^ (-1) * (1 / 2).
  Tanh(x) is replaced by ((exp(x)) ^ 2 + 1) ^ (-1) * ((exp(x)) ^ 2 + (-1))
- LOG_TAG, which represents the base-ten logarithm function log( ), is replaced by the equivalent natural logarithm expression. Thus, log(x) is replaced by ln(x) ∗ (ln(10)^(-1)).
- SIN_TAG, COS_TAG, and TAN_TAG are replaced by equivalent expressions using a two-argument tag called SIN2_TAG. SIN2_TAG is an internal only tag that represents the function sin2(x, k) defined as sin(x + (k ∗ π / 2)). Since cos( ) can be represented as a shifted sin( ), and since tan( ) can be represented as a ratio of sin( ) and cos( ), sin2( ) is used to represent them all. The representation is SIN2_TAG on top of the representation of the first argument x on top of the representation of the shift argument k. Thus, sin(x) becomes sin2(x, 0); cos(x) becomes sin2(x, 1); tan(x) becomes sin2(x, 0) * sin2(x, 1)^(-1).
- I_TAG, which represents the imaginary number (√(-1)), is replaced by an equivalent expression using a two-argument tag called IM_RE_TAG. IM_RE_TAG is an internal only tag whose two arguments are the real and imaginary parts of an expression. The real and imaginary parts must be real values. Thus, the expression x + i ∗ y tokenizes as X_VAR_TAG I_TAG Y_VAR_TAG MULTIPLY_TAG ADD_TAG. Since the system assumes that the variables x and y are real, the simplifier replaces this external form with the internal form X_VAR_TAG Y_VAR_TAG IM_RE_TAG.

- XOR_TAG, which represents the exclusive-or operator xor, is replaced by an equivalent expression using the and operator and the or operator. Thus, a xor b is replaced by not a and b or a and not b.
- The tags representing transformation functions, such as factor( ) (FACTOR_TAG), expand( ) (EXPAND_TAG), and so on, never appear in the internal tokenized form.

Calculator users prefer to see results in a more standard form, for example, x − y rather than x + (-1 ∗ y) and a / b rather than a * (b^(-1)). Therefore, the system provides a routine called **replace_top_with_post_simplified** to transform internal tokenized form to external tokenized form. CHS_TAG, SUBTRACT_TAG, DIVIDE_TAG, E_TAG, SINH_TAG, COSH_TAG, TANH_TAG, SIN_TAG, COS_TAG, TAN_TAG, and I_TAG are restored where they make the result more readable.

System routines typically accept only one tokenized form as input and produce only one tokenized form as output. Applications must not pass external tokenized form to a routine that expects internal tokenized form and must not pass internal tokenized form to a routine that expects external tokenized form. The external only tags listed above may cause an internal only routine to throw errors or may cause unexpected behavior. Similarly, internal only tags such as EXP_TAG, SIN2_TAG, and IM_RE_TAG will cause an external only routine to throw errors or behave unexpectedly. Appendix A: System Routines describes many entry points that operate on tokenized expressions. Each of the entry point descriptions specifies the acceptable input form and the output form that is returned.

15.4. Most Main Ordering and Internal Representations of Exponentiation, Multiplication, and Addition

Another important aspect of internal tokenized form is ordering. When the elements of an expression can be reordered, the simplifier does so using most main ordering. Some of the aspects of most main ordering are:

- Single alphabetic variables are ordered r > s > t . . . > x > y > z > a > b . . . > p > q.
- Single alphabetic variables are more main than other variables. Thus, x is more main than y, but y is more main than xx.
- Single nonalphabetic variables and multicharacter variables are ordered by ASCII sequence. Thus, z is more main than a, but aa is more main than zz.
- Variables are more main than symbolic constants such as π.
- Symbolic constants are more main than numbers.

See the description of the system routine **compare_expressions** for more information on most main ordering.

The internal representation of exponentiation depends primarily on the type of exponent. If the exponent is a number, then the representation is an EXPONENTIATION_TAG on top of the internal representation of the base on top of the internal representation of the exponent. Thus, x ^ 2 is simplified to 2 1 NONNEGATIVE_INTEGER_TAG X_VAR_TAG EXPONENTIATION_TAG. The one important exception occurs when the base is *e*. The symbol *e* raised to any exponent, numeric or otherwise, is simplified to EXP_TAG on top of the exponent. Thus, *e* ^ 2 simplifies to 2 1 NONNEGATIVE_INTEGER_TAG EXP_TAG, and *e* ^ x simplifies to X_VAR_TAG EXP_TAG.

The internal representation of any other base raised to any non-numeric exponent uses the exponential (EXP_TAG) and natural logarithm (LN_TAG) functions. For example, x ^ y is represented as exp(y * ln(x)) which is Y_VAR_TAG X_VAR_TAG LN_TAG MULTIPLY_TAG EXP_TAG. The only exception is that 0 ^ u is represented internally as an EXPONENTIATION_TAG on top of a zero on top of the internal representation of expression u.

Since multiplication can be reordered, the simplifier orders products with the most main factor highest and the least main factor lowest. Thus, x * y is externally tokenized as X_VAR_TAG Y_VAR_TAG MULTIPLY_TAG. Then, the simplifier reorders this to Y_VAR_TAG X_VAR_TAG MULTIPLY_TAG, because x is more main than y.

In internal form division is represented as a product with the denominator raised to the minus one power. Thus, x / y is externally tokenized as X_VAR_TAG Y_VAR_TAG DIVIDE_TAG. Then, the simplifier changes this to a product and reorders it as 1 1 NEGATIVE_INTEGER_TAG Y_VAR_TAG EXPONENTIATION_TAG X_VAR_TAG MULTIPLY_TAG.

Another important aspect of the internal representation of products is that the first (most main) operand of a product is never a product. Thus, (a * b) * (c * d) is externally tokenized as A_VAR_TAG B_VAR_TAG MULTIPLY_TAG C_VAR_TAG D_VAR_TAG MULTIPLY_TAG MULTIPLY_TAG due to the parentheses used in the text. In this external form each of the operands of the topmost MULTIPLY_TAG is also a product. The simplifier reorders this expression so that the topmost operand of each MULTIPLY_TAG is not a product. The result is D_VAR_TAG C_VAR_TAG MULTIPLY_TAG B_VAR_TAG MULTIPLY_TAG A_VAR_TAG MULTIPLY_TAG.

The simplifier performs a similar process with addition. Since addition can be reordered, the simplifier reorders sums with the most main term highest and the least main term lowest. Subtraction is changed to a sum with the second operand negated. Finally, the simplifier makes sure that the topmost operand of each ADD_TAG is not a sum. Thus, a + b becomes B_VAR_TAG A_VAR_TAG ADD_TAG. The expression a − b becomes 1 1 NEGATIVE_INTEGER_TAG

B_VAR_TAG MULTIPLY_TAG A_VAR_TAG ADD_TAG. The expression (a + b) + (c + d) becomes D_VAR_TAG C_VAR_TAG ADD_TAG B_VAR_TAG ADD_TAG A_VAR_TAG ADD_TAG.

15.5. The Expression Stack

The simplification of an expression usually requires intermediate operations, such as the replacement of variables with their assigned values, or the computation of partial results that are combined to form the final result. The Operating System uses a generalized stack called an expression stack (estack) to perform these operations. The tokenizer also produces the external tokenized form on the estack (expression at the highest address).

The system allocates the estack in a fixed location as an array of Quantums. The bottom of the stack is at the lowest address, and the stack grows toward higher addresses. This stack is described as generalized because the system allows a variety of operations on any expression on the stack, not just the top expression.

References to expressions on the estack are via pointers defined in the system by the C declaration:

```
typedef Quantum * EStackIndex;
```

This pointer type is used to point to tokenized expressions whether they are on the estack or elsewhere in memory.

The system also defines a macro for accessing expressions via estack pointers. The C declaration is:

```
#define ESTACK(i) (*(i))
```

The bottom of the estack is delimited by a global EStackIndex called **bottom_estack**. This pointer does not change and always points to an END_OF_SEGMENT_TAG to denote the end of the stack. The topmost occupied Quantum of the estack is accessed by a global EStackIndex called **top_estack**.

The system provides a variety of routines that perform operations on the estack. Routines whose names begin with “push_” push something on the estack. For example, **push_parse_text** pushes the external tokenized form of a text expression onto the estack; **push_quantum** pushes a single Quantum value onto the estack; and **push_between** pushes the data that resides between two pointers onto the estack.

Routines whose names begin with “replace_” replace one or more expressions that are on top of the stack with a new expression. For example, **replace_top2_with_sum** replaces the top two expressions on the estack with the sum of those two expressions; **replace_top_with_reciprocal** replaces the

top expression with its reciprocal; and **replace_top_with_post_simplified** replaces the top expression (assumed to be internal form) with its external tokenized form.

Routines whose names begin with “is_” are used to get information about expressions. They have no effect on the contents of the estack or the expressions they inspect. However, sometimes they must perform some temporary computation to determine the requested information. Under these circumstances the estack may temporarily grow. For example, **is_negative** tests whether an expression is negative; **is_real** tests whether an expression is real; and, **is_equivalent_to** tests whether one expression is equivalent to another.

Routines whose names begin with “index_” or end with “_index” are used to locate expressions or subexpressions. They also have no effect on the estack or expressions. They simply return the EStackIndex of the located expression. For example, **next_expression_index** returns the index of the next expression below the expression pointed to by its input argument; **lead_factor_index** returns the index of the first factor of the multiplication pointed to by its input argument; and, **remaining_factors_index** returns the index of the remaining factors following the first factor of the multiplication pointed to by its input argument.

See **Appendix A: System Routines** for a complete list of the system routines that perform estack operations.

15.6. An Example of Working on the EStack

This section takes a simple C language programming example and works through alternative implementations to show how the same operations can be done using estack operations. We begin with a C language implementation of a function to compute the future value of a lump sum present value given the periodic interest rate and the number of periods. The formula for this computation is future_value = present_value * (interest_rate + 1) ^ number_of_periods.

C programming language example:

```
/* This function takes three BCD16 arguments.
   pv = present value
   ir = interest rate
   np = number of periods
   The function returns future value fv as a BCD16.
*/
BCD16 fv (BCD16 pv, BCD16 ir, BCD16 np)
{  return pv * pow(ir + 1.0, np);
}
```

15.6.1. Estack Arguments and Results

Lets modify the future value function to accept its arguments as a tail on the estack and return its result on the estack. This example continues to accept and return only floating-point values. The changes use the following features.

- An EstackIndex called arg is used to access each of the arguments in turn.
- BCD16 variables fv, pv, ir, and np are declared to receive the argument values and perform the computation.
- The system function **next_expression_index** is used to step from each argument to the next.
- The system function **estack_to_float** is used to copy the tagged floats from the estack into the BCD16 variables.
- The system function **push_Float** is used to push the BCD16 result onto the estack as a tagged float.

```
/* This function takes three tagged BCD16 arguments.
   The arguments are required to be in a tail on top
      of the expression stack in the following order.
   pv = present value
   ir = interest rate
   np = number of periods
   The function returns future value fv as a tagged
      float on top of the expression stack.
*/
void fv (void)
{  EStackIndex arg;         /* argument pointer */
   BCD16 fv, pv, ir, np;    /* BCD16 variables */
   /* point arg to the first argument in the tail */
   arg = top_estack;
   /* get the present value argument */
   pv = estack_to_float (arg);
   /* advance the argument pointer to the next argument */
   arg = next_expression_index (arg);
   /* get the interest rate argument */
   ir = estack_to_float (arg);
   /* advance the argument pointer to the next argument */
   arg = next_expression_index (arg);
   /* get the number of periods argument */
   np = estack_to_float (arg);
   /* perform the future value calculation */
   fv = pv * pow (ir + 1.0, np);
   /* push the future value on the estack */
   push_Float (fv);
}
```

15.6.2. Estack Calculations

Now lets modify the example to perform the calculation on the estack rather than in BCD16 variables. This extension will be necessary if the function must handle arguments other than floating-point numbers. If the arguments to a function can be floats, rationals, symbolic constants, variables, expressions, or lists of any of these, then the computations are best done on the estack. The changes use the following features.

- EStackIndexes are declared to point to the arguments and temporary results.
- The system function **push_arg_plus_1** is used to add one to an argument.
- The system function **push_exponentiate** is used to raise a value to a power.
- The system function **push_product** is used to multiply two values.
- The system function **delete_between** is used to delete temporary results.

```
void fv (void)
{  EStackIndex pv, ir, np, tmp;   /* argument pointers */
   /* point to the present value argument */
   pv = top_estack;
   /* point to the interest rate argument */
   ir = next_expression_index (pv);
   /* point to the number of periods argument */
   np = next_expression_index (ir);
   /* perform the future value calculation */
   /* add 1 to the interest rate */
   push_arg_plus_1 (ir);
   /* point to the temporary result */
   tmp = top_estack;
   /* raise (ir + 1) to the np power */
   push_exponentiate (tmp, np);
   /* point to the temporary result */
   tmp = top_estack;
   /* multiply by the present value */
   push_product (tmp, pv);
   /* now the future value is on top of the estack */
   /* delete intermediate results */
   delete_between (pv, tmp);
}
```

This version of the example is longer and more complicated than either of the previous versions. Thus, an obvious question is “what has been gained?” The answer is a great deal of power and flexibility. This latest version does not care about the types of the arguments. If all of the arguments are rational numbers, the result will be a rational number. If the arguments are symbolic, the result will be symbolic. If the arguments are of valid but differing types, they will be combined in an appropriate way. If the arguments are not valid for the specified calculation, an appropriate error will be reported.

15.7. Working With Lists

This section describes some of the routine ways of working with lists and tails. A list is represented as a LIST_TAG on top of a tail. A tail is a sequence of expressions on top of an END_TAG. For example, the list {a, 1, ln(x)} takes the tokenized form

END_TAG X_VAR_TAG LN_TAG 1 1 NONNEGATIVE_INTEGER_TAG A_VAR_TAG LIST_TAG

where the END_TAG is at the lowest address and the LIST_TAG is at the highest address.

The system routines that implement calculator functions understand and correctly handle lists. For example, **push_ln** automatically computes the natural logarithm of each element of a list. **push_sum** automatically adds two lists, element by element, and throws an appropriate error if the lists do not have the same number of elements. **push_negate** changes the sign of each element of a list, and so on. Thus, depending upon the operations involved, it is often possible to write code that does not need to check whether its input arguments are lists. The last future value function of the previous section is an example. Since each of the called system routines understands lists, the resulting future value function correctly handles lists.

Sometimes new code must be written to perform some new process on lists. These new processes generally fall into two categories based on their result. Either they create a new version of the list or they do not. The functions mentioned in the previous paragraph create new lists from input lists. Here are examples that do not create new lists. **is_constant** determines whether every

element of the list is a constant value and returns a Boolean result. **push_sumlist** returns an expression that represents the sum of all the elements of the list. **push_dimension** returns the number of elements in the list.

Functions that do not create new lists generally use a loop to walk through the elements of the list. Here is a function that returns the number of elements in a list.

```
unsigned short number_of_elements (EStackIndex i)
{  unsigned short count = 0;   /* initialize counter */
   --i;   /* move index from LIST_TAG to first list element */
   while (END_TAG != ESTACK(i))   /* while not at end of list */
   {  ++count;   /* increment counter */
      i = next_expression_index (i);   /* step to next element */
   }
   return count;
}
```

This function illustrates three key elements of a process that loops over the elements of a list.

- Decrement the index to move it from the LIST_TAG to the first list element.
- Test for END_TAG at the current location to determine when to stop.
- Use **next_expression_index** to advance the index through the elements of the tail.

Here is another example that illustrates this pattern — a possible implementation of **push_sumlist**.

```
void push_sumlist (EStackIndex i)
{  push0 ();   /* push a zero on the estack */
   --i;      /* move index to first element of list */
   while (END_TAG != ESTACK(i))   /* while not at end of list */
   {  add_to_top (i);   /* add current element to sum */
      i = next_expression_index (i);   /* step to next element */
   }
}
```

Note that the three key elements are identical. The differences from the previous example are in the initialization (**push0**), the operation (**add_to_top**), and the completion (return value on the estack).

Looping is less applicable to procedures that create a new copy of a list. No two elements of a list are necessarily the same size. A computed result is not necessarily the same size as the corresponding input value. Therefore, overwriting each element of a list with a newly computed element is not a reasonable approach. Also looping operates on the list from the first element (highest on the stack) down to the last element (lowest on the stack). If the operation of the loop is to push a computed value based on each element, the resulting new list is in reverse order. Another loop can be added simply to reverse the order of the elements. However, this approach requires the additional stack space to make another copy of the list and requires the additional time to make the correctly ordered copy, and finally, delete the incorrectly ordered copy.

An alternative implementation for routines that make new copies of lists is to use recursion. The following example represents a pattern that occurs frequently. The key elements are:

- A main routine that calls a subroutine to operate on the tail of the list and then pushes a LIST_TAG on top of the resulting tail to form the new list.
- A subroutine that recurs down to the END_TAG of the tail doing nothing on the way and then pushes each newly computed value on the way out of the recursion giving the resulting list in correct order.

Here is a pair of functions that combine to compute the square root of each element of a list.

```
void push_sqrt_list (EStackIndex i)
{   /* i indexes a list */
    push_sqrt_tail (i – 1u); /* compute the sqrt of the tail */
    push_quantum (LIST_TAG); /* push a LIST_TAG on top */
}

void push_sqrt_tail (EStackIndex i)
{   /* i indexes a tail.
       Pushes a tail of the square roots of the elements. */
    if (END_TAG == ESTACK (i)) /* if at the bottom of the tail */
        push_quantum (END_TAG); /* push bottom of new tail */
    else
    {   /* recur to next element of tail */
        push_sqrt_tail (next_expression_index (i));
        /* on the way out, compute sqrt of each element */
        push_sqrt (i);
    }
}
```

The recursive alternative has the advantage of automatically creating the new list in the correct order. The disadvantage is that recursion consumes more hardware stack for the recursive stack frames. This approach makes a recursive subroutine call, thereby using a stack frame for each element of the list, and finally, the END_TAG that terminates the list.

The recursive pattern for computing a list result from a list input is so common that the system includes a generalized procedure that provides the recursion. The **map_tail** routine makes it unnecessary to write the recursive subroutine as shown in the previous example. The first argument in **map_tail** is a pointer to a function that pushes a single result value based on a single input value. Its second argument is a tail of input values. It performs the recursion, applying the specified function to each element of the tail. Thus, push_sqrt_list can be implemented as follows, making push_sqrt_tail unnecessary.

```
void push_sqrt_list (EStackIndex i)
{  /* apply sqrt function to the tail */
   map_tail (push_sqrt, i – 1u);
   /* push the LIST_TAG on top of the tail */
   push_quantum (LIST_TAG);
}
```

16. Working with Numbers

16.1. Overview

This chapter describes the two separate number subsystems that are built into the AMS Operating System — the rational system and the float system. The numeric representations used by these systems are described in section **15.2.2. Numbers**. Briefly, the rational system is an exact number system that uses tagged integers and tagged fractions. The float system is an approximation number system that uses BCD floating point numbers.

16.2. Rational System vs. Float System

The primary advantage of the rational system is no loss of precision. So long as no operation overflows or underflows, rational results are exact. The primary disadvantage of the rational system is that the representation size is not fixed. As tagged integers increase in magnitude the size of the representation increases accordingly. As the magnitudes of numerators and denominators increase, the representation size of fractions grows as well. Indexes into arrays of rational numbers cannot be directly computed. To reach a specific array element, the code must “step over” each of the preceding elements. Thus, depending upon the type of operations, the rational system can be slower than the float system.

The primary advantage of the float system is the fixed size of the presentation. As a result the speed of operations is more predictable, and indexes into arrays of float numbers can be directly computed. The primary disadvantage of the float system is loss of precision. Since the representation size is fixed, float results must be rounded or truncated to a fixed number of significant digits after each operation. Thus, a float result is always assumed to be an approximation.

Loss of precision makes the float system less suitable than the rational system for computer algebra, where many of the most powerful results depend upon the ability to maintain exact intermediate results. The rational system is less suitable when fast approximate results are desired, such as during graphing. Since the TI‑89 / TI‑92 Plus calculators need both these capabilities, the Operating System includes both types of numbers.

A natural question for any application is whether to focus on or force the use of only one number system. For the most part the attitude of the Operating System is “let the calculator user decide.” The system provides a mode setting, described in the next section, that allows the calculator user to control this issue. There are exceptions. For example, the graphing application and the statistical calculations require the use of the floating-point system, and so, ignore the current mode setting.

16.3. EXACT/APPROX/AUTO Modes

The EXACT/APPROX mode setting controls the way numbers are treated by the computer algebra system. In simplest terms EXACT mode causes the simplifier to convert float numbers to integers or fractions. APPROX mode causes the simplifier to convert integers and fractions to float numbers. In AUTO mode the simplifier does not alter the number types unless an operation must combine a float number with a nonfloat number. When this combination occurs, the nonfloat number is converted to a float number before they are combined.

Number conversions due to mode are performed by **push_internal_simplify** when it encounters each number. Lower level routines in the computer algebra assume that the enforcement of the mode setting has occurred before they are called. Thus, if an application calls **push_internal_simplify**, **push_simplify**, or **push_simplify_statements** to evaluate an expression, the numbers in the expression will be handled according to the mode setting. However, if an application directly calls lower level computer algebra routines with numeric arguments, the mode setting will not be enforced.

For example, if the mode setting is EXACT, and an application passes the expression 1.5 + 2 to **push_internal_simplify**, the result will be 7/2. The float value 1.5 is automatically converted to 3/2. However, if the application passes 1.5 and 2 to the **push_sum** routine, the result will be 3.5. **push_sum** does not enforce the mode setting, and the default action for combining floats and nonfloats is to convert the nonfloat into a float.

To duplicate the computer algebra's numeric behavior, an application has two choices.

- Always enter the computer algebra simplifier through one of the three main entry points (**push_internal_simplify, push_simplify, push_simplify_statements)**. Thus, the mode setting will be enforced by **push_internal_simplify**.
- Take responsibility for checking the mode setting and when necessary, applying the appropriate conversions to its numeric arguments before calling lower level routines.

Finally, an application may decide to ignore the current mode setting and enforce one of its own choosing. An example is the built-in graphing application. The grapher saves the current mode setting, changes the mode setting to APPROX while it is active, and restores the current mode setting when it finishes.

The current status of the EXACT/APPROX mode setting is maintained in the global variable **NG_control.** The macros IS_ARITH_EXACT, IS_ARITH_APPROX, and IS_ARITH_AUTO are used to test the status of the mode. The macros SET_ARITH_EXACT, SET_ARITH_APPROX, and SET_ARITH_AUTO are used to alter the mode setting.

Here is a coding example from a system routine which must temporarily alter this mode setting. Numeric integration (**push_nint)** requires that all evaluation be done with float numbers. So, this routine saves the mode, changes it, and restores it before returning.

```
void push_nint (EStackIndex i, EStackIndex vi, EStackIndex j,
        EStackIndex k)
{
   Access_AMS_Global_Variables;
   CONTROL_BITS old_NG_control = NG_control;
   SET_ARITH_APPROX;
   .
   .
   .
   /* apply the quadrature algorithm */
   .
   .
   .
   NG_control = old_NG_control;
}
```

16.4. Floating Point Numbers

Applications can work with float numbers on the estack or in C floating-point variables. The compiler supports two forms of floating-point values as described in Chapter 2 of the compiler documentation. The calculator implementation uses the standard C type double. The symbols BCD16 and Float are also defined to be double. BCD16 is the recommended type for declaring floating-point variables in applications.

This type uses a 16-digit mantissa and provides more accuracy. Thus, BCD16 variables provide the best results when implementing iterative algorithms that require a great deal of floating-point computation.

push_Float is the routine that converts a C floating-point value into a tagged floating-point value on the expression stack. The 16-digit value is rounded to 14-digits, pushed onto the estack, and then a FLOAT_TAG is pushed on top.

BCD floating point supports floating point infinities. However, **push_Float** converts these values to their symbolic equivalents. In other words, **push_Float** converts a floating point plus infinity to PLUS_INFINITY_TAG, a floating point minus infinity to MINUS_INFINITY_TAG, a floating point unsigned infinity to PLUS_OR_MINUS_INFINITY_TAG, and a floating point NAN to UNDEFINED_TAG.

BCD floating point supports an exponent range from -16384 to 16383. Tagged float exponents are limited to the calculator range of -999 to 999. **push_Float** converts overflow values to the corresponding symbolic infinity and underflow values to zero. Thus, while any tagged float can be moved into a C floating point variable, not all C floating point values can be converted to tagged floats.

Tagged floating point values are the floats available externally to the users of the calculators. TI BCD floating-point values (C floats) must be converted to tagged floats before displaying or storing to a calculator variable, and all the special floating-point values in the TI BCD floating-point system such as infinity and NAN (may also be referred to as undefined or invalid float) must be converted to the symbolic equivalents before being made available to the user. All of this is automatically handled by **push_Float**.

Occasionally an application developer may want to check for C float values not valid in a tagged float without doing the actual **push_Float** conversion. For example, an algorithm that has been written using TI BCD floating-point values may need to take different paths or throw an error based on whether the result of a previous operation was infinity or undefined. Routines such as **is_float_transfinite** and **is_nan** are available for this purpose. See **Appendix A: System Routines — Direct Floating Point Operations** for more routines that test for other special values. **round14** can be used on any BCD16 value to round the number of digits in the mantissa to 14. **ck_valid_float** rounds a BCD16 value to 14 digits, underflows to 0 if the exponent is less than -999, and returns a floating-point NAN if the original value is transfinite or the exponent is greater than 999. If **push_Float** had been used, the floating-point transfinite values and an exponent greater than 999 would have resulted in the symbolic equivalents on the estack. However, the NAN allows the developer to continue with the algorithm if desired but **is_nan** may be called directly after **ck_valid_float** to test for the NAN instead.

Since tagged floats have 14 digit mantissas, sometimes a series of operations performed with tagged floats may get a different result from the one obtained by doing the same series using BCD16 floats which have 16 digit mantissas. Usually the 16 digit mantissas result in greater accuracy and are preferred for that reason but a developer may want to match the external result which the user would see if he entered a particular expression on the command line, which would cause it to be executed on the estack and therefore use tagged floats. The BCD14 format is available for this purpose but it should be noted that a BCD16 value will not cast to a BCD14 value (i.e. there will still be 16 digits in the mantissa after the cast), and an explicit **round14** must be done in this case . It is recommended that tagged floats on the estack be used when trying to match external user results and that BCD16 floats be used when greater accuracy is desired.

See **Appendix A: System Routines — Direct Floating Point Operations** for details on the system routines that operate on BCD16 (double) arguments. Most of these routines compute and return a corresponding function value; for example, **sin**, **cos**, **tan**, **ln**, **sqrt**, etc. Others test for special values, for example, **is_float_infinity, is_float_positive_zero**, etc. Some are conversion routines. **estack_number_to_Float** is the primary routine for converting any tagged

number into a BCD16 float value. **push_Float** as previously described is the primary routine for converting a BCD16 float value into a tagged float value on the expression stack.

See **Appendix B: Global Variables — Direct Floating Point Operations** for details on commonly used stored BCD16 values and how to access them. Also see **Appendix B: Global Variables — Math** for a description of global EStackIndexes of stored floating point values.

16.5. Rational Numbers

The rational system operates on the expression stack. The range of tagged integers is approximately from -10^614 to 10^614, which is much larger than the range of C integer variables. This is the opposite situation from the float system. Any C integer can be converted to a tagged integer, but most tagged integers are too large to fit in C integer variables. Also C does not support a “fraction” variable type to correspond to tagged fractions.

The system provides some routines to convert between C integers and tagged integers. **push_long_to_integer**, **push_ulong_to_integer**, and **push_ushort_to_integer** provide the means to convert C integers to tagged integers on the estack. **estack_to_short** and **estack_to_ushort** convert tagged integers to C integers. See **Appendix A: System Routines — EStack Utilities** for descriptions of these routines.

Since the float range is bigger than the rational range, rational overflows and underflows quietly convert to float values. Clearly any rational value can be converted to a corresponding float value, but some floats are outside the rational range and cannot be converted to rational values. **estack_number_to_Float** is the primary routine for converting rational values to floating-point values. **push_Float_to_rat** is the primary routine for converting floating-point values to rational values. See **Appendix A: System Routines** for descriptions of these routines.

See **Appendix B: Global Variables — Math** for a description of global EStackIndexes of stored rational values.

16.6. EStack Arithmetic

Performing numeric operations on the expression stack is simple, because the system routines understand all the tagged data types in the internal tokenized form and how to operate on them appropriately. For example, to add two values, simply pass the two values to the **push_sum** routine. **push_sum** understands tagged integers, tagged fractions, tagged floats, and in fact, all algebraic data

types that can be added. The primary system routines for estack arithmetic are **push_sum**, **push_difference**, **push_product**, **push_ratio**, and **push_exponentiate**.

Rational values combine to form rational values unless the operation overflows or underflows. Since the float range is larger than the rational range, rational operations quietly overflow and underflow into float values.

Float values combine to form float values. Float operations overflow to the correctly signed symbolic infinity.

Rational values combine with float values to form float values. The rational values are converted to float values to facilitate these combinations.

In addition to the primary routines, the system provides some specialized routines. **replace_top2_with_sum**, **replace_top2_with_difference**, **replace_top2_with_prod**, **replace_top2_with_ratio**, and **replace_top2_with_pow** perform the corresponding operation on the top two entries on the expression stack. **add_to_top**, **subtract_from_top**, **times_top**, **divide_top**, and **raise_to_top** perform the corresponding operation with the top entry on the estack and a specified input argument.

See **Appendix A: System Routines — EStack Arithmetic** for descriptions of these and other routines for performing arithmetic operations on the expression stack.

16.7. Complex Numbers

The representation of complex values is different in the external and internal tokenized forms. The external tokenized form uses the I_TAG which represents the imaginary number. So, $1 + 2i$ tokenizes into an expression involving tagged integers, addition, multiplication, and the I_TAG. Since the I_TAG might appear anywhere in a general expression, this representation makes it difficult to recognize and operate on complex values.

The simplifier, via **push_internal_simplify**, converts complex values to internal tokenized form which uses an IM_RE_TAG on top of the imaginary part on top of the real part of the complex value. This form places the knowledge that the value is complex at the top of the representation in the form of the IM_RE_TAG. This change greatly facilitates recognizing and operating on complex values.

External tokenized values are only handled by **push_internal_simplify**, which converts them to internal form, and by **push_simplify_statements** and **push_simplify**, which use **push_internal_simplify**. Do not pass external tokenized form to other evaluation/simplification routines. External tokenized values are also handled by the display routines; for example, **display_statements**, **Parse1DExpr**, etc.

Internal tokenized values are handled by all of the evaluation/simplification routines. For example, **push_sum**, **push_difference**, **push_product**, and **push_ratio** automatically handle complex arithmetic. **push_abs** computes the magnitude of a complex value. **push_phase** computes the phase angle of a complex value. **push_sin**, **push_cos**, **push_tan**, **push_ln**, and so on, all understand complex values in internal tokenized form and compute and return the appropriate result in internal tokenized form. However, the display routines do not take this form. Do not pass internal tokenized form to **display_statements**, **Parse1DExpr**, etc. First, use **replace_top_with_post_simplified** to convert the internal form to external form. Then the result can be handled by the display routines.

17. Graphing

This chapter discusses the Graph application on the TI-89 / TI-92 Plus calculators and how to interface with it.

17.1. The Graph Screen

The graph screen always has an odd number of pixels vertically and horizontally, even if the window itself has an even number of pixels. In this case, the rightmost column and/or the bottom row will not be used for graphing. The odd number of pixels insures that there will always be a pixel in the center of the graph, which is where the origin of the axis is with the default window settings. The Window variable, xmin, corresponds to the value at the center of the leftmost pixel column, while xmax is the center of the rightmost pixel column used for graphing. ymin and ymax values correspond to the center of the pixels of the top and bottom rows respectively. Δx and Δy are measured from the center of one pixel to the center of the next and are computed when the graph screen is displayed. If the Window variables or screen size has changed since the last time the graph was displayed, Δx and Δy may not have been recomputed yet and may be invalid.

Figure 17.1 shows the four pixels in the upper left corner of the graph screen and the relationship between the x, y viewing window coordinates and the row, column pixel coordinates. Some system routines and TI‑BASIC calculator commands use pixel values and others require viewing window coordinates. For drawing objects such as lines, dots, and circles on the screen, the system routines described in section **11.2. Windows** are provided. These routines all expect pixel coordinate inputs. Several system routines, listed in section **17.6. Available Graph System Routines and Global Variables**, convert between pixel and viewing window coordinates.

It is important to remember that when using system routines or calculator commands with x, y viewing window coordinates as inputs that the outer half of the first and last columns and the outer half of the top and bottom rows are outside the viewing window. If the TI‑BASIC command PtOn xmin-$\Delta x/4$, ymax is entered on the Home screen, the upper left pixel on the graph screen will not be set. Even though xmin-$\Delta x/4$ is within the range covered by the first column (xmin-$\Delta x/2$ to xmin+$\Delta x/2$), it is less than xmin and therefore outside the viewing window.

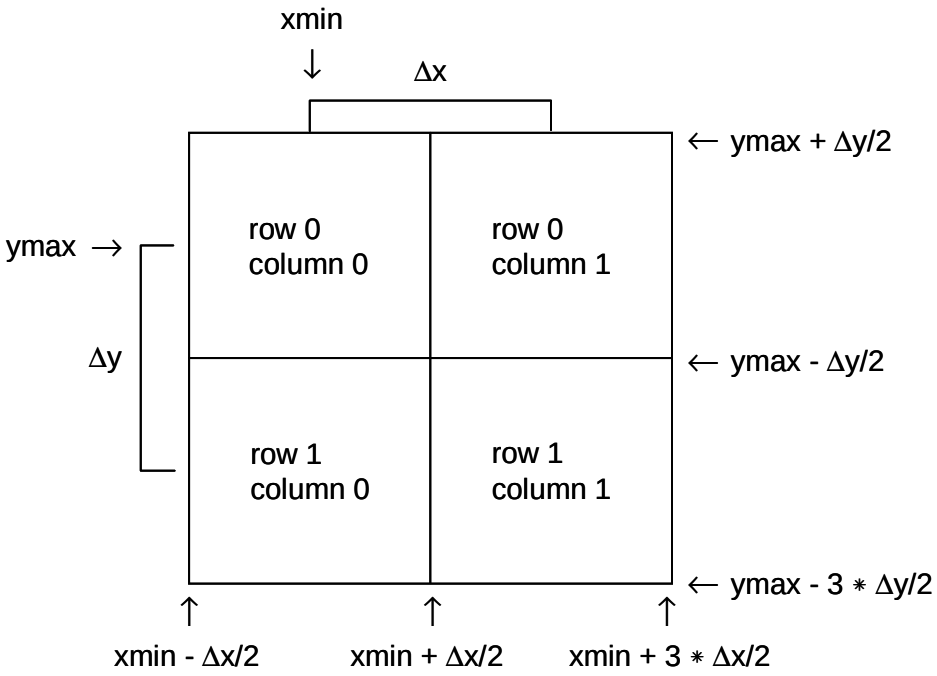


Figure 17.1: Upper Left Corner of Graph Screen

The graphing application has a backup screen associated with it. This enables the Smart Graph feature to work. As functions are graphed, the pixels are set in both the backup screen and on the display. If none of the formats, variables, or functions used during graphing have changed since the last time the graph was displayed, the backup screen can immediately be shown instead of regraphing all the functions. A few things are only drawn to the display, not the backup screen, such as axis labels which must be redrawn every time the graph is displayed, and cursor coordinates which are constantly changing as the cursor moves.

17.2. Working with the Graph Application

If an app or ASM will be interacting with the Graph application, it is probably a good idea to make sure the calculator is in one graph mode, by either setting MO_OPT_SPLIT_SCREEN = D_MODE_SPLIT_FULL or MO_OPT_NUMBER_OF_GRAPHS = D_MODE_GRAPHS_1 (see section **8.1. Mode Settings**), or an error can be displayed if the calculator is not in the correct mode when the app or ASM starts. In the default mode with one graph, any reference to mode settings, format settings, Window variables and editor, Y= functions and editor, Table, stat plots, or graph databases refers to the same graph. In split screen mode, the applications in both windows refer to the same graph, enabling you to see a graph and table, for example, generated from the same data. If the user changes to two graph mode after an application is open, a CM_MODE_CHANGE event message will be received by the app, allowing it to take any desired action.

There is always an active graph, even when the Graph application is not displayed. This allows access to the graph system variables and settings from the Home screen or any other application. The graph system functions are stored in the symbol table and appear on the VAR-LINK screen in the MAIN folder. All other data needed by the active Graph application is stored in a GR_WIN_VARS structure that is pointed to by the global variable **gr_active**. This struct also contains pointers to the current Window variables, which are kept in system memory, not part of the symbol table. The system routine **VarStore** must be used to store to the graph system variables. This insures that the values are valid and all necessary system flags will be set when appropriate. The following code sample demonstrates how to define a graph function from an app or ASM and shows an example of storing to the Window variables:

```
EStackIndex volatile old_top = top_estack;
EStackIndex name;
UCHAR buf[25];

TRY

/* buf = "Define y1(x)=x" */
memset(buf, 0, 25);
strcat((char *) buf, (const char *) XR_stringPtr(XR_DefineB));
strcat((char *) buf, (const char *) "y1(x)=x" );

/* Execute buf to define graph function y1(x) */
push_quantum( END_OF_SEGMENT_TAG );
push_parse_text( buf );
push_simplify_statements( top_estack );

/* store -10 to xmin */
push_parse_text( (UCHAR *) XR_stringPtr(XR_XMIN_STR));
name = top_estack;
push_Float( -10.0 );
VarStore( (BYTE *)name, STOF_ESI, 0, top_estack );

/* OK to access system variables directly, but not store. */
if((gr_active->rngp)[GR_XMAX] < 0.0 )
{       /* if xmax is negative, make it 10.0 instead */
   push_parse_text( (UCHAR *) XR_stringPtr(XR_XMAX_STR));
   name = top_estack;
   push_Float( 10.0 );
   VarStore( (BYTE *)name, STOF_ESI, 0, top_estack );  /* xmax=10 */
}
top_estack = old_top;  /* restore top_estack */

ONERR
top_estack = old_top;  /* restore top_estack */
PASS;

ENDTRY
```

17.3. Two Graph Mode

If the app or ASM may be executing when the calculator is in two graph mode and needs to interact with the Graph application, graphing system variables or graph system functions, or any graph related application (Window Editor, Y= Editor, or Table), it must be aware of how two graph mode works. In two graph mode (split screen selected and Number of Graphs = 2 on the MODE screen) two independent graphs can be shown at the same time. The graph mode for each is set separately. Other settings on the MODE screen are global and apply to both graphs, such as Current Folder, Angle mode and Complex Format. Stat plot definitions and graph functions are also global although different functions and stat plots can be selected for each graph.

In two graph mode, any graph related application or reference to graphing system variables or graph system functions always refers to the graph corresponding to the active split screen window. In the top or left split (AP_SIDE_A), this will always be Graph 1. Graph 2 is always in the bottom or right split (AP_SIDE_B). A calculator user keeps track of this visually, with the active graph number and its mode both shown in the status line which gets updated as the user switches from one window to the other. Graph 1 and Graph 2 can both be different graph modes or they can have the same mode. When they have the same mode, the function definitions and styles are shared (if the Y= Editor is displayed in both windows, they show the same function definitions), but different functions can be selected to be plotted in each screen. The Smart Graph feature still applies to each graph individually as much as possible so changing a function that is only graphed on one screen does not cause the other screen to also regraph. The Window variables, graph format settings, and table editors are completely independent for each graph, even when both have the same graph mode. If a graph is in one window and a table is in the other, one is using Graph 1 data and the other is using Graph 2 data. Two applications generated from the same set of graph data cannot be shown at the same time in two graph mode.

System apps and routines, including **VarStore**, access all graph related data through the global variables **gr_active** and **gr_other**. **gr_active** is a pointer to a GR_WIN_VARS struct containing all the information for the active graph. **gr_other** points to the information for the second graph in two graph mode. As the calculator user switches between the two windows in two graph mode, the pointers in **gr_active** and **gr_other** are swapped so that **gr_active** is always referring to the active graph. An app or ASM will be referencing graph related data that corresponds to whichever window is active when the app or ASM is executing. This means that the first time your app is opened, **gr_active** may be referring to Graph 1, and if it is opened again, it may be referring to Graph 2, depending on which window the app happens to be opened in. Although the user refers to Graph 1 and Graph 2 to distinguish between the two graphs, internally an app or ASM is usually not aware whether it is working with Graph 1 or 2 since

gr_active and **gr_other** can point to either. An app should also be aware that a user can change graph modes or even change to one graph mode while the app is open, which may cause the app to suddenly start referencing the other graph. If any MODE settings are changed while the app is open, a CM_MODE_CHANGE event message will be sent to the app (see section **8.1. Mode Settings**). An ASM can change which window is active, allowing it to choose Graph 1 or Graph 2 if desired.

When the calculator is returned to one graph mode, the graph that is kept as the current graph will be the one corresponding to the active split screen at that time. If the top or left split is the active window, Graph 1 will be the current active graph. If the bottom or right split is the active window, Graph 2 will be the current graph. The data for the graph that is not current is not lost, however. If the graphs were different modes, selecting the mode of the other graph will restore that graph as the current active graph. If both graphs were the same mode, all the data is saved but can only be viewed again by going back into two graph mode and setting both graphs to the same mode again. The graph formats and window settings for the second graph will be the same ones that were there before. Since the functions are shared, they will contain the current definitions. If the previous definitions are desired, a graph database should be created before leaving two graph mode.

17.4. Graphing Functions

Each variable in the symbol table has two graph reference flags, one for the graph associated with **gr_active** and the other for the graph associated with **gr_other**. Before starting a graph, the **gr_active** graph reference flags and graph backup screen are cleared and the graph in progress flag is set (`gr_flags.gr_in_progress`). While the graph in progress flag is set, the graph reference flag for every variable accessed will be set. The graph in progress flag is reset when the graph is stopped for any reason, whether it is complete or not. The dirty flag (`gr_active->gr_win_flags & GR_DIRTY`) is used to tell the system that the graph must be regraphed the next time it is displayed. It is set if the graph is interrupted for any reason, leaving an incomplete graph on the screen, so the next time the graph screen is displayed another regraph will occur. When the graph is complete and error-free, the dirty flag is reset. These flags are the basis for the Smart Graph feature. Any time a variable is changed, the graph reference flags are checked. If either is set, the dirty flag for the appropriate graph (**gr_active** or **gr_other** or both) will be set, triggering a regraph the next time that graph is displayed. Many other things can also cause the dirty flag to be set, such as selecting a new split setting, changing the angle mode, changing any of the Window variables, changing a selected graph system function, etc. Selecting an additional graph function or defining a new graph function (which automatically selects it for graphing), does not set the dirty flag. When a new function is added to the graph, the dirty flag and graph in progress flags operate

as described above. The only difference is that the backup screen and the graph reference flags in the variables are not cleared first.

During graphing, each function is evaluated at every point that the trace cursor will fall on naturally to insure that the cursor will always be directly on the function when traced. To retain floating point accuracy when line clipping is necessary and when computing the values to use for the independent variable in each graph mode, the Window variables xmin, xmax, ymin, ymax, tmin, tmax, etc., are limited to 12 significant digits in the mantissa, while Δx, Δy, tstep, etc. use all 14 significant digits available in a floating-point number. **VarStore** automatically rounds values to 12 digits when storing to the min/max Window variables. The first x value plotted in function mode graphing is always xmin. The last x value will either be xmax or, if no trace point falls on xmax due to the value of xres, the first xres increment greater than xmax insuring that the graph of the function does not end before the edge of the screen. In the modes with an independent variable other than x, the first value is tmin, θmin, etc., and the final value is the last computed value for the independent variable that does not go beyond tmax, θmax, etc.

Each segment of the graph is drawn as the functions are evaluated at every computed value of the independent variable. Either or both of the end-points of any segment may be outside the viewing window, so that line clipping is required. Line clipping involves interpolating using the given end-points and the viewing window variables. The system routine **GrLineFlt** performs all necessary clipping based on the Window variables, while drawing the line segment in the specified style.

Most errors encountered while graphing will cause the graph to stop, leaving the dirty flag set so the graph will be regraphed the next time it is displayed. However, the errors FIRST_OVERFLOW, FIRST_ZERO_DIVIDE, FIRST_DOMAIN_ERR, ER_SINGULARMAT, and FIRST_UNREAL_ERR are ignored while graphing, merely causing the point where the error occurred to be skipped. The function is evaluated as usual at the next value of the independent variable and the dirty flag is not set.

17.5. Graph Application Memory Usage

In the RAM area set aside for system use, memory is permanently reserved for two GR_WIN_VARS structs, two sets of Window variables and graph format settings for all six graph modes, and two sets of table variables among other things, to insure that all data is available for two graph mode. During system initialization, the **gr_active** graph window is opened (although not displayed since the Home screen is shown at first) and its backup screen is created. The backup screen for this graph window is always large enough for a full screen graph, even if the calculator is later put into split screen mode.

When two graph mode is entered, memory for another backup graph screen is reserved. This one is the size of the largest window allowed in any split screen setting. If there is not enough heap available, a memory error will be displayed. The calculator will be in two graph mode but a memory error will be shown every time the Graph application is selected for Graph 2. This does not affect the other graph related applications for Graph 2 or anything in Graph 1. The Table, Y= Editor, and Window editor for Graph 2 are all still available. Graph databases can still be opened or created. To be able to show the Graph application for Graph 2, the calculator must be returned to one graph mode and enough variables must be deleted or archived to make room for the backup screen before re-entering two graph mode.

The Graph application also uses lots of temporary memory while graphing. Anytime a user-defined function or program is executed, a temporary folder is created for the local variables. During graphing, the same folder is used for all the graph functions so that time is not wasted by constantly creating and deleting the temporary folder for each separate function. Many arrays of data are needed for 3D graphs, and sequence mode and differential equation mode both need to save lists of previously computed values. In addition, functions created by the Graph or Table commands are stored in another temporary folder which is deleted by executing the ClrGraph command or activating the Y= Editor application.

17.6. Available Graph System Routines and Global Variables

Any TI-BASIC graph command not specifically listed here can be accessed by entering the command as a string and executing it as described in section **8.4. Interfacing with TI-BASIC**.

Graph Global Variables:

gr_active — Pointer to the GR_WIN_VARS structure containing graph information for the active graph.

gr_other — Pointer to the GR_WIN_VARS structure containing graph information for the nonactive graph in two graph mode.

gr_flags — Structure containing flags used by the Graph application.

Graph System Routines:

CkValidDelta — Verify that Δx, Δy, or the step value of the independent graph variable has a valid exponent.

cmd_clrdraw — Calculator command ClrDraw.

cmd_clrgraph — Calculator command ClrGraph.

cmd_rclgdb — Calculator command RclGDB.

cmd_stogdb — Calculator command StoGDB.

CptDeltax — Compute graph system variable Δx for the current active graph.

CptDeltay — Compute graph system variable Δy for the current active graph.

CptFuncX — Compute the x value based on the current Window variables for a specified pixel.

CptIndep — Compute the value of the independent variable for the specified iteration.

EQU_select — Turn on/off/toggle the select flag for the specified graph system function.

EQU_setStyle — Set the style of the specified graph system function.

FindFunc — Return the HSYM of the specified graph system function if it is selected for graphing.

FindGrFunc — Return a pointer to the symbol table entry of the specified graph system function.

gr_CptIndepInc — Compute the iteration for the given value of the independent variable.

gr_delete_fldpic — Delete the graph system variable fldpic if it exists.

gr_DispLabels — Draw the graph axis labels on the graph screen.

gr_xres_pixel — Find the first pixel number that is a multiple of the graph system variable xres and is greater than or equal to the given pixel number.

GraphActivate — Activate the Graph application if not already active.

GrAxes — Draw the axes for the specified graph.

GrClipLine — Clip the end-points of the specified line if necessary based on the current Window variables.

GrLineFlt — Draw the specified line, using the given style, on the current graph screen.

GT_Regraph — Force a regraph of the current graph.

GT_Regraph_if_neccy — Regraph the current graph if necessary.

StepCk — Verify that the step value of the independent variable for polar or parametric mode is valid.

XCvtFtoP — Convert the given floating point x coordinate to a pixel column number based on the specified Window variables.

XCvtPtoF — Convert the given pixel column number to the corresponding floating point x coordinate at the center of that column, based on the specified Window variables.

YCvtFtoP — Convert the given floating point y coordinate to a pixel row number based on the specified Window variables.

YCvtPtoF — Convert the given pixel row number to the corresponding floating point y coordinate at the center of that row, based on the specified Window variables.

18. TI **FLASH** Studio **(IDE) Overview**

18.1. Introduction

TI **FLASH** Studio™ is a development tool that uses an Integrated Development Environment (IDE) to give the user a familiar Windows interface. The TI **FLASH** Studio provides the capability to simulate the TI-89 / TI-92 Plus calculator on the PC to allow application development and debugging. The TI **FLASH** Studio allows the developer to use a set of development tools under the control of a single interface. The tools that are accessible through the control of the IDE include a project manager, a language sensitive editor, compiler, assembler, linker, and a simulator/debugger.

18.2. Development System

The IDE is for the development of Apps and assembly programs. The steps for setting it up and getting started are presented in the following sections.

The IDE allows the user to:

- Create project files.
- Use templates to create projects.
- Create and edit source files.
- Build executable software for the simulator.
- Build downloadable software for developer calculators (Educational and Professional versions).
- Integrate simulator/debugger functions.

18.2.1. Requirements

To properly run TI **FLASH** Studio, the development system PC must meet the following requirements:

- IBM PC compatible Pentium-based machine.
- 32 MB of RAM (64 MB recommended).
- VGA video adapter.
- 35 MB of available hard drive space.
- Mouse or pointing device.

- Microsoft Windows 95, Windows 98, Windows ME or Windows NT 4.0.
- Microsoft Virtual Machine (Microsoft VM) build 3319 or higher. Microsoft VM can be downloaded from the Internet at http://www.microsoft.com/java/download.htm.

Also, it is recommended that the development system contain the following features:

- Adobe Acrobat Reader 4.0. or higher.
- A screen resolution of 800X600 or better.
- Serial connection port and a TI-GRAPH LINK™ cable for communication with the calculator.
- 150 MHz processor or faster.

18.2.2. Installation

Visit the Texas Instruments Developer's World to obtain the latest software.

1. Review the readme file to obtain updated information and requirements.
2. If the system does not contain Microsoft VM, download the Microsoft VM from the Internet prior to installing TI **FLASH** Studio.
3. Download TI **FLASH** Studio and save in a temporary location on the development computer.
4. Install TI **FLASH** Studio by navigating and double clicking from the Windows file manager or by using Start/Run menu and typing the filename that was saved from the download (default is FSInst.EXE).
5. Follow the install procedure. The system may require a reboot prior to starting the program.
6. TI **FLASH** Studio is installed in the default directory found on the Start menu under Programs/TI FLASH Studio.

Caution:	Sierra C™ Assembler tools are installed as a part of the setup in the C:/Sierra directory. Any previously existing version of Sierra tools at this location will be overwritten.

18.2.3. Compiler/Assembler/Linker

A compiler, assembler, and linker are installed with TI **FLASH** Studio. The user can write software in C and create calculator programs and applications using this compiler. The language tools are customized to provide code for the TI calculators and the license that must be accepted prohibits other use of the language tools. More information on the language tool can be found in the TI‑89 / TI‑92 Plus Sierra C Assembler Reference Manual.

For most development, the specific configuration of the language tools will be transparent to the user. There are ways to change the command line switches as discussed in the TI‑89 / TI‑92 Plus Sierra C Assembler Reference Manual.

18.2.4. Simulator/Debugger

The TI **FLASH** Studio simulator/debugger allows the user to load and debug their applications. TI-89 and TI-92 Plus calculators are simulated. TI **FLASH** Studio supports applications written in C for the 68000 family of processors.

18.2.5. IDE Overview

TI **FLASH** Studio provides the user with an intuitive, easy to use graphical interface. Invoke TI **FLASH** Studio by double clicking, from the Windows file manager, the TI Flash Studio.exe or from Start/Programs/TI Flash Studio. The TI **FLASH** Studio user interface is composed of several windows that allow access to various parts of the IDE. The Home screen is shown in Figure 18.1.

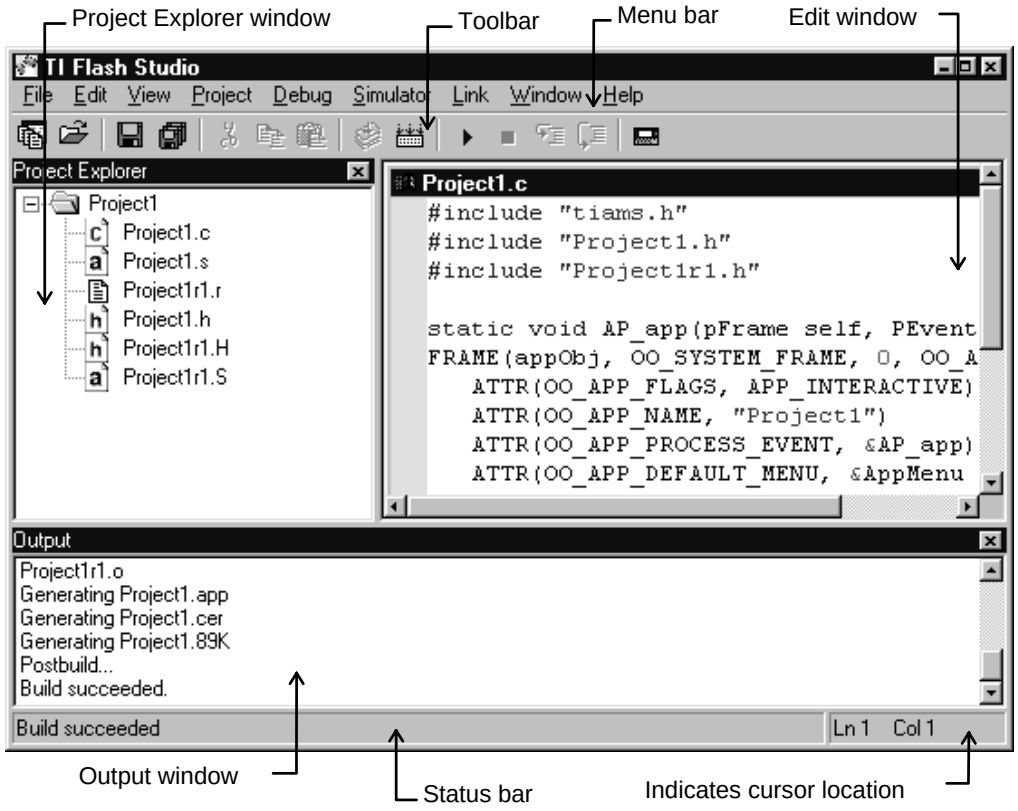


Figure 18.1: TI FLASH Studio Home Screen

A window in the user interface can be moved and docked with another window by clicking and holding the window's grip bar. Windows displayed in Figure 18.1 include: Project Explorer, Edit, Output, and Status bar.

- A Project Explorer window uses a graphic tree to display the project's source files, dependencies, and object files. Dependencies are automatically updated when opening a project.
- An Output window allows the user to observe for errors during the compile, assemble, and link phases of the build process. All generic output of TI **FLASH** Studio is written to this window.
- An Edit window is used to edit and debug the code. The Edit window uses a color-coded language-sensitive editor that shows instructions, comments, and assembler directives.
- The Status bar is located at the bottom of the window and indicates the status of the simulator/debugger. The current location of the cursor is displayed towards the right side of the window.

18.2.6. Uninstalling

When TI **FLASH** Studio is installed, an uninstaller is created on the host PC. When removing TI **FLASH** Studio from the host PC, use the uninstall utility to properly restore the Windows operating environment.

To uninstall TI **FLASH** Studio:

- Select Start/Programs/TI FLASH Studio/Uninstall TI FLASH Studio from the windows toolbar.
- Choose the automatic uninstall utility. Click 'Yes' when asked if you are sure to completely remove TI **FLASH** Studio and all its components.
- The Uninstall shield removes all elements of TI **FLASH** Studio from the host PC.

18.2.7. Support

A user discussion group is available to share information. A link for the Software Development Kit discussion group can be found at www.ti.com/calc/developers/support.htm. TI provides e-mail support to users that purchase an Educational or Professional versions. Bugs can be reported via the Problem Report Form on the TI web site at www.ti.com/calc/developers/sdkproblemreport.htm.

18.2.8. References

There are various reference manuals available that contain more information on calculator programming. These are useful resources for developing TI-89 / TI-92 Plus applications. They include:

TI-89 / TI-92 Plus Developer Guide (this book)
TI-89 / TI-92 Plus Sierra C Assembler Reference Manual
TI-89 / TI-92 Plus Graphing Calculator Guidebook

18.3. TI FLASH Studio Interface

Invoke TI **FLASH** Studio, see section **18.2.5. IDE Overview**. The IDE presents the Home screen. This section describes the menu and toolbars of the TI **FLASH** Studio user interface, see Figure 18.2 and Figure 18.3.

Selecting a menu item performs one of these functions:

- Selecting a menu item with an arrow displays a submenu.
- Selecting a menu item without an arrow causes the selected task to be automatically executed.

The default toolbar allows quick and convenient access to the menu items.

Figure 18.2: TI FLASH Studio Menu Bar

Figure 18.3: TI FLASH Studio Toolbar

18.3.1. File Menu

The File menu items are used to create, open, and save projects, see Figure 18.4.

File management is currently not supported by TI **FLASH** Studio. However, file management (i.e., deleting, renaming, etc. of projects or project files) can be accomplished through the Windows Explorer.

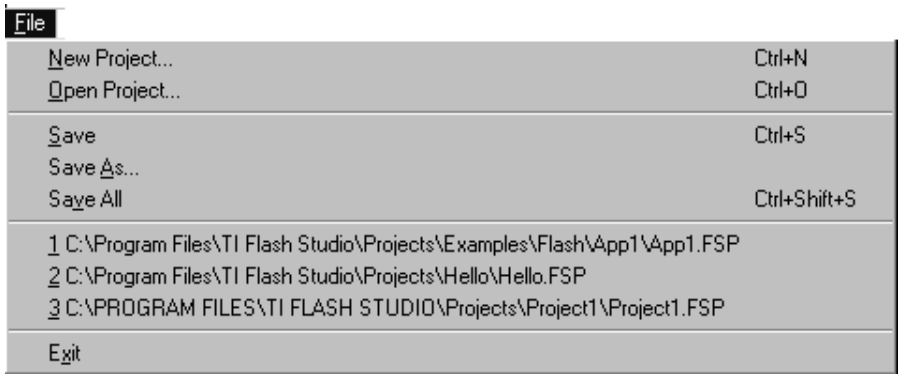


Figure 18.4: File Menu

Item	Icon	Action
New Project		Creates a new project. The basic templates are automatically created whenever a new project is created. The user can modify these templates based on the application.
Open Project		Opens an existing project. Some example project templates are provided and can be loaded using this menu item.
Save		Saves currently active file.
Save All		Saves all open files and the project configuration.
<Project listings>		Lists four of the most recently used projects. This provides the user with a quick and convenient method to load recent projects.
Exit		Exits TI **FLASH** Studio. If any files have been modified, TI **FLASH** Studio will ask the user if they should be saved.

18.3.2. Edit Menu

The Edit menu contains the edit commands displayed in Figure 18.5. All commands are limited to the text in the edit window.

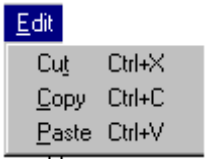


Figure 18.5: Edit Menu

Item	Icon	Action
Cut		Removes the currently selected text and copies it to the clipboard.
Copy		Copies the currently selected text to the clipboard.
Paste		Pastes items in the clipboard to the active window's cursor location.

18.3.3. View Menu

The View menu items allow the user to customize the TI **FLASH** Studio interface, see Figure 18.6.

Figure 18.6: View Menu

Item	Action
Project Explorer	Opens the Project Explorer window.
Output	Opens the Output window.
Registers	Shows the CPU view or, in other words, the contents of the 68000 processor registers and the flags (see Figure 18.7 for more information).
Autos	Displays the Watch window with default symbols and their values.
Watch	Displays the Watch window with user defined symbols and their values. (See Figure 18.9 for more information.)

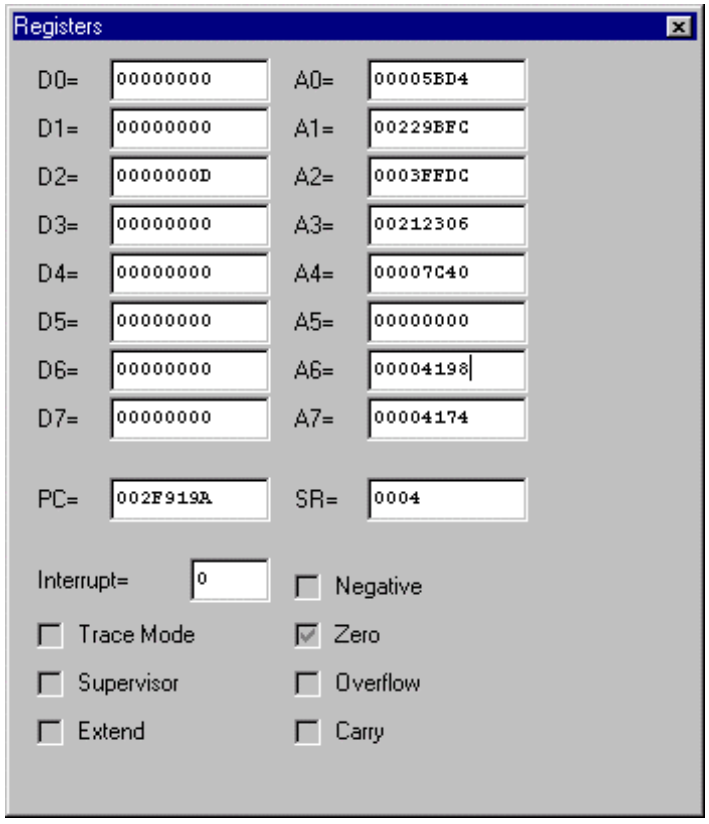


Figure 18.7: Registers

Column	Information
D0 to D7	These fields represent the values of the eight 32-bit general purpose data registers.
A0 to A7	These fields show the values of the 32-bit address registers. The first seven registers (A0 to A6) and the user stack pointer are used as software stack pointers and base address registers.
PC	Represents the 32-bit Program Counter.
SR	Represents the Status Register. The SR contains the interrupt mask (eight levels available) and the following condition codes: overflow, zero, negative, carry, and extend. Additional status bits indicate that the processor is in the Trace mode and/or in the Supervisor state. See Figure 18.8. Bits 5, 6, 7, 11, 12, and 14 are undefined and reserved for future expansion.

Column	Information
Interrupt	This field indicates the interrupt priority, ranging from 1-7. The status register contains a 3-bit mask indicating the current interrupt priority, and interrupts are inhibited for all priority levels less or equal to the current priority.
Negative	Indicates if the negative flag is set(N).
Zero	Indicates if the zero flag is set(Z).
Overflow	Indicates the status of the overflow flag(O).
Carry	Indicates the status of the carry flag (C).
Extend	Indicates the condition code for extend(X).
Trace Mode	To aid in program development, the 68000 processor includes a facility to allow tracing following each instruction. This field is set when Trace Mode is enabled. When tracing is enabled, an exception is forced after each instruction is executed. Thus, a debugging program like the TI **FLASH** Studio, can monitor the Program under test.
Supervisor	Indicates if the processor is in Supervisor mode. The processor executes instructions in one of two modes — User mode or Supervisor mode. The User mode provides the execution environment for the majority of application programs. The Supervisor mode allows some additional instructions and privileges and is used by the Operating System and other system software.

Note: Please refer to the M68000 8-16-32-Bit Microprocessors User's Manual for more information.

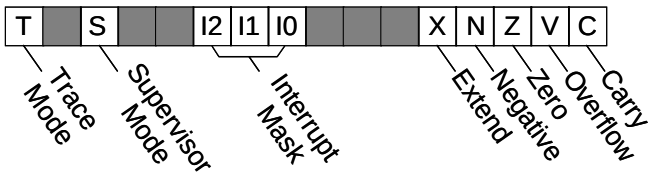


Figure 18.8: Status Register

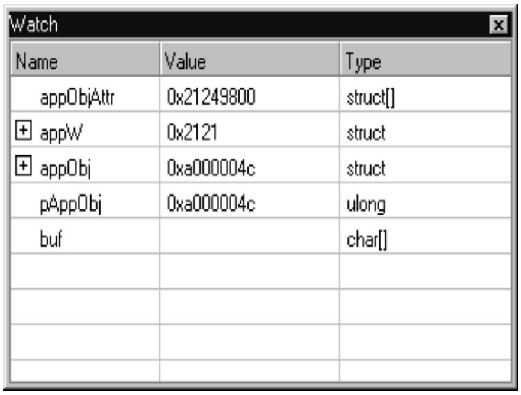


Figure 18.9: Watch

Column	Information
Name	Displays the symbol's name.
Value	Shows the symbol's value in hexadecimal.
Type	Shows the symbol's type, which can be struct, int, long, etc.

Note: Symbol information in the Watch window is updated every time a debug operation is processed.

18.3.4. Project Menu

The Project menu contains items necessary for compiling and building a project, see Figure 18.10.

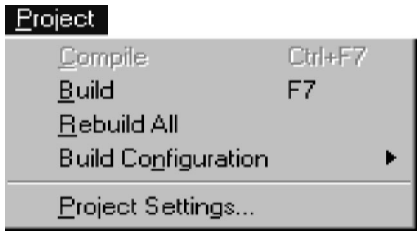


Figure 18.10: Project Menu

Item	Icon	Action
Compile		Assembles/Compiles the currently open file in the Editor window. The Sierra C compiler/assembler is invoked during this step. Sierra tools are installed as part of the setup.
Build		Builds all the files in a project that are newer than the object file.
Rebuild All		Rebuilds all files in a project from scratch, regardless of what files have been modified.
Build Configuration		Allows customization of the parameters of the build process, i.e., Release mode or Debug mode.
Project Settings		Allows the user to change the commands for assembling, linking, compiling and building of the project. The default values for both Release mode and Debug mode can be viewed from this submenu item.

18.3.5. Debug Menu

After a new project is created and all settings are configured, the next step is to debug and build the file, see Figure 18.11.

Figure 18.11: Debug Menu

Item	Icon	Action
Step Into		Single steps through the instructions in the program, and enters each function call that is encountered. This menu item is activated only when a breakpoint is hit.
Step Over		Steps over functions and macros. Single steps through instructions in the program. Executes without stepping through the function instructions when this command is used as a function call. This menu item is activated when a breakpoint is hit.
Breakpoints		Opens the Breakpoints submenu (see Figure 18.12 for more detailed information). Breakpoints can also be set, removed, enabled, or disabled at any point in the code by right clicking the mouse at that location on the edit window.
Trace Options		Opens the Trace Options submenu. The range of the trace can be set from this submenu.

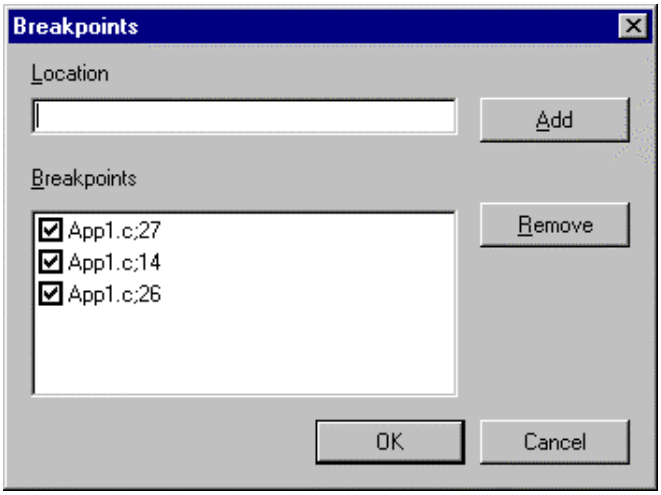


Figure 18.12: Breakpoints Submenu

Option	Function
Location	Specify the address location where you would like to add the breakpoint.
Breakpoint	List all the breakpoints.
Add	Adds a breakpoint at that specified memory location.
Remove	Removes the highlighted breakpoint.

Note: To remove the breakpoint, highlight the breakpoint (not just select it) and press the Remove button.

18.3.6. Simulator Menu

The Simulator menu items perform various actions on the TI-89 / TI-92 Plus simulator, see Figure 18.13.

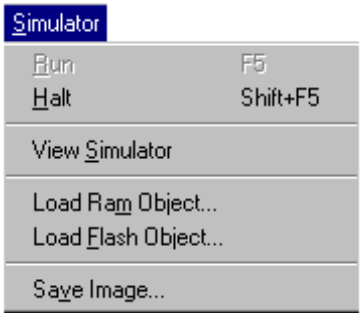


Figure 18.13: Simulator Menu

Item	Icon	Action
Run		Starts the simulator corresponding to the project type.
Halt		Stops the simulator.
View Simulator		Brings up the calculator (simulator) image.
Load RAM Object		Loads the TI-89 / TI-92 Plus RAM applications into the corresponding simulator/debugger.
Load Flash Object		Loads the TI-89 / TI-92 Plus Flash applications into the corresponding simulator/debugger.
Save Image		Saves the calculator image into a user-specified file (<filename>.clc).

18.3.7. Link Menu

The Link menu allows the user to communicate between TI **FLASH** Studio and the calculator, see Figure 18.14.

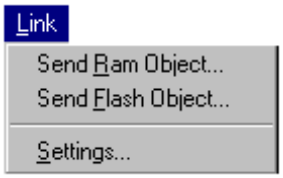


Figure 18.14: Link Menu

Item	Action
Send RAM Objects	Allows the user to download TI-89 or TI-92 Plus RAM applications to the corresponding calculator.
Send Flash Objects	Allows the user to download TI-89 or TI-92 Plus Flash applications to the corresponding calculator.
Settings	Allows the user to configure the communication (serial) port settings and the type of TI-GRAPH LINK cable used.

18.3.8. Window Menu

The Window menu contains the following standard Windows commands, see Figure 18.15.

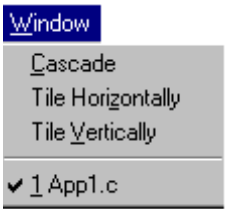


Figure 18.15: Window Menu

Item	Action
Cascade	Displays all open windows in a cascade format.
Tile Horizontally	Displays all open windows in a horizontal tile format.
Tile Vertically	Displays all open windows in a vertical tile format.

18.3.9. Help Menu

The Help menu has only one menu item, About, see Figure 18.16. This item displays the current version of the TI **FLASH** Studio in a pop-up window.

Figure 18.16: Help Menu

18.4. Example

This example walks through the TI **FLASH** Studio application development process.

18.4.1. Creating a Flash Studio Project

To create a new project, select the File menu and then the New Project submenu. In the New Project dialog box, set the name of the project and select the type of application you want to create, see Figure 18.17. The four possible application types are TI-89 — Flash Application, TI-89 — RAM Application, TI-92 Plus — Flash Application, and TI-92 Plus — RAM Application. The basic templates are automatically created whenever a new project is created. The user can modify and add code to these templates based on the application.

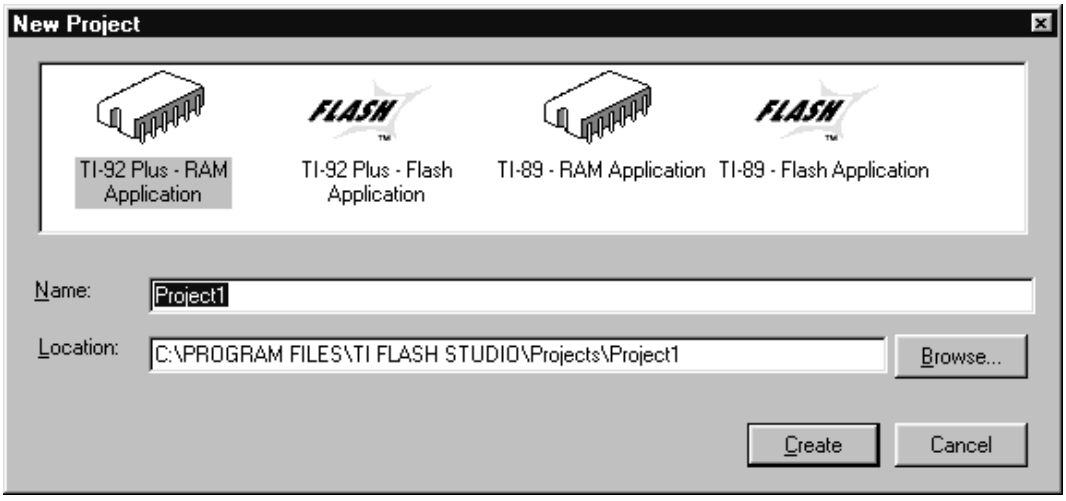


Figure 18.17: New Project Screen

To open an already existing project or one of the example applications, select the File menu and then the Open Project submenu.

18.4.2. Building the Application

Select Project menu, and then Rebuild All submenu. The TI development architecture is based on the TI **FLASH** Studio using the Sierra C compiler and Assembler. This step calls the Sierra C Assembler, which compiles, assembles and links the code. The following text should appear in the output window:

```
Prebuild
Compiling resources . . .
RAM Resource compiler version 2.13
.
.
.
Build Successful.
```

Failure in the build process implies bugs in the source code.

18.4.3. Loading the Application into the Simulator

After a successful build, the unsigned app is created in the C:Programs/TI **Flash** Studio/Projects/<Project Name> directory. The default extension is <file name>.8xk for TI-89 application and <file name>.9xk for TI-92 Plus application. Selecting the Load RAM Object submenu or the Load Flash Object submenu from the Simulator pull down menu allows the user to load the RAM application or Flash application into the simulator.

18.4.4. Debugging the Application

The TI **FLASH** Studio uses two files for debugging:

- <filename>.89d which contains debug information (breakpoints).
- <filename>.clc which contains the calculator memory contents.

Various debug tools like Step Into, Step Over, Trace log and the ability to set breakpoints are available from the Debug menu.

18.4.5. Terminating TI FLASH Studio

Selecting Exit from the File menu allows the user to terminate the session. The user will be prompted to save the changes made to the code before exiting.

Note:	The default extension for the saved application is <file name>.8xk or <file name>.9xk depending on whether it is a TI-89 application or a TI-92 Plus application. The <file name>.clc file is also saved.

18.4.6. Preparing the Application for Site Testing

In normal calculator usage, an application is installed in a calculator by downloading it from a PC or another calculator via the link cable. When the app is received, it is examined for a valid TI digital signature by the Operating System loader. All Flash apps to be distributed must be digitally signed by TI before they will be accepted by the Operating System. Since all apps must be signed, an app build must go through TI before it can be loaded in the normal way. Since signing is 1) an external process for developers, 2) is limited, and 3) has turnaround time associated with it, a single calculator debugging technique is available to facilitate code development.

The debugging technique used with the TI **FLASH** Studio circumvents this restriction, but it only works on the simulator, not a real calculator. Once an application has become well developed, some developers may need to perform testing on their calculator. The Educational and Professional tools allow the developer to download an application to their developer calculator. To support this need, TI may issue certificate for the testing calculator and set the developer up so that they can sign the app instead of TI. In this situation, the app will only run on the selected calculator.

18.4.6.1. Educational and Professional Developers

TI will assign the developer a Developer ID, generate a digital key and create a unit certificate for the calculator. The key is contained in one file: <filename>.key. The unit certificate file will be named: <filename>.89q or <filename>.9xq depending on whether it is for a TI-89 or TI-92 Plus calculator. Typically all this information can be emailed to the developer.

After receiving the key file and the unit certificate, you are ready to sign your apps for site testing. The procedure is as follows:

1. Make sure the <filename>.key file is in the same directory as the TI **FLASH** Studio executable (TI Flash Studio.exe). Default directory is C:/Programs/TI Flash Studio.

2. When you load and build the app, a directory with the same name as your key file is created in the C:/Programs/TI Flash Studio/Projects/<Project Name>/<filename> directory. This directory has the signed application (.9xk file). Please note that an unsigned app is also created in the C:/Programs/TI Flash Studio/Projects/<Project Name> directory.

3. Download the developer certificate to your calculator (the .89q or .9xq that has been emailed to you).

4. Now you can download the signed app to your calculator using the TI-GRAPH LINK cable. Please use the latest version of the TI-GRAPH LINK software (v2.1).

18.4.7. Preparing for Public Release

When applications are ready to be distributed, they must go through a signing process at Texas Instruments. When they are ready, Educational and Professional developers will be sent a set of instructions for the most current signing process.

Glossary

Aggregate	Either a list or a matrix.
AMS	**A**dvanced **M**athematics **S**oftware.
apps	Downloadable Flash applications.
ASCIIZ	**A**merican **S**tandard **C**ode for **I**nformation **I**nterchange **Z**ero — a convention for encoding characters and numerals in a seven or eight-bit binary number which is terminated with a zero byte.
ASM	Assembly-language program.
bignum	Another name for the rational number system that includes tagged integers and tagged fractions.
Dirty	All equations and the variables referenced by those equations are monitored after a graph is complete. If any of the equations or referenced variables changes, the graph is marked as “dirty” so that the next time the user views the graph it is regraphed.
estack	An abbreviation for expression stack.
expression stack	A generalized stack defined as an array of Quantums. The tokenizer and simplifier use this stack to perform their tasks.
external form	The tokenized Polish representation produced by the parser and provided to the 1D and 2D display procedures. External form is a contiguous representation consisting of tagged constants, variables, unary tags on top of one expression, binary tags on top of two expressions, or other tags on top of a tail. External form has a direct correspondence to the 1D input and output.
Freeware	Programs or databases that an individual may use without payment of money to the author. Commonly, the author will copyright the work as a way of legally insisting that no one change it prior to getting approval. Commonly, the author will issue a license defining the terms under which the copyrighted program may be used. With freeware, there is no charge for the license.
Garbage collection	A procedure that automatically determines what memory is no longer being used and recycles it for other use. This is also known as heap compression.

Heap	The Heap is an area of memory where all dynamic data is allocated from. All references to the heap are through handles which are unsigned 16 bit quantities. Dereferencing a handle returns a pointer to the actual data for the handle. Unless the handle is locked, the data pointed to by a dereferenced handle may change whenever the heap is compressed.
IDE	**I**ntegrated **D**evelopment **E**nvironment / TI **FLASH** Studio™.
internal form or internally-simplified	A form similar to external form but with fewer tags and some tags that do not occur in external form. For example, SUBTRACT_TAG, DIVIDE_TAG, CHS_TAG, LOG_TAG, SIN_TAG, COS_TAG, TAN_TAG, SINH_TAG, COSH_TAG, TANH_TAG, . . . do not appear in internal form. Complex expressions (those having nonzero imaginary parts) are represented using two expressions under an IM_RE_TAG rather than using an I_TAG.
Localization	Changing the calculator to use the local language of another country.
Operating System (OS)	The software loaded on all TI‑89 and TI‑92 Plus calculators. The OS contains the features that are of interest to customers, as well as behind-the-scenes functionality that allows the calculator to operate and communicate.
Pretty print	Format a mathematical expression so it is easier to read with the goal of making the expression look the way it customarily appears in math text books.
Quantum	Defined in the system by the C declaration typedef unsigned char Quantum.
SDK	**S**oftware **D**evelopment **K**it — a set of tools that allow developers to write software for specific platforms.
Shareware	Sometimes called **User Supported** or **Try Before You Buy** software. Shareware is not a particular kind of software, it is a way of marketing software. Users are permitted to try the software on their own computer systems (generally for a limited period of time) without any cost of obligation. Payment is required if the user has found the software to be useful or if the user wishes to continue using the software beyond the evaluation (trial) period. Payment of the registration fee to the author will bring the user a license to continue using the software. Most authors will include other materials in return for the registration fee—like printed manuals, technical support, bonus or additional software, or upgrades.

	Shareware is commercial software, fully protected by copyright laws. Like other business owners, shareware authors expect to earn money from making their software available. In addition, by paying, the user may then be entitled to additional functions, removal of time limiting or limits on use, removal of so-called **nag** screens, and other things as defined in the documentation provided by the program's author.
System-protected	Includes all user data types EXCEPT expressions, relations, strings, lists, and matrices.
System routines	Callable locations in the Operating System corresponding to pieces of code that exhibit some coherent functionality.
tag	A single Quantum value that is used in the tokenized form to represent an element of the structure or to delimit an element whose representation requires more than a single Quantum.
TI-BASIC	The programming language commonly used on the TI-89 and TI-92 Plus. It is the language that is used for PRGM variables. Its main drawback is that these programs run slower, since it is an interpreted language, rather than a compiled language.
top tag	The tag at the highest address.

Appendix A: System Routines

The following is the format in which each of the entry points will appear. The entry points are listed alphabetically by category.

Name Name used to identify a function.

Declaration: How the function is declared. Arguments appear in italics.

Category(ies): Algebra Utilities, Data Utilities, etc.

Description: Brief description of usage/purpose. How the function works.

Inputs: Explanation of input parameters and anything that needs to be set up before calling.

Outputs: Explanation of return information.

Assumptions: Description of appropriate usage context, limitations, and any other useful information.

Side Effects: Description of any side effect to be noted, including whether heap compression may occur or if an error may be thrown.

Availability: Whether it is available on TI‑89, TI‑92 Plus, or both and on which version of the AMS.

TI‑89 / TI‑92 Plus Differences: Explanation of any differences that may exist between the function's operation on the different calculators.

See Also: List of similar functions/global variables or functions/global variables to be used in conjunction with this one. References to other documentation if applicable.

Example:

```
Sample code.
```

Appendix A: System Routines — Algebra Utilities

See Also:

are_expressions_identical

Declaration: Boolean **are_expressions_identical** (EStackIndex *i*, EStackIndex *j*)

Category(ies): Algebra Utilities

Description: Determines whether the expressions indexed by *i* and *j* are syntactically identical. Floats are never identical to rational numbers.

Inputs: *i*, *j* — Indices of the top tags of expressions.

Outputs: Returns TRUE if the expressions indexed by *i* and *j* are syntactically identical.

Assumptions: None

Side Effects: None

Availability: All versions of the TI‑89 / TI‑92 Plus.

TI‑89 / TI‑92 Plus Differences: None

See Also: **compare_expressions, is_equivalent_to**

Example:

```
push_Float (3.0);
j = top_estack;
push_quantum_as_nonnegative_int (3u);
are_expressions_identical (top_estack, j);  /* Returns FALSE */
```

compare_expressions

Declaration: int **compare_expressions** (EStackIndex *i*, EStackIndex *j*)

Category(ies): Algebra Utilities

Description: Returns an int that is 0 if expressions indexed by *i* and *j* are equal in the sense that they have the same structure, variables, function names, and numbers that "compare equal." A float and a rational number "compare equal" if converting the rational number to a float produces an identical number. Otherwise returns a positive int if expression indexed by *i* is more main, or returns a negative int if the expression indexed by *i* is less main.

Glossing over details, variables are more main than symbolic constants such as π, which are more main than numbers. If the user enters an expression such as expand (. . . , var) or integral (. . . , var), then that variable is most main. Otherwise, the 26 Roman 1-letter variables order r>s> . . . >z>a>b> . . . >q, which order more main than all other variables, which order alphabetically. Functions and operators are typically ordered by recursively comparing their first arguments, with ties broken by comparing their second arguments, etc., then finally comparing the operators or functions, if necessary.

Examples:

i indexes	*j* indexes	Compare expression returns
-2.0	-2	0
-2.0	-1	-1
π	4	1
x	4	1
x	r	-1
x	ln(y)	1
x	ln(x)	-1

Inputs: *i*, *j* — Indices of the top tags of internally-simplified expressions.

Outputs: Returns an int that is 0 if expressions indexed by *i* and *j* are equal in the sense that they have the same structure, variables, function names, and numbers that "compare equal."

Assumptions: None

Side Effects: None

Availability: All versions of the TI-89 / TI-92 Plus.

(continued)

compare_expressions *(continued)*

TI-89 / TI-92 Plus Differences: None

See Also: **compare_numbers, compare_Floats, compare_complex_magnitudes**

Example:

```
Boolean does_term_have_base (EStackIndex term, EStackIndex base)
/* Returns TRUE if and only if term has base. */
{  Int s;
   while (! IS_NUMBER_TAG (ESTACK (term)))   /* Loop over factors of term */
      {  s = compare_expressions (lead_base_index (term), base);
         if (s)
            if (s < 0)
               return FALSE;  /* base > all remaining bases in term */
            else
               term = remaining_factors_index (term);
         else
            return TRUE;
      }
   return FALSE;
}
```

did_push_lincf

Declaration: Boolean **did_push_lincf** (EStackIndex *i*, EStackIndex *vi*)

Category(ies): Algebra Utilities

Description: Determines if the expression indexed by *i* is not linear in the variable indexed by *vi*.

Inputs: *i* — Index of the top tag of an internally-simplified expression.

vi — Index of the top tag of a variable.

Outputs: Returns FALSE if the expression indexed by *i* is not linear in the variable indexed by *vi*. Otherwise pushes the coefficient of *vi* (perhaps 0), then returns TRUE.

Assumptions: None

Side Effects: None

Availability: All versions of the TI‑89 / TI‑92 Plus.

TI‑89 / TI‑92 Plus Differences: None

See Also: **push_poly_deg_in_var_or_kernel, linear_degree, index_if_pushed_binomial_info, index_if_pushed_qquad_info**

Example: If *i* indexes (x^2 ∗ y + x + y) ∗ z and *vi* indexes y, then **did_push_lincf**(*i*, *vi*) pushes (x^2 + y) ∗ z and returns TRUE.

```
Boolean did_push_recip_lincf (EStackIndex i, EStackIndex vi)
/* If the expression indexed by i is linear in the variable indexed by vi,
   pushes the reciprocal of the linear coefficient then returns TRUE.
   Otherwise returns FALSE.
*/
{  if (did_push_lincf (i, vi))
      {  replace_top_with_reciprocal ();
         return TRUE;
      }
   return FALSE;
}
```

factor_base_index

Declaration: EStackIndex **factor_base_index** (EStackIndex *k*)

Category(ies): Algebra Utilities

Description: If *k* indexes an exponentiation tag, returns the index of the base. Otherwise returns *k*.

Inputs: *k* — Index of the top tag of an algebraic expression.

Outputs: If *k* indexes an exponentiation tag, returns the index of the base. Otherwise returns *k*.

Assumptions: None

Side Effects: None

Availability: All versions of the TI‑89 / TI‑92 Plus.

TI‑89 / TI‑92 Plus Differences: None

See Also: **factor_exponent_index**

Example:

```
EStackIndex lead_base_index (EStackIndex i)
/* Returns the index of the lead base of any algebraic expression indexed by i. */
{  return factor_base_index (lead_factor_index (i));
}
```

factor_exponent_index

Declaration: EStackIndex **factor_exponent_index** (EStackIndex *k*)

Category(ies): Algebra Utilities

Description: If *k* indexes an exponentiation tag, returns the index of the exponent. Otherwise, if IS_ARITH_APPROX is true, returns Float1Index. If IS_ARITH_APPROX is false, returns Integer1Index.

Inputs: *k* — Index of the top tag of an algebraic expression.

Outputs: If *k* indexes an exponentiation tag, returns the index of the exponent.

Assumptions: None

Side Effects: None

Availability: All versions of the TI‑89 / TI‑92 Plus.

TI‑89 / TI‑92 Plus Differences: None

See Also: **factor_base_index**

Example:

```
EStackIndex lead_exponent_index (EStackIndex i)
/* Returns index of the lead exponent of any expression indexed by i. */
{  return factor_exponent_index (lead_factor_index (i));
}
```

has_different_variable

Declaration: Boolean **has_different_variable** (EStackIndex *i*, EStackIndex *vi*, Boolean *ignore_func*)

Category(ies): Algebra Utilities

Description: *i* indexes an expression and *vi* indexes a variable or list thereof. Returns TRUE if the expression has another variable. Otherwise returns FALSE. If *ignore_func* is TRUE, user-function names are treated as variables.

Inputs:

i	—	Indexes the top tag of an internally-simplified algebraic expression.
vi	—	Indexes the top tag of a variable or of a list thereof.
ignore_func	—	TRUE if user-function names should be treated as variables.

Outputs: Returns TRUE if the expression has another variable. Otherwise returns FALSE.

Assumptions: None

Side Effects: None

Availability: All versions of the TI‑89 / TI‑92 Plus.

TI‑89 / TI‑92 Plus Differences: None

See Also: **main_gen_var_index, next_var_or_kernel_index, push_var_kern_tail**

Example: Returns *ignore_func* if *i* indexes x + f(x) and *vi* indexes x.

```
push_quantum (8u);
vi = top_estack;    /* vi becomes variable x */
push_quantum (9u);  /* Push variable y */
has_different_variable (top_estack, vi, FALSE);  /* Returns TRUE */
```

im_index

Declaration: EStackIndex **im_index** (EStackIndex *k*)

Category(ies): Algebra Utilities

Description: If *k* indexes an IM_RE_TAG, returns the index of the expression below it, which is the imaginary part for internally-simplified algebraic expressions. Otherwise returns the index of an integer or float 0, depending on the arithmetic mode.

Inputs: *k* — Indexes the top tag of an internally-simplified algebraic expression.

Outputs: If *k* indexes an IM_RE_TAG, returns the index of the expression below it, which is the imaginary part for internally-simplified algebraic expressions. Otherwise returns the index of an integer or float 0, depending on the arithmetic mode.

Assumptions: None

Side Effects: None

Availability: All versions of the TI-89 / TI-92 Plus.

TI-89 / TI-92 Plus Differences: None

See Also: **re_index, replace_top2_with_imre**

Example:

```
push_expression (Integer1Index);
real_part = top_estack;
push_expression (FloatPiIndex);
replace_top2_with_imre (real_part);
im_index (top_estack);  /* Returns the index of a float pi */
```

index_if_pushed_binomial_info

Declaration: EStackIndex **index_if_pushed_binomial_info** (EStackIndex *i*, EStackIndex *vi*)

Category(ies): Algebra Utilities

Description: If the algebraic expression indexed by *i* is a generalized binomial in the variable or kernel indexed by *vi*, pushes the constant term, then the coefficient of the nonconstant term, then the degree; then returns the index of the constant term. Otherwise returns NULL_INDEX. The degree can be any nonzero number, including negative and/or fractional.

Inputs: *i* — Index of the top tag of an internally-simplified algebraic expression.

vi — Index of the top tag of a variable or kernel.

Outputs: If the algebraic expression indexed by *i* is a generalized binomial in the variable or kernel indexed by *vi*, pushes the constant term, then the coefficient of the nonconstant term, then the degree; then returns the index of the constant term. Otherwise returns NULL_INDEX. The degree can be any nonzero number, including negative and/or fractional.

Assumptions: None

Side Effects: None

Availability: All versions of the TI-89 / TI-92 Plus.

TI-89 / TI-92 Plus Differences: None

See Also: **push_poly_deg_in_var_or_kernel, linear_degree, did_push_lincf, index_if_pushed_qquad_info**

Example: If *i* indexes (x^2 * ln(y)^-(1/2) + x + ln(y)^-(1/2)) * z
and *vi* indexes ln(y), pushes x * z, then (x^2 + 1) * z,
then -1/2, then returns TRUE.

```
/* Pushes a tagged integer 2 constant term, then a tagged float coefficient 0.5,
   then a tagged integer 1 degree, then returns the index of the tagged integer 2
   constant terms that it pushed.
*/
push_quantum (8u);
vi = top_estack;                      /* vi = variable x */
push_product (vi, FloatHalfIndex);    /* top_estack -> x * (1/2) */
add_to_top (Integer2Index);           /* top_estack -> x * (1/2) + 2 */
index_if_pushed_binomial_info (top_estack, vi);
```

index_if_pushed_qquad_info

Declaration: EStackIndex **index_if_pushed_qquad_info** (EStackIndex *i*, EStackIndex *x*, EStackIndex *h*)

Category(ies): Algebra Utilities

Description: a ∗ *x^(2h)* + b ∗ *x^h* + c is quasi-quadratic in *x*, with half-degree *h*.

h indexes a number, *x* indexes a variable, and *i* indexes a polynomial in that variable — generalized to allow non-negative fractional powers.

Determines whether or not the polynomial is a quasi-quadratic of half-degree *h* in *x*. If you do not know *h*, first use **push_poly_deg_in_var_or_kernel** (. . .).

For example: If *i* indexes u^3 ∗ (v^6 + v^3 ∗ w) + u + 1 and *x* indexes v and *h* indexes 3, then pushes u + 1, then u^3 ∗ w, then u^3, then returns the index of the pushed u^3 ∗ w.

Inputs:

i — Index of the top tag of an internally-simplified algebraic expression.

x — Index of the top tag of a variable.

h — Index of the top tag of a tagged number.

Outputs: Returns NULL_INDEX if the polynomial is not quasi-quadratic of half-degree *h* in *x*. Otherwise pushes onto the estack the constant, then middle, then lead coefficient; then returns the index of the middle coefficient.

Assumptions: None

Side Effects: None

Availability: All versions of the TI‑89 / TI‑92 Plus.

TI‑89 / TI‑92 Plus Differences: None

See Also: **push_poly_deg_in_var_or_kernel, linear_degree, did_push_lincf, index_if_pushed_binomial_info**

(continued)

index_if_pushed_qquad_info *(continued)*

Example:

```
EStackIndex index_var_if_pushed_qquad_info (EStackIndex i)
/* Returns NULL_INDEX if the expression indexed by i is not quasi-quadratic
   in any if its variables.
   Otherwise for the most main such variable, pushes onto the estack the
   HALF degree, then the constant, then middle, then lead coefficient;
   then returns an index to the variable.
*/
{  Access_AMS_Global_Variables;
   EStackIndex j,
               vq = main_gen_var_index (i),
               old_top = top_estack;
   while (vq)
   {  if (is_variable (vq))
      {  push_reciprocal_of_quantum (2u);
         j = top_estack;
         push_poly_deg_in_var_or_kernel (i, vq);
         replace_top2_with_prod (j);
         if (index_if_pushed_qquad_info (i, vq, top_estack))
            return vq;
         top_estack = old_top;
      }
      vq = next_var_or_kernel_index (i, vq);
   }
   return NULL_INDEX;
}
```

index_numeric_term

Declaration: EStackIndex **index_numeric_term** (EStackIndex *i*)

Category(ies): Algebra Utilities

Description: *i* indexes an internally-simplified algebraic expression. Internally-simplified sums and differences have at most one term with a numeric tag, in which case it is the deepest term. Returns the index of this term if there is one. Otherwise, returns Float0Index if IS_ARITH_APPROX is true and Integer0Index if IS_ARITH_APPROX is false.

Note that Float0Index and Integer0Index are not ordinarily physically within the expression indexed by *i*.

Inputs: *i* — Index of the top tag of an internally-simplified algebraic expression.

Outputs: Returns the index of the numeric term if there is one. Otherwise, returns Float0Index if IS_ARITH_APPROX is true and Integer0Index if IS_ARITH_APPROX is false.

Assumptions: None

Side Effects: None

Availability: All versions of the TI‑89 / TI‑92 Plus.

TI‑89 / TI‑92 Plus Differences: None

See Also: **push_constant_terms**

(continued)

index_numeric_term *(continued)*

Example:

If i indexes the internally-simplified expression
`x^2 + 7 + 1 + x`
then **index_numeric_term**(i) returns index(8).

If i indexes the internally-simplified expression
`x^2 + x`
then **index_numeric_term**(i) returns Float0Index if IS_ARITH_APPROX is true and Integer0Index if IS_ARITH_APPROX is false.

If i indexes the internally-simplified expression
`x^2 + π`
then **index_numeric_term**(i) returns Float0Index if IS_ARITH_APPROX is true and Integer0Index if IS_ARITH_APPROX is false.

```
Boolean sum_has_modest_numeric_term (EStackIndex i)
/* i indexes a sum.
   Returns TRUE if it does not have a top-level numeric term or if its
   magnitude is < MAX_EXP_ARG. Otherwise returns FALSE.
*/
{  i = index_numeric_term (i);
   return NULL_INDEX == i ||
          FABS (estack_number_to_Float (i)) < MAX_EXP_ARG;
}
```

index_of_lead_base_of_lead_term

Declaration: EStackIndex **index_of_lead_base_of_lead_term** (EStackIndex *i*)

Category(ies): Algebra Utilities

Description: Returns the index of the lead base of the lead term of any algebraic expression indexed by *i*.

Inputs: *i* — Index of the top tag of an algebraic expression.

Outputs: Returns the index of the lead base of the lead term of any algebraic expression indexed by *i*.

Assumptions: None

Side Effects: None

Availability: On AMS 1.05 and higher.

TI-89 / TI-92 Plus Differences: None

See Also: **factor_base_index, lead_base_index, lead_term_index**

Example:

```
push_quantum_as_nonnegative_int (2u);
exponent = top_estack;
push_quantum (8u);    /* Push variable x */
replace_top2_with_pow (exponent);
xsq = top_estack;    /* top_estack -> x^2 */
push_quantum (9u);   /* Push variable y */
replace_top2_with_prod (xsq);  /* top_estack -> x^2 * y  */
index_of_lead_base_of_lead_term (top_estack);
   /* : Returns index of variable x */
add1_to_top (top_estack);      /* top_estack -> x^2 * y + 1  */
index_of_lead_base_of_lead_term (top_estack);
   /* : Returns index of variable x */
```

index_reductum_with_tag_base

Declaration: EStackIndex **index_reductum_with_tag_base** (EStackIndex *i*, Quantum *tag*, Boolean *exponent_must_be_1*)

Category(ies): Algebra Utilities

Description: Returns *i* or the index of the first reductum of the expression indexed by *i* whose lead term has a base beginning with *tag*. Returns NULL_INDEX if there is no such term. If *exponent_must_be_1* is TRUE then the base's exponent must be 1. You can use this function together with **push_but_term** to push a "sum" of all but the first syntactic term containing a base beginning with *tag* in the expression indexed by *i*.

Inputs:

i	—	Index of the top tag of an internally-simplified algebraic expression.
tag	—	A primary tag.
exponent_must_be_1	—	FALSE if the base can have an exponent ≠ 1.

Outputs: Returns *i* or the index of the first reductum of the expression indexed by *i* whose lead term has a base beginning with *tag*. Returns NULL_INDEX if there is no such term.

Assumptions: None

Side Effects: None

Availability: On AMS 2.02 and higher.

TI-89 / TI-92 Plus Differences: None

See Also: **push_but_term, index_numeric_term, push_constant_terms, push_nonconstant_terms, push_dependent_terms, push_independent_terms**

Example:

```
Boolean does_denom_have_sin2 (EStackIndex i)
/* Returns TRUE if the term indexed by i has a SIN2 sufficiently exposed in its
   denominator to warrant putting that term + another term over a common denominator.
*/
{  EStackIndex j;
   while (! IS_NUMBER_TAG (ESTACK (i)))
   {  j = lead_factor_index (i);
      if (EXPONENTIATION_TAG == ESTACK (j) &&
         is_negative(next_expression_index(--j)) &&
         index_reductum_with_tag_base (j, SIN2_TAG, FALSE) )
         return TRUE;
      i = remaining_factors_index (i);
   }
   return FALSE;
}
```

index_rmng_factor

Declaration: EStackIndex **index_rmng_factor** (EStackIndex *j*, EStackIndex *i*)

Category(ies): Algebra Utilities

Description: This function can be used to determine if the expression indexed by *j* is identical to the one indexed by *i* or any of its syntactic factors. If so, you can then use **push_but_factor** to form the cofactor of that factor.

Inputs: *i*, *j* — Index of the top tags of internally-simplified algebraic expressions.

Outputs: Returns NULL_INDEX if the expression indexed by *j* is not identical to the one indexed by **lead_factor_index**(*i*) or **lead_factor_index** (**remaining_factors_index**(*i*)) or **lead_factor_index** (**remaining_factors_index** (**remaining_factors_index**(*i*))), etc.

Otherwise returns the shallowest index *k* such that the expression indexed by *j* is identical to the one indexed by **lead_factor_index**(*k* = *i*) or **lead_factor_index** (*k* = **remaining_factors_index**(*i*)) or **lead_factor_index** (*k* = **remaining_factors_index** (**remaining_factors_index**(*i*))), etc.

Assumptions: None

Side Effects: None

Availability: All versions of the TI-89 / TI-92 Plus.

TI-89 / TI-92 Plus Differences: None

See Also: **index_rmng_fctrs_start_fctr_tag, index_rmng_fctrs_start_base, index_rmng_fctrs_start_base_tag** (*Beware: Their arguments are in the opposite order!*)

Example:

```
push_quantum (10u);
push_quantum (EXP_TAG);   /* exp(z) */
push_quantum (9u);
push_quantum (EXP_TAG);   /* exp(y) */
push_quantum (MULTIPLY_TAG);
push_quantum (8u);
push_quantum (LN_TAG);    /* ln(x) */
push_quantum (MULTIPLY_TAG);
i = top_estack;  /* ln(x) * exp(y) * exp(z)  */
push_quantum (9u);
push_quantum (EXP_TAG);
j = top_estack;  /* exp(y) */
k = index_rmng_factor (j,i);
if (NULL_INDEX != k) push_but_factor(i,k);  /* pushes ln(x) * exp(z) */
```

index_rmng_fctrs_start_base

Declaration: EStackIndex **index_rmng_fctrs_start_base** (EStackIndex *i*, EStackIndex *j*)

Category(ies): Algebra Utilities

Description: This function can be used to determine if the expression indexed by *j* or any power of that expression is identical to *i* or any of its syntactic factors. If so, use **push_but_factor** to form its cofactor.

Inputs: *i*, *j* — Index of the top tags of internally-simplified algebraic expressions.

Outputs: Returns NULL_INDEX if the expression indexed by *j* is not identical to the one indexed by **lead_base_index**(*i*) or **lead_base_index** (**remaining_factors_index**(*i*)) or **lead_base_index** (**remaining_factors_index** (**remaining_factors_index**(*i*))), etc.

Otherwise returns the shallowest index *k* such that the expression indexed by *j* is identical to the one indexed by **lead_base_index**(*k* = *i*) or **lead_base_index** (*k* = **remaining_factors_index**(*i*)) or **lead_base_index** (*k* = **remaining_factors_index** (**remaining_factors_index**(*i*)), etc.

Assumptions: None

Side Effects: None

Availability: All versions of the TI-89 / TI-92 Plus.

TI-89 / TI-92 Plus Differences: None

See Also: **index_rmng_fctrs_start_fctr_tag, index_rmng_fctrs_start_base_tag, index_rmng_factor**

Example:

```
push_quantum (10u);
push_quantum (EXP_TAG);  /* exp(z) */
push_quantum_as_nonnegative_int (2u);
push_quantum (9u);
push_quantum (EXP_TAG);
push_quantum (EXPONENTIATION_TAG);  /* exp(y)^2 */
push_quantum (MULTIPLY_TAG);
push_quantum (8u);
push_quantum (LN_TAG);  /* ln(x) */
push_quantum (MULTIPLY_TAG);
i = top_estack;  /* ln(x) * exp(y) * exp(z) */
push_quantum (9u);
push_quantum (EXP_TAG);
j = top_estack;  /* exp(y) */
k = index_rmng_fctrs_start_base (i,j);
if (NULL_INDEX != k) push_but_factor(i,k);  /* pushes ln(x) * exp(z) */
```

index_rmng_fctrs_start_base_tag

Declaration: EStackIndex **index_rmng_fctrs_start_base_tag** (EStackIndex *i*, Quantum *q*)

Category(ies): Algebra Utilities

Description: This function can be used together with **push_but_factor** to push a product of all but the first syntactic factor having base tag *q* in expression *i*.

Inputs: *i* — Index of the top tag of an internally-simplified algebraic expression.

q — Primary tag.

Outputs: Returns NULL_INDEX if ESTACK (**lead_base_index**(*i*)) != *q* and ESTACK (**lead_base_index** (**remaining_factors_index**(*i*))) != *q* and ESTACK (**lead_base_index** (**remaining_factors_index** (**remaining_factors_index**(*i*)))) != *q*, etc.

Otherwise returns the shallowest *k* such that ESTACK (**lead_base_index**(*k*)) == *q* and (*k* = *i* or *k* = **remaining_base_index**(*i*) or *k* = **remaining_base_index** (**remaining_factors_index**(*i*)) or . . .).

Assumptions: None

Side Effects: None

Availability: All versions of the TI-89 / TI-92 Plus.

TI-89 / TI-92 Plus Differences: None

See Also: **rmng_fctrs_start_fctr_tag, index_rmng_fctrs_start_base, index_rmng_factor**

Example:

```
push_quantum (10u);
push_quantum (EXP_TAG);             /* exp(z) */
push_quantum_as_nonnegative_int (2u);
push_quantum (9u);
push_quantum (EXP_TAG);
push_quantum (EXPONENTIATION_TAG);  /* exp(y)^2 */
push_quantum (MULTIPLY_TAG);
k = top_estack;
push_quantum (8u);
push_quantum (LN_TAG);              /* ln(x) */
push_quantum (MULTIPLY_TAG);
i = top_estack;                     /* ln(x) * exp(y)^2 * exp(z) */
index_rmng_fctrs_start_base_tag (top_estack, EXP_TAG);   /* Returns k */
index_rmng_fctrs_start_base_tag (top_estack, LN_TAG);    /* Returns i */
index_rmng_fctrs_start_base_tag(top_estack,ABS_TAG);     /* Returns NULL_INDEX */
```

index_rmng_fctrs_start_fctr_tag

Declaration: EStackIndex **index_rmng_fctrs_start_fctr_tag** (EStackIndex *i*, Quantum *q*)

Category(ies): Algebra Utilities

Description: This function can be used together with **push_but_factor** to push a product of all but the first syntactic factor having tag *q* in expression *i*.

Inputs: *i* — Index of the top tag of an internally-simplified algebraic expression.

q — Primary tag.

Outputs: Returns NULL_INDEX if ESTACK (**lead_factor_index**(*i*)) != *q* and ESTACK (**lead_factor_index** (**remaining_factors_index**(*i*))) != *q* and ESTACK (**lead_factor_index** (**remaining_factors_index** (**remaining_factors_index**(*i*)))) != *q*, etc.

Otherwise returns the shallowest *k* such that ESTACK (**lead_factor_index**(*k*)) == *q* and (*k* = *i* or *k* = **remaining_factors_index**(*i*) or *k* = **remaining_factors_index** (**remaining_factors_index**(*i*)) or . . .).

Assumptions: None

Side Effects: None

Availability: All versions of the TI-89 / TI-92 Plus.

TI-89 / TI-92 Plus Differences: None

See Also: **index_rmng_fctrs_start_base_tag, index_rmng_fctrs_start_base, index_rmng_factor**

Example:

```
push_quantum (10u);
push_quantum (EXP_TAG);          /* exp(z) */
push_quantum (9u);
push_quantum (EXP_TAG);          /* exp(y) */
push_quantum (MULTIPLY_TAG);
k = top_estack;                  /* exp(y) * exp(z) */
push_quantum (8u);
push_quantum (LN_TAG);           /* ln(x) */
push_quantum (MULTIPLY_TAG);
i = top_estack;                  /* ln(x) * exp(y) * exp(z) */
index_rmng_fctrs_start_fctr_tag (top_estack, EXP_TAG);   /* Returns k */
index_rmng_fctrs_start_fctr_tag (top_estack, LN_TAG);    /* Returns i */
index_rmng_fctrs_start_fctr_tag(top_estack,ABS_TAG);     /* Returns NULL_INDEX  */
```

is_free_of_tag

Declaration: Boolean **is_free_of_tag** (EStackIndex *i*, Quantum *q*)

Category(ies): Algebra Utilities

Description: Determines whether the expression indexed by *i* contains tag *q*.

Inputs: *i* — Index of the top tag of an expression.

q — A primary tag.

Outputs: Returns TRUE if the expression indexed by *i* does not contain tag *q*. Otherwise returns FALSE.

Assumptions: None

Side Effects: None

Availability: All versions of the TI‑89 / TI‑92 Plus.

TI‑89 / TI‑92 Plus Differences: None

See Also: **is_independent_of, is_independent_of_tail, is_tail_independent_of**

Example:

```
/*
** Title: push_radians
** Description: Pushes the result of converting the specified value
**              (interpreted as radians) to the currently selected measure.
** Input: estack index i of the radian value
** Output: Pushes the value converted to the currently selected measure.
*/
void push_radians (EStackIndex i)
{  if (! is_free_of_tag (i, IM_RE_TAG))
      ER_THROW (ER_DOMAIN);
   if (IS_RADIANS)
      push_expression (i);
   else
   {  push_pi_on_quantum (180);
      push_ratio (i, top_estack);
      delete_expression (next_expression_index (top_estack));
   }
}
```

is_independent_of

Declaration: Boolean **is_independent_of** (EStackIndex *i*, EStackIndex *j*)

Category(ies): Algebra Utilities

Description: Determines whether the expression indexed by *i* is syntactically independent of the expression indexed by *j*.

Inputs: *i*, *j* — Indices of the top tags of internally-simplified expressions.

Outputs: Returns TRUE if the expression indexed by *i* is syntactically independent of the expression indexed by *j*. Otherwise returns FALSE.

Assumptions: None

Side Effects: None

Availability: All versions of the TI‑89 / TI‑92 Plus.

TI‑89 / TI‑92 Plus Differences: None

See Also: **is_free_of_tag, is_independent_of_tail, is_tail_independent_of**

Example:

```
Boolean is_tail_independent_of (EStackIndex i, EStackIndex j)
{  for (;;)
   {  if (END_TAG == ESTACK (i))
         return TRUE;
      else if (is_independent_of (i, j))
         i = next_expression_index (i);
      else
         return FALSE;
   }
}
```

is_independent_of_tail

Declaration: Boolean **is_independent_of_tail** (EStackIndex *i*, EStackIndex *j*)

Category(ies): Algebra Utilities, Lists and Matrices

Description: Determines whether the expression indexed by *i* is syntactically independent of all the elements in the tail indexed by *j*.

Inputs: *i* — Index of the top tag of an internally-simplified expression.

j — Index of a tail of internally-simplified expressions.

Outputs: Returns TRUE if the expression indexed by *i* is syntactically independent of all the elements in the tail indexed by *j*. Otherwise returns FALSE.

Assumptions: None

Side Effects: None

Availability: All versions of the TI‑89 / TI‑92 Plus.

TI‑89 / TI‑92 Plus Differences: None

See Also: **is_free_of_tag, is_independent_of, is_tail_independent_of**

Example:

```
Boolean is_independent_of_elements (EStackIndex i, EStackIndex j)
/* Returns TRUE if the expression indexed by i is syntactically independent
   of all the elements in the list indexed by j.
*/
{  return is_independent_of_tail (i, j - 1u);
}
```

is_neg_lead_numr_coef_re_part

Declaration: Boolean **is_neg_lead_numr_coef_re_part** (EStackIndex *i*)

Category(ies): Algebra Utilities

Description: Determines whether the lead numeric coefficient of the real part of the internally-simplified algebraic expression indexed by *i* is negative. This function is useful for exploiting symmetry (f(⁻z) -> f(z)) and antisymmetry (f(⁻z) -> ⁻f(z)).

Inputs: *i* — Index of the top tag of an internally-simplified expression.

Outputs: Returns TRUE if the lead numeric coefficient of the real part of the internally-simplified algebraic expression indexed by *i* is negative. Otherwise returns FALSE.

Assumptions: None

Side Effects: None

Availability: All versions of the TI‑89 / TI‑92 Plus.

TI‑89 / TI‑92 Plus Differences: None

See Also: **is_negative**

(continued)

is_neg_lead_numr_coef_re_part *(continued)*

Example:

```
void push_integer_part (EStackIndex i)
{  Access_AMS_Global_Variables;
   EStackIndex j, old_top = top_estack;
   if (LIST_TAG == ESTACK (i))
   {  map_tail (push_integer_part, i - 1u);
      push_quantum (LIST_TAG);
   }
   else if (is_nonnegative (i))  push_floor (i);
   else if (is_nonpositive (i))  push_ceiling (i);
   else if (integer_non_unknown (i) > 0)  push_expression (i);
   else if (is_neg_lead_numr_coef_re_part(i)) /* iPart(-x) -> -iPart(x) */
   {  push_negate (i);
      i = top_estack;
      push_integer_part (i);
      delete_between (old_top, i);
      negate_top ();
   }
   else if (IM_RE_TAG == ESTACK (i))
   {  push_integer_part (next_expression_index (--i));
      j = top_estack;
      push_integer_part (i);
      replace_top2_with_imre (j);
   }
   else
   {  push_expression (i);
      push_quantum (INT_PART_TAG);
   }
} /* end push_integer_part */
```

is_polynomial_in_var_or_kern

Declaration: Boolean **is_polynomial_in_var_or_kern** (EStackIndex *i*, EStackIndex *vi*)

Category(ies): Algebra Utilities

Description: Determines whether the expression indexed by *i* is polynomial in the variable or kernel indexed by *vi*, generalized to permit negative and fractional powers of *vi*.

Inputs: *i* — Index of the top tag of an internally-simplified expression.

vi — Index of the top tag of an internally-simplified variable or kernel.

Outputs: Returns TRUE only if the expression indexed by *i* is polynomial in the variable or kernel indexed by *vi*. Otherwise returns FALSE.

Assumptions: None

Side Effects: None

Availability: All versions of the TI‑89 / TI‑92 Plus.

TI‑89 / TI‑92 Plus Differences: None

See Also: **is_totally_polynomial**

Example: If *i* indexes x^-(3/2) + ln(y), then **is_polynomial_in_var_or_kern** returns TRUE if *vi* indexes x or ln(y) or z, but returns FALSE if *vi* indexes y.

```
Boolean is_polynomial_in_var_or_kern (EStackIndex i, EStackIndex vi)
{  for (;;)
   {  if (are_expressions_identical (i, vi) ||
          IS_NUM_VAR_OR_ZERO_ARG_TAG (ESTACK (i)))
         return TRUE;
      if (EXPONENTIATION_TAG == ESTACK (i) &&
          ! (are_expressions_identical (vi, POWER_BASE_INDEX (i)) ||
             is_whole_number (POWER_EXPONENT_INDEX (i)) ||
             is_independent_of (POWER_BASE_INDEX (i), vi) ) )
         return FALSE;
      if (IS_ARITH_OR_POWER_TAG (ESTACK (i)) &&
          is_polynomial_in_var_or_kern (i - 1u, vi) )
         i = next_expression_index (i - 1u);
      else return is_independent_of (i, vi);
   }
}
```

is_tail_independent_of

Declaration: Boolean **is_tail_independent_of** (EStackIndex *i*, EStackIndex *j*)

Category(ies): Algebra Utilities, Lists and Matrices

Description: Determines whether all of the elements of the tail indexed by *i* are syntactically independent of the expression indexed by *j*.

Inputs: *i* — Index of the top tag of a tail of internally-simplified expressions.

j — Index of the top tag of an internally-simplified expression.

Outputs: Returns TRUE if all of the elements of the tail indexed by *i* are syntactically independent of the expression indexed by *j*. Otherwise returns FALSE.

Assumptions: None

Side Effects: None

Availability: All versions of the TI‑89 / TI‑92 Plus.

TI‑89 / TI‑92 Plus Differences: None

See Also: **is_independent_of, is_independent_of_tail**

Example:

```
push_quantum (END_TAG);
push_quantum (8u);   \* push variable x *\
push_expression (Integer1Index);
is_tail_independent_of (top_estack, IntegerMinus1Index);   \* returns TRUE *\
```

is_term_improper

Declaration: Boolean **is_term_improper** (EStackIndex *i*)

Category(ies): Algebra Utilities, Lists and Matrices

Description: Determines whether the algebraic expression indexed by *i* is a nonsum and if the degree of the expanded numerator would be at least as large as that of the expanded denominator in the main variable of the expression.

Inputs: *i* — Index of the top tag of an internally-simplified algebraic expression.

Outputs: Returns TRUE if the algebraic expression indexed by *i* is a nonsum and if the degree of the expanded numerator would be at least as large as that of the expanded denominator in the main variable of the expression. Otherwise returns FALSE.

Assumptions: None

Side Effects: None

Availability: All versions of the TI‑89 / TI‑92 Plus.

TI‑89 / TI‑92 Plus Differences: None

See Also: **push_make_proper, push_numerator, push_denominator, index_main_var, push_poly_deg_in_var_or_kern, push_poly_qr**

Example:

```
push_quantum (8u);
numerator = top_estack;                    /* Push variable x */
push_sum (numerator, Integer1Index);       /* Push x + 1 */
replace_top2_with_ratio (numerator);       /* top_estack -> x/(x + 1) */
is_term_improper (top_estack);             /* returns TRUE */
```

is_totally_polynomial

Declaration: Boolean **is_totally_polynomial** (EStackIndex *i*)

Category(ies): Algebra Utilities

Description: Determines whether the expression indexed by *i* is polynomial in all of its variables, generalized to permit non-negative fractional powers of variables.

Inputs: *i* — Index of the top tag of an internally-simplified algebraic expression.

Outputs: Returns TRUE if the expression indexed by *i* is polynomial in all of its variables, generalized to permit non-negative fractional powers of variables. Otherwise returns FALSE.

Assumptions: None

Side Effects: None

Availability: All versions of the TI‑89 / TI‑92 Plus.

TI‑89 / TI‑92 Plus Differences: None

See Also: **is_polynomial_in_var_or_kern**

Example: Returns TRUE if *i* indexes (x^(1/2) + y) ∗ x.
Returns TRUE if *i* indexes π + ln(2).
Returns FALSE if *i* indexes x^-1 + 2.
Returns FALSE if *i* indexes ln(x).

```
Boolean is_totally_polynomial (EStackIndex i)
{  Quantum q;
   for (;;)
   {  if (is_variable (i) || is_constant (i)) return TRUE;
      if (EXPONENTIATION_TAG == (q = ESTACK (i)))
      {  q = ESTACK (next_expression_index (--i));
         if (is_variable (i))
             return NONNEGATIVE_INTEGER_TAG == q ||
                    POSITIVE_FRACTION_TAG == q;
         if (NONNEGATIVE_INTEGER_TAG != q) return FALSE;
      }
      else if ((ADD_TAG == q || MULTIPLY_TAG == q) &&
               is_totally_polynomial (--i) )
         i = next_expression_index (i);
      else return FALSE;
   }
}
```

lead_base_index

Declaration: EStackIndex **lead_base_index** (EStackIndex *i*)

Category(ies): Algebra Utilities

Description: Returns **factor_base_index** (**lead_factor_index** (*i*)).

Inputs: *i* — Index of the top tag of an internally-simplified algebraic expression.

Outputs: Returns **factor_base_index** (**lead_factor_index** (*i*)).

Assumptions: None

Side Effects: None

Availability: On AMS 1.05 and higher.

TI-89 / TI-92 Plus Differences: None

See Also: **factor_base_index, index_of_lead_base_of_lead_term**

Example:

```
Boolean has_unit_base (EStackIndex i)
/* Returns TRUE if at least one of the top_level factors has a physical unit or
   physical constant as a base. Otherwise returns FALSE.
*/
{  while (! IS_NUMBER_TAG (ESTACK (i)))
   {  if (IS_UNIT (lead_base_index (i)))
            return TRUE;
      i = remaining_factors_index (i);
   }
   return FALSE;
}
```

lead_factor_index

Declaration: EStackIndex **lead_factor_index** (EStackIndex *i*)

Category(ies): Algebra Utilities

Description: If *i* indexes a MULTIPLY_TAG, returns the index of the shallower of its two operands. Otherwise returns *i*. Internally-simplified products and ratios have the most main factor shallowest, with less main factors deeper. Also, the lead factor of an internally-simplified product is never a product.

For example:

If i indexes the internally-simplified expression
`(3 * x^2) * y`
then **lead_factor_index**(i) returns index(x^2).

If i indexes the internally-simplified expression
`x^2`
then **lead_factor_index**(i) returns index(x^2).

Internally-simplified numeric denominator factors are combined with numeric numerator factors into a fractional numeric factor.

For example:

If i indexes the internally-simplified expression
`3/2`
then **lead_factor_index**(i) returns index(3/2).

Non-numeric denominator factors are internally simplified to be merged with numerator factors as negative powers.

For example:

If i indexes the internally-simplified expression
`2/x`
then **lead_factor_index**(i) returns index(x^-1).

A factor having a sum as its base orders shallower than a factor having the sum's main variable as its base.

For example:

If i indexes the internally-simplified expression
`(x + 1)^-2 * x^3`
then **lead_factor_index** returns index((x + 1)^-2).

Inputs: *i* — Index of the top tag of an internally-simplified algebraic expression.

Outputs: If *i* indexes a MULTIPLY_TAG, returns the index of the shallower of its two operands. Otherwise returns *i*. Internally-simplified products and ratios have the most main factor shallowest, with less main factors deeper. Also, the lead factor of an internally-simplified product is never a product.

(continued)

lead_factor_index *(continued)*

Assumptions: None

Side Effects: None

Availability: On AMS 1.05 and higher.

TI-89 / TI-92 Plus Differences: None

See Also: **remaining_factors_index, lead_term_index, reductum_index**

Example:

```
EStackIndex lead_base_index (EStackIndex i)
/* Returns index of the lead base of any expression indexed by i. */
{  return factor_base_index (lead_factor_index (i));
}
```

lead_term_index

Declaration: EStackIndex **lead_term_index** (EStackIndex *i*)

Category(ies): Algebra Utilities

Description: If *i* indexes an ADD_TAG, returns the index of the shallower of its two operands. Otherwise returns *i*.

Internally-simplified sums and differences have the most main term shallowest, with less main terms deeper. Also, the lead term of an internally-simplified sum is never a sum.

For example:

If i indexes the internally-simplified expression
`(2 + x) + y`
then **lead_term_index**(i) returns index(x).

If i indexes the internally-simplified expression
`x^2 * y`
then **lead_term_index**(i) returns index(x^2 ∗ y).

For the default mode IS_RECURSIVE, similar powers of the main variable are collected in internally-simplified expressions.

For example:

If i indexes the internally-simplified expression
`x^2 * y + x^2 + 5`
then **lead_term_index**(i) returns index(x^2 ∗ (y + 1)).

Inputs: *i* — Index of the top tag of an internally-simplified algebraic expression.

Outputs: If *i* indexes an ADD_TAG, returns the index of the shallower of its two operands. Otherwise returns *i*.

Assumptions: None

Side Effects: None

Availability: All versions of the TI‑89 / TI‑92 Plus.

TI‑89 / TI‑92 Plus Differences: None

See Also: **reductum_index, lead_factor_index, remaining_factors_index**

(continued)

lead_term_index *(continued)*

Example:

```
Boolean has_constant_term (EStackIndex i)
/* Returns TRUE if the expression indexed by i has a nonzero top-level
   constant term.  Otherwise returns FALSE.
*/
{  while (! is0 (i))
      if (is_constant (lead_term_index (i)))
         return TRUE;
      else
         i = reductum_index (i);
   return FALSE;
}  /* end has_constant_term */
```

linear_degree

Declaration: EStackIndex **linear_degree** (EStackIndex *i*, EStackIndex *vi*)

Category(ies): Algebra Utilities

Description: Returns 0 if the expression indexed by *i* is independent of the variable or kernel indexed by *vi*, or returns 1 if the expression is obviously linear in *vi*. Otherwise returns -1.

Inputs: *i* — Index of the top tag of an internally-simplified algebraic expression.

vi — Index of the top tag of an internally-simplified variable or kernel.

Outputs: Returns 0 if the expression indexed by *i* is independent of the variable or kernel indexed by *vi*, or returns 1 if the expression is obviously linear in *vi*. Otherwise returns -1.

Assumptions: None

Side Effects: None

Availability: All versions of the TI-89 / TI-92 Plus.

TI-89 / TI-92 Plus Differences: None

See Also: **did_push_lincf, push_poly_deg_in_var_or_kernel, index_if_pushed_binomial_info, index_if_pushed_qquad_info**

Example: If *i* indexes (x^2 * ln(y) + x + ln(y)) * z and *vi* indexes ln(y), then **linear_degree**(*i*, *vi*) returns 1.

```
EStackIndex is_linear_any (EStackIndex i)
/* If the expression indexed by i is obviously linear in any of its generalized
   variables, returns an index of the most main such variable.
   Otherwise returns NULL_INDEX.
*/
{  EStackIndex vi = main_gen_var_index (i);
   for (;;) if (vi)
               if (linear_degree (i, vi) == 1)
                  return vi;
               else
                  vi = next_var_or_kernel_index (i, vi);
            else
               return NULL_INDEX;
}
```

main_gen_var_index

Declaration: EStackIndex **main_gen_var_index** (EStackIndex *i*)

Category(ies): Algebra Utilities

Description: Returns the main generalized variable of any internally-simplified algebraic expression indexed by *i*.

A generalized variable can be a variable, the base of a noninteger power, or a kernel, meaning any other irrational subexpression.

More specifically, the rules are:
main_gen_var (sum) -> **main_gen_var** (leadTerm)
main_gen_var (product) -> **main_gen_var** (leadFactor)
main_gen_var (integerPower) -> **main_gen_var** (base)

Otherwise **main_gen_var** (expression) -> expression.

Inputs: *i* — Index of the top tag of an internally-simplified algebraic expression.

Outputs: Returns the main generalized variable of any internally-simplified algebraic expression indexed by *i*.

Assumptions: None

Side Effects: None

Availability: All versions of the TI-89 / TI-92 Plus.

TI-89 / TI-92 Plus Differences: None

See Also: **next_var_or_kernel_index**

Example: `main_gen_var (sin(x)^2 * y + ln(z))`. Returns sin(x).
`main_gen_var (3^(1/5) + 2)`. Returns 3.
`main_gen_var ((x + y)^(1/2) + x)`. Returns x + y.

```
void push_var_kern_tail (EStackIndex i)
/* i indexes an expression. Pushes onto the estack an END_TAG terminated list
   of its generalized variables, with the most main deepest.
*/
{  EStackIndex vi = main_gen_var_index (i);
   push_quantum (END_TAG);
   while (vi)  {  push_expression (vi);
                  vi = next_var_or_kernel_index (i, vi);
               }
}
```

map_unary_over_comparison

Declaration: void **map_unary_over_comparison** (void (* *proc1*) (EStackIndex), EStackIndex *comparison*);

Category(ies): Algebra Utilities

Description: Applies *proc1* to the deeper argument of *comparison*, then the shallower argument of *comparison*, then pushes the top tag of *comparison*.

Beware that without warning this function might lose solutions (such as by squaring both sides of an inequality) or introduce spurious solutions (such as by squaring both sides of an equation).

Inputs:

proc1 — Address of a procedure that takes one EStackIndex argument, pushes an expression onto the estack, then returns nothing.

comparison — Index of the top tag of a simplified equation or inequality.

Outputs: None

Assumptions: None

Side Effects: None

Availability: All versions of the TI‑89 / TI‑92 Plus.

TI‑89 / TI‑92 Plus Differences: None

See Also: **map_tail, all_tail, any_tail**

Example: If *comparison* indexes n>3, then `map_unary_over_comparison (&push_factorial, comparison)` pushes the comparison n! > 6.

```
push_quantum_as_nonnegative_int (3u);
push_quantum (8u);        /* push variable x */
push_quantum (EQUATION_TAG);
map_unary_over_comparison (&push_factorial, top_estack);
   /* Pushes x != 6 onto the estack. */
```

next_var_or_kernel_index

Declaration: EStackIndex **next_var_or_kernel_index** (EStackIndex *i*, EStackIndex *vi*)

Category(ies): Algebra Utilities

Description: Returns **main_gen_var_index**(*i*) if *vi* = NULL_INDEX. Otherwise returns an index of the variable or kernel in expression *i* that is next-most-main to *vi*, or returns NULL_INDEX if there is no such variable or kernel.

Inputs:

i — Indexes the top tag of an internally-simplified algebraic expression.

vi — Indexes NULL_INDEX or the top tag of an internally-simplified variable or kernel.

Outputs: Returns **main_gen_var_index**(*i*) if *vi* = NULL_INDEX. Otherwise returns an index of the variable or kernel in expression *i* that is next-most-main to *vi*, or returns NULL_INDEX if there is no such variable or kernel.

Assumptions: None

Side Effects: None

Availability: All versions of the TI‑89 / TI‑92 Plus.

TI‑89 / TI‑92 Plus Differences: None

See Also: **main_gen_var_index**

Example:

```
void push_var_kern_tail (EStackIndex i)
/* i indexes an expression. Pushes onto the estack an END_TAG terminated
   tail of its variables, with the most main deepest.
*/
{  EStackIndex vi = main_gen_var_index (i);
   push_quantum (END_TAG);
   while (vi)
   {  push_expression (vi);
      vi = next_var_or_kernel_index (i, vi);
   }
}
```

numeric_factor_index

Declaration: EStackIndex **numeric_factor_index** (EStackIndex *i*)

Category(ies): Algebra Utilities

Description: Internally-simplified products have at most one factor with a numeric tag, in which case it is the deepest factor.

Returns the index of this syntactic factor if there is one. Otherwise returns Float1Index if IS_ARITH_APPROX is true and Integer1Index if IS_ARITH_APPROX is false.

Numeric factors in numerators and denominators are simplified into a single numeric factor.

For example:

If i indexes the internally-simplified expression
`6 * x/(4 * y)`
then **numeric_factor_index**(i) returns index(3/2).

Inputs: *i* — Indexes the top tag of an internally-simplified algebraic expression.

Outputs: Returns the index of this syntactic factor if there is one. Otherwise returns Float1Index if IS_ARITH_APPROX is true and Integer1Index if IS_ARITH_APPROX is false.

Assumptions: None

Side Effects: None

Availability: All versions of the TI-89 / TI-92 Plus.

TI-89 / TI-92 Plus Differences: None

See Also: **push_constant_factors, index_numeric_term, push_constant_terms**

(continued)

numeric_factor_index *(continued)*

Example:

If i indexes the internally-simplified expression
`3 * x^2 * y`
then **numeric_factor_index**(i) returns index(3).

If i indexes the internally-simplified expression `x`
then **numeric_factor_index**(i) returns Float1Index if IS_ARITH_APPROX is true and Integer1Index if IS_ARITH_APPROX is false.

If i indexes the internally-simplified expression
`π * x`
then **numeric_factor_index**(i) returns Float1Index if IS_ARITH_APPROX is true and Integer1Index if IS_ARITH_APPROX is false.

If i indexes the internally-simplified expression
`2 * x + 2`
then **numeric_factor_index**(i) returns Float1Index if IS_ARITH_APPROX is true and Integer1Index if IS_ARITH_APPROX is false.

```
/* Returns the index of a tagged integer or Float one. */
push_quantum (PI_TAG);
numeric_factor_index (top_estack);
```

push_but_factor

Declaration: void **push_but_factor** (EStackIndex *i*, EStackIndex *j*)

Category(ies): Algebra Utilities

Description: This function can be used to push all but a selected syntactic factor out of an algebraic expression.

i indexes an internally-simplified algebraic expression and (*j* == *i* or *j* == **remaining_factors_index**(*i*) or *j* == **remaining_factors_index** (**remaining_factors_index**(*i*)) or . . .).

Pushes the product of all of *i* but **lead_factor**(*j*) onto the estack.

If *i* == *j* and ESTACK(*i*) != MULTIPLY_TAG, pushes Float1 if IS_ARITH_APPROX is true and Integer1 if IS_ARITH_APPROX is false.

Functions such as **index_rmng_fctrs_start_fctr_tag**, **index_rmng_fctrs_start_base_tag**, **index_rmng_fctrs_start_base**, and **index_rmng_factor** can be used to select such a factor.

Inputs:

i — Index of the top tag of an internally-simplified algebraic expression.

j — *i* or the index of one of the partial products of the expression indexed by *i*.

Outputs: None

Assumptions: None

Side Effects: None

Availability: All versions of the TI-89 / TI-92 Plus.

TI-89 / TI-92 Plus Differences: None

See Also: **lead_factor_index, remaining_factors_index**

Example: If *i* indexes x * y * 3 and *j* indexes the partial product y * 3, then `push_but_factor(i, j)` pushes x * 3.

```
void push_but_factor (EStackIndex i, EStackIndex j)
{  if (i == j)
      push_expression (remaining_factors_index (i));
   else  {  push_but_factor (remaining_factors_index (i), j);
            times_top (lead_factor_index (i));
         }
}
```

push_but_term

Declaration: void **push_but_term** (EStackIndex *i*, EStackIndex *j*)

Category(ies): Algebra Utilities

Description: Pushes onto the estack the sum of the terms of expression *i* without the leading term of *j*.

If *i* == *j* and ESTACK(*i*) != ADD_TAG, pushes Float0 if IS_ARITH_APPROX is true and Integer0 if IS_ARITH_APPROX is false.

Use this function to push all but a selected syntactic term out of an expression.

If *i* indexes x + y + 3 and *j* indexes the partial reductum y + 3, then `push_but_term(i, j)` pushes x + 3.

Functions such as **index_reductum_with_tag_base** can be used to select such a term.

Inputs:

i — Index of the top tag of an internally-simplified algebraic expression.

j — *i* or the index of one of the reductums of the expression indexed by *i*.

Outputs: None

Assumptions: None

Side Effects: None

Availability: All versions of the TI‑89 / TI‑92 Plus.

TI‑89 / TI‑92 Plus Differences: None

See Also: **lead_term_index, reductum_index**

Example:

```
void push_but_term (EStackIndex i, EStackIndex j)
{  if (i == j)
      push_expression (reductum_index (i));
   else
      {  push_but_term (reductum_index (i), j);
         add_to_top (lead_term_index (i));
      }
}
```

push_constant_factors

Declaration: void **push_constant_factors** (EStackIndex *k*)

Category(ies): Algebra Utilities

Description: Pushes onto the estack the product of all syntactic factors of the expression indexed by *k* that do not contain variables.

If there are no constant factors, pushes Float1 if IS_ARITH_APPROX is true and Integer1 if IS_ARITH_APPROX is false.

Inputs: *k* — Index of the top tag of an internally-simplified algebraic expression.

Outputs: None

Assumptions: None

Side Effects: None

Availability: All versions of the TI-89 / TI-92 Plus.

TI-89 / TI-92 Plus Differences: None

See Also: **push_nonconstant_factors, numeric_factor_index, push_nonnumeric_factors, push_dependent_factors, push_independent_factors**

Example:

```
push_pi_on_quantum (2u);
foo = top_estack;
push_quantum (8u);                    /* push variable x */
replace_top2_with_prod (foo);
push_constant_factors (top_estack);  /* Pushes pi*(1/2) */
```

push_denominator

Declaration: void **push_denominator** (EStackIndex *i*)

Category(ies): Algebra Utilities

Description: Pushes the denominator of the expression indexed by *i* onto the estack. The denominator of a float or integer is 1.

DIVIDE_TAG does not occur in internally-simplified expressions. Therefore, the denominator of a power is the reciprocal of the power if the degree is negative; otherwise the denominator is 1. The denominator of a product is the passive product of the reciprocals of the factors that have negative degrees. Otherwise the denominator is 1.

Note that:
The denominator of 1.5 is 1.
The denominator of x^-2 is x^-2.
The denominator of x ∗ (y + 1)^-1 ∗ y^-1 ∗ 3 is (y + 1) ∗ y.
The denominator of x^-1 + 3 is 1.

Inputs: *i* — Indexes the top tag of an internally-simplified algebraic expression.

Outputs: None

Assumptions: None

Side Effects: None

Availability: All versions of the TI‑89 / TI‑92 Plus.

TI‑89 / TI‑92 Plus Differences: None

See Also: **push_numerator, push_standardize, push_comdenom**

Example:

```
void push_denominator_of_com_denom (EStackIndex i)
/* Pushes the denominator of the expression that would be obtained by
   putting the expression indexed by i over a common denominator.
*/
{  Access_AMS_Global_Variables;
   EStackIndex old_top = top_estack;
   push_standardize (i);
   i = top_estack;
   push_denominator (i);
   delete_between (old_top, i);
}
```

push_dependent_factors

Declaration: void **push_dependent_factors** (EStackIndex *i*, EStackIndex *var*)

Category(ies): Algebra Utilities

Description: Pushes onto the estack the product of all top-level syntactic factors of the expression indexed by *i* that are dependent on the variable or kernel indexed by *var*. If there are no dependent factors, pushes Float1 if IS_ARITH_APPROX is true and Integer1 if IS_ARITH_APPROX is false.

Inputs:

i — Indexes the top tag of an internally-simplified algebraic expression.

var — Indexes the top tag of an internally-simplified variable or kernel.

Outputs: None

Assumptions: None

Side Effects: None

Availability: On AMS 2.02 and higher.

TI-89 / TI-92 Plus Differences: None

See Also: **push_independent_factors, push_constant_factors, push_nonconstant_factors, push_nonnumeric_factors, numeric_factor_index, push_but_factor, index_rmng_fctrs_start_base, index_rmng_fctrs_start_base_tag, index_rmng_fctrs_start_fctr_tag, index_rmng_factor**

Example:

```
void push_dependent_factors (EStackIndex i, EStackIndex var)
/* Pushes onto the estack the product of all factors of the term
   indexed by i that are dependent on the kernel indexed by var.
*/
{  if (MULTIPLY_TAG == ESTACK (i))
      {  push_dependent_factors (next_expression_index (--i), var);
         if (is_independent_of (i, var))
            return;
         else
            times_top (i);
      }
   else if (is_independent_of (i, var))
         push1 ();
   else  push_expression (i);
}
```

push_dependent_terms

Declaration: void **push_dependent_terms** (EStackIndex *i*, EStackIndex *var*)

Category(ies): Algebra Utilities

Description: Pushes onto the estack the sum of all top-level syntactic terms of the expression indexed by *i* that are dependent on the variable or kernel indexed by *var*. If there are no dependent terms, pushes Float0 if IS_ARITH_APPROX is true and Integer0 if IS_ARITH_APPROX is false.

Inputs:

i — Indexes the top tag of an internally-simplified algebraic expression.

var — Indexes the top tag of an internally-simplified variable or kernel.

Outputs: None

Assumptions: None

Side Effects: None

Availability: On AMS 2.02 and higher.

TI-89 / TI-92 Plus Differences: None

See Also: **push_independent_terms, push_constant_terms, push_nonconstant_terms, index_numeric_term**

Example:

```
void push_dependent_terms (EStackIndex i, EStackIndex var)
/* Pushes onto the estack the sum of all terms of the expression
   indexed by i that are dependent on the variable indexed by var.
*/
{  if (ADD_TAG == ESTACK (i))
      {  push_dependent_terms (next_expression_index (--i), var);
         if (is_independent_of (i, var))
            return;
         else
            add_to_top (i);
      }
   else if (is_independent_of (i, var))
            push0 ();
   else
      push_expression (i);
}
```

push_desolve

Declaration: void **push_desolve** (EStackIndex *i*)

Category(ies): Algebra Utilities

Description: Pushes the particular solution of a differential equation if there are any given initial or boundary conditions. Otherwise pushes the general solution.

If invoked via **push_internal_simplify**, the independent and dependent variables, then the differential equation and any initial conditions are simplified to deepest variables.

Inputs: *i* — Index of the top tag of a differential equation (perhaps with one or two equations establishing initial or boundary conditions) on top of its independent variable, on top of its dependent variable, on top of an END_TAG.

Outputs: None

Assumptions: None

Side Effects: None

Availability: On AMS 2.02 and higher.

TI-89 / TI-92 Plus Differences: None

See Also: None

Example:

```
push_quantum (END_TAG);
push_quantum (9u);  /* Push dependent variable y */
push_quantum (8u);  /* Push independent variable x */
push_quantum (9u);  /* Push y as right side of the differential equation */
push_quantum (9u);
push_quantum (PRIME_TAG);   /* Push y' as left side */
push_quantum (EQUATION_TAG);
push_desolve (top_estack);  /* Push general solution y = @1 * e^x  */
```

push_div_dif_1c

Declaration: void **push_div_dif_1c** (EStackIndex *i*, EStackIndex *vi*, EStackIndex *j*)

Category(ies): Algebra Utilities

Description: Pushes onto the estack the centered first difference (TI-BASIC function nDeriv) of the expression indexed by *i*, with respect to the variable indexed by *vi*, using the expression indexed by *j* as the step size.

For example: nDeriv (f(x), x, h) -> (f(x + h) - f(x - h))/(2h).

If invoked via the **push_internal_simplify** function, *vi* then *i* are simplified to deepest variable. However, if the deepest variable value of *vi* has a such-that or STO▸ value, that value is substituted for the deepest variable value after computing the first difference.

For example, nDeriv (x^3, x, h) | x = 1 -> h^2 + 3.

Inputs:

i — Index of the top tag of an internally-simplified algebraic expression, comparison or aggregate thereof.

vi — Index of a variable.

j — Index of the top tag of an internally-simplified algebraic expression or an aggregate thereof.

Outputs: None

Assumptions: None

Side Effects: None

Availability: On AMS 2.02 and higher.

TI-89 / TI-92 Plus Differences: None

See Also: **push_div_dif_1f**

Example:

```
push_Float (1e-4);
h = top_estack;
push_quantum (8u);
vi = top_estack;  /* Push variable x */
push_quantum (EXP_TAG);               /* top_estack -> e^x */
push_div_dif_1c (top_estack, vi, h);  /* push nDeriv(e^x, x, 1e-4) */
```

push_div_dif_1f

Declaration: void **push_div_dif_1f** (EStackIndex *i*, EStackIndex *vi*, EStackIndex *j*)

Category(ies): Algebra Utilities

Description: Pushes onto the estack the forward first difference (TI-BASIC function avgRC) of the expression indexed by *i*, with respect to the variable indexed by *vi*, using the expression indexed by *j* as the step size.

For example: avgRC (f(x), x, h) -> (f(x + h) - f(x))/h.

If invoked via the **push_internal_simplify** function, *vi* then *i* are simplified to deepest variable. However, if the deepest variable value of *vi* has a such-that or STO▸ value, that value is substituted for the deepest variable value after computing the first difference.

For example, avgRC (x^3, x, h) | x = 1 -> h^2 + 3h + 3.

Inputs:

i — Index of the top tag of an internally-simplified algebraic expression, comparison or aggregate thereof.

vi — Index of a variable.

j — Index of the top tag of an internally-simplified algebraic expression or an aggregate thereof.

Outputs: None

Assumptions: None

Side Effects: None

Availability: On AMS 2.02 and higher.

TI-89 / TI-92 Plus Differences: None

See Also: **push_div_dif_1c**

Example:

```
push_Float (1e-4);
h = top_estack;
push_quantum (8u);
vi = top_estack;  /* Push variable x */
push_quantum (EXP_TAG);               /* top_estack -> e^x */
push_div_dif_1f (top_estack, vi, h);  /* push avgRC(e^x, x, 1e-4) */
```

push_independent_factors

Declaration: void **push_independent_factors** (EStackIndex *i*, EStackIndex *var*)

Category(ies): Algebra Utilities

Description: Pushes onto the estack the product of all syntactic factors of the expression indexed by *i* that are independent of the variable or kernel indexed by *var*. If there are no independent factors, pushes Float1 if IS_ARITH_APPROX is true and Integer1 if IS_ARITH_APPROX is false.

Inputs:

i — Index of the top tag of an internally-simplified algebraic expression.

var — Index of the top tag of an internally-simplified variable or kernel.

Outputs: None

Assumptions: None

Side Effects: None

Availability: On AMS 2.02 and higher.

TI-89 / TI-92 Plus Differences: None

See Also: **push_dependent_factors, push_constant_factors, push_nonconstant_factors, push_nonnumeric_factors, numeric_factor_index, push_but_factor, index_rmng_fctrs_start_base, index_rmng_fctrs_start_base_tag, index_rmng_fctrs_start_fctr_tag, index_rmng_factor**

Example:

```
push_quantum (PI_TAG);
k = top_estack;
push_quantum (8u);          /* Push variable x */
replace_top2_with_prod (k);
k = top_estack;
push_quantum (9u);
j = top_estack;             /* Push variable y */
push_ln (j);                /* Push ln(y); */
replace_top2_with_prod(j);  /* top_estack -> ln(y) * y  */
replace_top2_with_prod(k);
k = top_estack;               /* top_estack -> x * ln(y) * y * pi  */
push_quantum (9u);          /* Push y */
push_independent_factors (k, top_estack);  /* Pushes x * pi  */
```

push_independent_terms

Declaration: void **push_independent_terms** (EStackIndex *i*, EStackIndex *var*)

Category(ies): Algebra Utilities

Description: Pushes onto the estack the sum of all syntactic terms of the expression indexed by *i* that are independent of the variable or kernel indexed by *var*. If there are no independent terms, pushes Float0 if IS_ARITH_APPROX is true and Integer0 if IS_ARITH_APPROX is false.

Inputs:

i — Index of the top tag of an internally-simplified algebraic expression.

var — Index of the top tag of an internally-simplified variable or kernel.

Outputs: None

Assumptions: None

Side Effects: None

Availability: On AMS 2.02 and higher.

TI-89 / TI-92 Plus Differences: None

See Also: None

Example:

```
push_quantum (PI_TAG);
k = top_estack;
push_quantum (8u);           /* Push variable x */
replace_top2_with_sum (k);
k = top_estack;             /* k -> x + pi */
push_quantum (9u);
j = top_estack;             /* Push variable y */
push_ln (j);                /* Push ln(y); */
replace_top2_with_sum(j);   /* top_estack -> ln(y) + y */
replace_top2_with_sum(k);
k = top_estack;               /* top_estack -> x + ln(y) + y + pi */
push_quantum (9u);          /* Push y */
push_independent_terms (k, top_estack);  /* Pushes x + pi */
```

push_integer_gcd

Declaration: void **push_integer_gcd** (EStackIndex *i*, EStackIndex *j*)

Category(ies): Algebra Utilities

Description: If *i* and *j* both index numbers, pushes their greatest common divisor. If *i* and/or *j* index(es) aggregate(s), maps over the elements of the aggregate(s). Otherwise pushes INT_GCD_TAG on top of a copy of expression *i* on top of a copy of expression *j*.

Inputs: *i*, *j* — Indices of the top tags of internally-simplified algebraic expressions or aggregates thereof.

Outputs: None

Assumptions: None

Side Effects: None

Availability: On AMS 2.02 and higher.

TI-89 / TI-92 Plus Differences: None

See Also: None

Example:

```
push_quantum_as_nonnegative_int (10u);
k = top_estack;
push_Float (15.0);
push_integer_gcd (k, top_estack);  /* Pushes tagged float 5.0 */
```

push_integer_lcm

Declaration: void **push_integer_lcm** (EStackIndex *i*, EStackIndex *j*)

Category(ies): Algebra Utilities

Description: If *i* and *j* both index numbers, pushes their least common multiple. If *i* and/or *j* index(es) aggregate(s), maps over the elements of the aggregate(s). Otherwise pushes INT_LCM_TAG on top of a copy of expression *i* on top of a copy of expression *j*.

Inputs: *i*, *j* — Indices of the top tags of internally-simplified algebraic expressions or aggregates thereof.

Outputs: None

Assumptions: None

Side Effects: None

Availability: On AMS 2.02 and higher.

TI-89 / TI-92 Plus Differences: None

See Also: None

Example:

```
push_quantum_as_nonnegative_int (10u);
k = top_estack;
push_Float (15.0);
push_integer_lcm (k, top_estack);  /* Pushes tagged float 30.0 */
```

push_nonconstant_factors

Declaration: void **push_nonconstant_factors** (EStackIndex *i*)

Category(ies): Algebra Utilities

Description: Pushes onto the estack the product of all syntactic factors of the expression indexed by *i* that include variables. If there are no nonconstant factors, pushes Float1 if IS_ARITH_APPROX is true and Integer1 if IS_ARITH_APPROX is false.

Inputs: *i* — Indexes the top tag of an internally-simplified algebraic expression.

Outputs: None

Assumptions: None

Side Effects: None

Availability: On AMS 2.02 and higher.

TI-89 / TI-92 Plus Differences: None

See Also: **push_constant_factors, numeric_factor_index, push_nonnumeric_factors, push_dependent_factors, push_independent_factors**

Example:

```
push_pi_on_quantum (2u);
foo = top_estack;
push_quantum (8u);                      /* push variable x */
replace_top2_with_prod (foo);           /* top-estack -> x * pi * 2  */
push_nonconstant_factors (top_estack);  /* Pushes x */
```

push_nonconstant_terms

Declaration: void **push_nonconstant_terms** (EStackIndex *i*)

Category(ies): Algebra Utilities

Description: Pushes onto the estack the sum of all syntactic terms of the expression indexed by *i* that include variables. If there are no nonconstant terms, pushes Float0 if IS_ARITH_APPROX is true and Integer0 if IS_ARITH_APPROX is false.

Inputs: *i* — Indexes the top tag of an internally-simplified algebraic expression.

Outputs: None

Assumptions: None

Side Effects: None

Availability: On AMS 2.02 and higher.

TI-89 / TI-92 Plus Differences: None

See Also: **push_constant_terms, push_dependent_terms, push_independent_terms, index_numeric_term,**

Example:

```
push_quantum (PI_TAG);
add1_to_top ();                      /* partial sum = top_estack */
push_quantum (8u);                   /* push variable x */
replace_top2_with_sum (foo);         /* top-estack -> x + pi + 1 */
push_nonconstant_terms (top_estack); /* Pushes x */
```

push_nonnumeric_factors

Declaration: void **push_nonnumeric_factors** (EStackIndex *i*)

Category(ies): Algebra Utilities

Description: Pushes onto the estack the product of all syntactic factors of the expression indexed by *i* except for the final numeric factor, if any. If there are no non-numeric factors, pushes Float1 if IS_ARITH_APPROX is true and Integer1 if IS_ARITH_APPROX is false.

Inputs: *i* — Indexes the top tag of an internally-simplified algebraic expression.

Outputs: None

Assumptions: None

Side Effects: None

Availability: On AMS 2.02 and higher.

TI-89 / TI-92 Plus Differences: None

See Also: **push_constant_factors, numeric_factor_index, push_nonconstant_factors, push_dependent_factors, push_independent_factors**

Example:

```
push_pi_on_quantum (2u);
foo = top_estack;
push_quantum (8u);                    /* push variable x */
replace_top2_with_prod (foo);         /* top-estack -> x * pi/2  */
push_nonnumeric_factors (top_estack); /* Pushes x * pi  */
```

push_numerator

Declaration: void **push_numerator** (EStackIndex *i*)

Category(ies): Algebra Utilities

Description: Pushes onto the estack the syntactic numerator of expression *i*.

The numerator of a float or integer is the float or integer.

DIVIDE_TAG does not occur in internally-simplified expressions, so:

- The numerator of a power is the power if the degree is positive; otherwise the numerator is 1.0 or 1 depending on IS_ARITH_APPROX.
- The numerator of a product is the product of the factors that do not have negative degrees.
- Otherwise the numerator is the entire expression.

Note that:
The numerator of 1.5 is 1.5.
The numerator of x^-2 is 1.0 or 1 depending on IS_ARITH_APPROX.
The numerator of x ∗ (y + 1)^-1 ∗ y^-1 ∗ 3 is x ∗ 3.
The numerator of x^-1 + 3 is the entire expression.

Inputs: *i* — Indexes the top tag of an internally-simplified algebraic expression.

Outputs: None

Assumptions: None

Side Effects: None

Availability: On AMS 2.02 and higher.

TI-89 / TI-92 Plus Differences: None

See Also: **push_denominator**

Example:

```
void push_numerator_of_com_denom (EStackIndex i)
/* Pushes the numerator of the expression that would be obtained by
   putting the expression indexed by i over a common denominator.
*/
{  Access_AMS_Global_Variables;
   EStackIndex old_top = top_estack;
   push_standardize (i);
   i = top_estack;
   push_numerator (i);
   delete_between (old_top, i);
}
```

push_percent

Declaration: void **push_percent** (EStackIndex *i*)

Category(ies): Algebra Utilities

Description: Pushes the expression indexed by *i*, divided by 100.

Inputs: *i* — Index of the top tag of an internally-simplified algebraic expression, algebraic comparison or aggregate thereof.

Outputs: None

Assumptions: None

Side Effects: None

Availability: On AMS 2.02 and higher.

TI-89 / TI-92 Plus Differences: None

See Also: None

Example:

```
push_Float (55.0)
push_percent (top_estack);  /* Pushes 0.55 */
```

push_poly_deg_in_var_or_kernel

Declaration: void **push_poly_deg_in_var_or_kernel** (EStackIndex *i*, EStackIndex *vi*)

Category(ies): Algebra Utilities

Description: Pushes onto the estack the degree of the polynomial indexed by *i* in the variable or kernel indexed by *vi*. Degree(0) is 0, not $-\infty$ as in some definitions.

Inputs:
i — Index of the top tag of an internally-simplified polynomial in *vi*, generalized to permit negative and fractional powers of *vi*.

vi — Index of the top tag of an internally-simplified variable or kernel.

Outputs: None

Assumptions: None

Side Effects: None

Availability: On AMS 2.02 and higher.

TI-89 / TI-92 Plus Differences: None

See Also: **is_polynomial_in_var_or_kern, is_totally_polynomial, linear_degree, did_push_lincf, index_if_pushed_binomial_info, index_if_pushed_qquad_info**

Example: If *i* indexes the internally-simplified expression `ln(y)^-(1/2) * (x + ln(y))^2`
and *vi* indexes the internally-simplified expression `ln(y)`,
then **push_poly_deg_in_var_or_kernel**(i, vi) pushes 3/2.

```
push_quantum (3u);
exponent = top_estack;
push_quantum (8u);  /* Push variable x */
replace_top2_with_pow (exponent);   /* top_estack -> x^3 */
add1_to_top ();
poly = top_estack;  /* top_estack -> x^3 + 1 */
push_quantum (8u);  /* Push variable x */
push_poly_deg_in_var_or_kernel (poly, top_estack);  /* Pushes tagged integer 3 */
```

push_subst_no_simp

Declaration: void **push_subst_no_simp** (EStackIndex *i*, EStackIndex *j*, EStackIndex *k*)

Category(ies): Algebra Utilities

Description: Pushes onto the estack without further simplification a copy of the expression indexed by *i* in which subexpressions that are identical to the expression indexed by *j* are replaced by the expression indexed by *k*.

Inputs: *i, j, k* — Indices the top tags of expressions.

Outputs: None

Assumptions: None

Side Effects: None

Availability: On AMS 2.00 and higher.

TI-89 / TI-92 Plus Differences: None

See Also: **push_substitute_simplify**

Example:

```
push_quantum (9u);
k = top_estack;       /* k -> variable y */
push_quantum (8u);
j = top_estack;       /* j -> variable x */
push_quantum_as_nonnegative_int(2u);
exponent = top_estack;
raise_to_top (j);
power = top_estack;   /* x^2 */
push_quantum_as_nonnegative_int(2u);
exponent = top_estack;
raise_to_top (k);    /* y^2 */
replace_top2_with_difference (power);   /* top_estack -> x^2 - y^2 */
push_subst_no_simp (top_estack, j, k);  /* Pushes y^2 - y^2 */
```

push_substitute_simplify

Declaration: void **push_substitute_simplify** (EStackIndex *i*, EStackIndex *j*, EStackIndex *k*)

Category(ies): Algebra Utilities

Description: Pushes onto the estack the result of substituting the expression indexed by *k* for the expression indexed by *j* in the expression indexed by *i*, and simplifying the result.

Inputs: *i*, *j*, *k* — Indices of the top tags of expressions.

Outputs: None

Assumptions: None

Side Effects: None

Availability: On AMS 2.04 and higher.

TI-89 / TI-92 Plus Differences: None

See Also: **push_subst_no_simp, push_lim**

Example:

```
push_quantum (9u);
k = top_estack;       /* k -> variable y */
push_quantum (8u);
j = top_estack;       /* j -> variable x */
push_quantum_as_nonnegative_int(2u);
exponent = top_estack;
raise_to_top (j);
power = top_estack;   /* x^2 */
push_quantum_as_nonnegative_int(2u);
exponent = top_estack;
raise_to_top (k);    /* y^2 */
replace_top2_with_difference (power);   /* top_estack -> x^2 - y^2 */
push_substitute_simplify (top_estack, j, k);   /* Pushes a zero. */
```

push_substitute_using_such_that

Declaration: void **push_substitute_using_such_that** (EStackIndex *i*, EStackIndex *vi*, EStackIndex *val*)

Category(ies): Algebra Utilities

Description: Substitutes a value for a variable throughout an expression.

Inputs:
i — EStackIndex of the target expression.

vi — EStackIndex of a variable.

val — EStackIndex of the value to substitute.

Outputs: Returns the fully simplified, internal tokenized form of *i* after substituting *val* for each occurrence of *vi*. Any pre-existing assigned value or substitution value for *vi* will be ignored during this operation.

Assumptions: None

Side Effects: May cause estack expansion, heap compression, or throw errors associated with the full simplification of the expression.

Availability: On AMS 2.04 and higher.

TI-89 / TI-92 Plus Differences: None

See Also: **NG_such_that_index**

Example:

If i indexes the bolded tag in the expression x + 1 as follows
1 1 NONNEGATIVE_INTEGER_TAG X_VAR_TAG **ADD_TAG**

and j indexes the bolded tag in the variable x as follows
X_VAR_TAG

and k indexes the bolded tag in the value 2 as follows
2 1 **NONNEGATIVE_INTEGER_TAG**

then

```
push_substitute_using_such_that (i, j, k);
```

will substitute 2 for x and then simplify the expression to 3 in the estack such that **top_estack** points to the bolded tag.
3 1 **NONNEGATIVE_INTEGER_TAG**

push_var_kern_tail

Declaration: void **push_var_kern_tail** (EStackIndex *i*)

Category(ies): Algebra Utilities

Description: Pushes onto the estack an END_TAG, then the most main variable or kernel, then the next most main variable or kernel, etc.

Inputs: *i* — Indexes the top tag of an internally-simplified algebraic expression.

Outputs: None

Assumptions: None

Side Effects: None

Availability: On AMS 2.02 and higher.

TI-89 / TI-92 Plus Differences: None

See Also: **main_gen_var_index, next_var_or_kernel_index, has_different_variable**

Example:

```
push_quantum (8u);   /* Push variable x */
add1_to_top ();
partial_sum = top_estack;  /* x + 1 */
push_quantum (9u);  /* Push variable y */
push_quantum (LN_TAG);
replace_top2_with_sum (partial_sum);  /* top_estack = x + ln(y) + 1 */
push_var_kern_tail (top_estack);
push_quantum (LIST_TAG);  /* top_estack -> {ln(y), x} */
```

re_index

Declaration: EStackIndex **re_index** (EStackIndex *k*)

Category(ies): Algebra Utilities

Description: If *k* indexes an IM_RE_TAG, returns the index of the second expression below it, which is the real part for internally-simplified algebraic expressions. Otherwise returns *k*.

Inputs: *k* — Indexes the top tag of an internally-simplified algebraic expression, an algebraic comparison, or an aggregate thereof.

Outputs: If *k* indexes an IM_RE_TAG, returns the index of the second expression below it, which is the real part for internally-simplified algebraic expressions. Otherwise returns *k*.

Assumptions: None

Side Effects: None

Availability: On AMS 1.05 and higher.

TI-89 / TI-92 Plus Differences: None

See Also: **push_im, replace_top2_with_imre**

Example:

```
push_expression (Integer1Index);
real_part = top_estack;
push_expression (FloatPiIndex);
replace_top2_with_imre (real_part);
re_index (top_estack);  /* Returns the index of a tagged integer 1 */
```

reductum_index

Declaration: EStackIndex **reductum_index** (EStackIndex *i*)

Category(ies): Algebra Utilities

Description: If *i* indexes an ADD_TAG, returns the index of the deeper of its two operands. Otherwise returns Float0Index if IS_ARITH_APPROX is true and Integer0Index if IS_ARITH_APPROX is false. Internally-simplified expressions have the most main term shallowest.

For example:

If i indexes the internally-simplified expression
`2 + x + y`
then **reductum_index**(i) returns index(y + 2).

If i indexes the internally-simplified expression (2)
then **reductum_index**(i) returns Float0Index if IS_ARITH_APPROX is true and Integer0Index if IS_ARITH_APPROX is false.

Note that Float0Index and Integer0Index are not ordinarily physically within the expression indexed by *i*.

For the default mode IS_RECURSIVE, similar powers of the main variable are collected.

For example:

If i indexes the internally-simplified expression
`x^2 * y + x^2 + 5)`
then **reductum_index**(i) returns index(5).

Internally-simplified differences are represented as sums with negated subtrahends.

For example:

If i indexes the internally-simplified expression
`x - y`
then **reductum_index**(i) returns index(y ∗ (-1)).

Inputs: *i* — Indexes the top tag of an expression.

Outputs: If *i* indexes an ADD_TAG, returns the index of the deeper of its two operands. Otherwise returns Float0Index if IS_ARITH_APPROX is true and Integer0Index if IS_ARITH_APPROX is false.

Assumptions: None

Side Effects: None

(continued)

reductum_index *(continued)*

Availability: On AMS 1.05 and higher.

TI-89 / TI-92 Plus Differences: None

See Also: **lead_term_index**

Example:

```
push_quantum (8u);  /* Push variable x */
add1_to_top ();
reductum_index (top_estack);  /* Returns the index of a one. */
```

remaining_factors_index

Declaration: EStackIndex **remaining_factors_index** (EStackIndex *i*)

Category(ies): Algebra Utilities

Description: If *i* indexes a MULTIPLY_TAG, returns the index of the deeper of its two operands. Otherwise returns Float1Index if IS_ARITH_APPROX is true and Integer1Index if IS_ARITH_APPROX is false.

For example:

If i indexes the internally-simplified expression
`(x^2)`
then **remaining_factors_index**(i) returns Float1Index if IS_ARITH_APPROX is true and Integer1Index if IS_ARITH_APPROX is false.

Internally-simplified products and ratios have the most main factor shallowest, with less main factors below that. Also, the lead factor of an internally-simplified product is never a product.

For example:

If i indexes the internally-simplified expression
`(3 * x^2) * y`
then **remaining_factors_index**(i) returns index(3 ∗ y).

For the default mode IS_RECURSIVE, similar powers of the main variable are collected.

For example:

If i indexes the internally-simplified expression
`x^2 * y + x^2`
then **remaining_factors_index**(i) returns index(y + 1).

Internally-simplified numeric denominator factors are combined with numeric numerator factors into a single numeric factor.

For example:

If i indexes the internally-simplified expression
`3/2`
then **remaining_factors_index**(i) returns Float1Index if IS_ARITH_APPROX is true and Integer1Index if IS_ARITH_APPROX is false.

Note that Float1Index and Integer1Index are not normally physically within the expression indexed by *i*.

(continued)

remaining_factors_index *(continued)*

Description: *(continued)*

Non-numeric denominator factors are internally simplified to be merged with numerator factors as negative powers.

For example:

If i indexes the internally-simplified expression
`2/x`
then **remaining_factors_index**(i) returns index (2).

Inputs:

i — Indexes the top tag of an expression.

Outputs:

If *i* indexes a MULTIPLY_TAG, returns the index of the deeper of its two operands. Otherwise returns Float1Index if IS_ARITH_APPROX is true and Integer1Index if IS_ARITH_APPROX is false.

Assumptions:

None

Side Effects:

None

Availability:

On AMS 1.05 and higher.

TI-89 / TI-92 Plus Differences:

None

See Also:

lead_factor_index

Example:

```
push_quantum (x);  /* Push variable x */
remaining_factors_index (top_estack); /* Returns an index of a one. */
```

replace_top2_with_imre

Declaration: void **replace_top2_with_imre** (EStackIndex *i*)

Category(ies): Algebra Utilities

Description: The top two expressions on the estack are internally-simplified algebraic expressions, and *j* indexes the deeper of these two expressions. If either is Float and the other is numeric, replaces with 0.0 either that has magnitude <= **IM_re_tol** * (magnitude of its companion). Then deletes the top expression if it is zero, otherwise pushing an IM_RE_TAG.

Inputs: *i* — Indexes the top tag of the internally-simplified algebraic expression **next_expression_index** (**top_estack**).

Outputs: None

Assumptions: The top two expressions are internally-simplified algebraic expressions.

Side Effects: None

Availability: On AMS 2.02 and higher.

TI-89 / TI-92 Plus Differences: None

See Also: None

Example:

```
/* Replaces the top two expressions with the expression 3.7 + 5.2i */
push_Float (3.7);
i = top_estack;
push_Float (5.2);
replace_top2_with_imre (i);
```

Appendix A: System Routines — Apps

See Also:

EV_getAppID

Declaration: AppID **EV_getAppID** (UCHAR const * *appName*)

Category(ies): Apps, Operating System

Description: Get the ID of an application given its internal name.

Inputs: *appName* — Internal name of application. Each application has a unique internal name (≤ 8 characters). The internal names of the built-in applications are listed in **Table 7.3 Internal Names of Build-in Applications**.

Outputs: ID of application.

Assumptions: The ID of an application and the memory handle to the app's ACB (application control block) are one and the same.

Side Effects: None

Availability: On AMS 2.00 and higher.

TI-89 / TI-92 Plus Differences: None

See Also: Section **7.1.3.2. Internal Application Name**.

Example:

```
AppID grapher = EV_getAppID("TIGRAPH");
```

EV_quit

Declaration: void **EV_quit** (void)

Category(ies): Apps, Operating System, Home Screen

Description: Switches from the current app to the Home screen.

Usually this forces your app to quit before switching to the Home screen. However, if the calculator is in split-screen mode and Home is already on the other side of the split, this will deactivate your app then activate the Home screen.

Inputs: None

Outputs: None

Assumptions: None

Side Effects: Any actions taken to deactivate or quit your application. Starting the Home screen may cause heap compression.

Availability: On AMS 2.00 and higher.

TI-89 / TI-92 Plus Differences: None

See Also: Not applicable.

Example:

```
EV_quit();
```

OO_appGetPublicStorage

Declaration: ULONG **OO_appGetPublicStorage** (void)

Category(ies): Apps

Description: Get the contents of the running app's public storage. See **OO_appSetPublicStorage** for a description of public storage.

Inputs: None

Outputs: Public storage may contain anything which fits in 32 bits. You must cast the return value to the type of data actually stored in public storage if it is not an unsigned integer.

Assumptions: This routine fetches public storage from the application indicated by the app ID in OS global variable **EV_runningApp**. TI-BASIC extension functions and shared-code libraries may exist in a different app. These types of routines are often called as subroutines from other applications, in which case **EV_runningApp** contains not the ID of the app containing your subroutine, but the ID of the calling application.

Side Effects: None

Availability: On AMS 2.00 and higher.

TI-89 / TI-92 Plus Differences: None

See Also: **OO_appSetPublicStorage**

Example: See **OO_appSetPublicStorage**.

OO_appIsMarkedDelete

Declaration: BOOL **OO_appIsMarkedDelete** (AppID *id*)

Category(ies): Apps

Description: Checks an application to see if it is marked to be deleted.

Inputs: *id* — ID of application to check.

Outputs: Returns TRUE if app *id* is marked to be deleted.

Assumptions: This routine does not check if app *id* exists. You could crash the calculator if app *id* does not exist.

Side Effects: None

Availability: On AMS 2.00 and higher.

TI-89 / TI-92 Plus Differences: None

See Also: Not applicable

Example:

```
AppID cabri = EV_getAppID((UCHAR *)"TICABRI");   /* Get ID of Cabri Geometry */
if (cabri != H_NULL)                          /* Is Cabri installed in calculator? */
   if (OO_appIsMarkedDelete(cabri))              /* Is it marked for deletion? */
   {
       .
       .
       .
  }
```

OO_appMarkDelete

Declaration: void **OO_appMarkDelete** (AppID *id*)

Category(ies): Apps, Operating System

Description: Marks an app to be deleted from Flash memory.

This routine does not cause the app to be deleted immediately. The calculator deletes marked apps upon returning to the Home screen. This allows an app to delete itself by calling **OO_appMarkDelete** with its own app ID.

Inputs: *id* — App ID of application to delete.

Outputs: None

Assumptions: None

Side Effects: When an app is deleted, all trailing undeleted apps must be moved up in Flash memory in a process called garbage collection. These apps are notified with a CM_PACK message which gives them a chance to save state before moving.

After Flash memory is compressed, the same apps are then sent a CM_UNPACK message so they can restore state.

Availability: On AMS 2.00 and higher.

TI-89 / TI-92 Plus Differences: None

See Also: Section **9.10. Installing, Moving, and Deleting an Application**.

Example:

```
OO_appMarkDelete(EV_currentApp); /* Delete myself */
```

OO_AppNameToACB

Declaration: AppID **OO_AppNameToACB** (UCHAR const * *appname*, BOOL *csen*)

Category(ies): Apps

Description: Looks up ID of application given the name of the app as it appears in the APPS menu.

> **Note:** The app's name of an application is subject to change by language localizers. Use **EV_getAppID** to find the ID of an app by its internal name. An app's internal name is unique and does not change when the language mode setting is changed.

Inputs:

appname — Name of application as it appears in APPS menu.

csen — Case-sensitive comparison, TRUE if *appname* must match case, FALSE if comparison is case-insensitive.

Outputs: ID of first app with matching *appname* or 0 if app cannot be found.

Assumptions: None

Side Effects: None

Availability: On AMS 2.00 and higher.

TI-89 / TI-92 Plus Differences: None

See Also: **EV_getAppID**

Example:

```
/* Get App ID of numeric solver. This may not work if the language
   mode setting has been changed to something besides English.
*/
AppID solverID = OO_AppNameToACB("Numeric Solver");
```

OO_appSetPublicStorage

Declaration: void **OO_appSetPublicStorage** (ULONG *ps*)

Category(ies): Apps

Description: Save a value in the running app's public storage.

The application control block of each app has a place to store user data. This public storage is large enough to contain a 32-bit value, typically a memory handle, but can be a pointer or integer.

Garbage collecting Flash memory may involve moving an app. When this happens, the app is reinitialized after it is moved to its new location. Consequently, the contents of the app's data segment (static and global variables) are reset to their initial values. Public storage is a convenient place to save data which is preserved through app reinitialization.

Inputs: *ps* — Value to save in the app's public storage.

Outputs: None

Assumptions: This routine assumes you want to store user data in the application control block indicated by the app ID in OS global variable **EV_runningApp**. This may not be the case with TI-BASIC extension functions and shared-code libraries. These types of routines are often called as subroutines from other applications, in which case **EV_runningApp** contains not the ID of your app, but the ID of the calling application. This is an instance when your routines should not modify the calling app's public storage.

Side Effects: None

Availability: On AMS 2.00 and higher

TI-89 / TI-92 Plus Differences: None

See Also: **OO_appGetPublicStorage**

Example: This example illustrates how to save your global variables during Flash memory garbage collection. The idea is to group all your global variables into a single structure. Then when it is time to garbage collect, it is easy to allocate a block of memory from the heap large enough to hold your globals, and store the memory block's handle in public storage.

(continued)

OO_appSetPublicStorage *(continued)*

```
typedef struct
{
   USHORT flags;
   char name[40];
   BCD16 result;
} GLOBALS;  ❶

GLOBALS g;  ❷
.
.
.
void main(pFrame self, Event *e)
{
   HANDLE h;

   switch (e->command)
   {
      .
      .
      .
      case CM_PACK:
         /* Getting ready to garbage collect -- save global variables */
         h = HeapAlloc(sizeof(GLOBALS));  ❸
         if (h != H_NULL)
         {
            GLOBALS *pg = HeapDeref(h);
            *pg = g;  ❹
            OO_appSetPublicStorage(h);  ❺
         }
         else
            OO_appSetPublicStorage(H_NULL);
         break;

      case CM_UNPACK:
         /* Garbage collect is finished -- restore global variables */
         h = OO_appGetPublicStorage();  ❻
         if (h != H_NULL)
         {
            GLOBALS *pg = HeapDeref(h);
            g = *pg;  ❼
            HeapFree(h);  ❽
         }
         break;
```

❶ Declare all the global variables that need to be saved in a single structure.

❷ Define a struct variable to hold the global variables.

❸ Allocate memory from the heap to hold the global variables.

❹ Copy the global variables to the heap.

❺ Save handle to global variables in public storage.

❻ Recover handle to global variables from public storage.

❼ Copy global variables from heap to data segment.

❽ Do not forget to release heap memory.

OO_CondGetAttr

Declaration: BOOL **OO_CondGetAttr** (pFrame *obj*, ULONG *selector*, void ** *value*)

Category(ies): Apps

Description: Conditionally retrieves an attribute of an object frame.

The prototype chain of each frame in the parent hierarchy beginning with *obj* is searched until attribute *selector* is found or the top of the parent hierarchy is reached.

Global variable **OO_SuperFrame** is updated with the parent of the frame where the attribute was located or NULL if the attribute was not found.

Inputs:

obj — Object frame pointer.

selector — Attribute selector number.

value — Address of location to return attribute value.

Outputs: Returns TRUE if attribute was found and contents stored in * *value*. Returns FALSE if attribute was not found and * *value* is not updated.

Assumptions: None

Side Effects: None

Availability: On AMS 2.00 and higher.

TI-89 / TI-92 Plus Differences: None

See Also: **OO_GetAppAttr, OO_GetAttr, OO_HasAttr, OO_SetAppAttr, OO_SetAttr**

Example:

```
char *helpMsg;
   .
   .
   .
if (OO_CondGetAttr(MyAppObj, helpMsgNum, (void *)&helpMsg))
{
   /* Display help message */
   .
   .
   .
}
```

OO_Deref

Declaration: Frame * **OO_Deref** (pFrame *pointer*)

Category(ies): Apps, Memory Management

Description: Object frame pointers (type pFrame) come in three flavors: handles, address pointers, and indirect references to the system object frame. This routine figures out the flavor of *pointer* and translates it into a real address.

Given the real address, the frame's header and attributes can be directly accessed.

Inputs: *pointer* — An object frame pointer.

Outputs: Returns address of frame referenced by *pointer*.

Assumptions: None

Side Effects: None

Availability: On AMS 2.00 and higher.

TI-89 / TI-92 Plus Differences: None

See Also: None

Example:

```
pFrame obj;
Frame *pobj;
ULONG i;
   .
   .
   .
obj = OO_New(H_NULL);  /* allocate new frame from the heap */
pobj = OO_Deref(obj);  /* get pointer to frame */
for (i = 0; i < pobj->head.count; i += 1)
{
   /* process each attribute of object frame */
   .
   .
   .
}
```

OO_Destroy

Declaration: pFrame **OO_Destroy** (pFrame *obj*)

Category(ies): Apps

Description: Releases space from an object frame and its prototype chain. Use this routine to free frames allocated by **OO_New**.

Inputs: *obj* — pointer to first object frame in prototype chain to free

Outputs: Returns a pointer to the first frame in the prototype chain which could not be freed (resides in Flash memory) or H_NULL if the entire prototype chain was freed.

Assumptions: The prototype chain of an object frame frequently ends with a Flash-resident frame, either a frame in your app or the system frame. This routine does not attempt to free a frame which resides in Flash memory.

Side Effects: None

Availability: On AMS 2.00 and higher.

TI-89 / TI-92 Plus Differences: None

See Also: **OO_DestroyAll, OO_New**

Example:

```
pFrame obj;
   .
   .
   .
obj = OO_New(H_NULL);  /* allocate a new object */
   .
   . /* use object */
   .
OO_Destroy(obj);  /* free object */
```

OO_DestroyAll

Declaration: pFrame **OO_DestroyAll** (pFrame *obj*)

Category(ies): Apps

Description: Releases space from an object frame, its prototype chain, its parent chain, and all their prototype chains. Use this routine to free frames allocated by **OO_New**.

Inputs: *obj* — Pointer to first object frame in prototype and parent chain to free.

Outputs: Returns a pointer to the first frame in the parent chain which could not be freed (resides in Flash memory) or H_NULL if the entire parent chain was freed.

Assumptions: The parent chain of an object frame frequently ends with a Flash-resident frame, either a frame in your app or the system frame. This routine does not attempt to free a frame which resides in Flash memory.

Side Effects: None

Availability: On AMS 2.00 and higher.

TI-89 / TI-92 Plus Differences: None

See Also: **OO_Destroy, OO_New**

Example:

```
pFrame obj;
   .
   .
   .
obj = OO_New(H_NULL);  /* allocate a new object */
   .
   . /* add more frames to parent chain of obj */
   .
OO_DestroyAll(obj);  /* free object and its parent and prototype chains */
```

OO_GetAppAttr

Declaration: void * **OO_GetAppAttr** (AppID *app*, ULONG *selector*)

Category(ies): Apps

Description: Retrieves an attribute of an app's object frame.

The prototype chain of each frame in the parent hierarchy beginning with the object frame of application *app* is searched until attribute *selector* is found.

Throws ER_ATTRIBUTE_NOT_FOUND error if attribute *selector* cannot be found.

Global variable **OO_SuperFrame** is updated with the parent of the frame where the attribute was located or NULL if the attribute was not found.

Inputs:

app — ID of application containing object frame where attribute search should begin.

selector — Attribute selector number.

Outputs: See description.

Assumptions: Rarely would you call **OO_GetAppAttr** directly. The FDL compiler (see section **7.3.3.3. Frame Description Language**) compiles attribute declarations into access macros with the call to **OO_GetAppAttr** containing the proper selector number and return type cast. You would call the macro instead of **OO_GetAppAttr**.

Side Effects: None

Availability: On AMS 2.00 and higher.

TI-89 / TI-92 Plus Differences: None

See Also: **OO_CondGetAttr, OO_GetAttr, OO_HasAttr, OO_SetAppAttr, OO_SetAttr**

Example:

```
int version;
version = (int const)OO_GetAppAttr(theApp, OO_APP_VERSION);

/* Better yet, call the macro generated by FDL: */
version = GetAppVersion(theApp);
```

OO_GetAttr

Declaration: void * **OO_GetAttr** (pFrame *obj*, ULONG *selector*)

Category(ies): Apps

Description: Retrieves an attribute of an object frame.

The prototype chain of each frame in the parent hierarchy beginning with *obj* is searched until attribute *selector* is found.

Throws ER_ATTRIBUTE_NOT_FOUND error if the attribute *selector* cannot be found.

Global variable **OO_SuperFrame** is updated with the parent of the frame where the attribute was located or NULL if the attribute was not found.

Inputs: *obj* — Object frame pointer.

selector — Attribute selector number.

Outputs: See description.

Assumptions: Rarely would you call **OO_GetAttr** directly. The FDL compiler (see section **7.3.3.3. Frame Description Language**) compiles attribute declarations into access macros with the call to **OO_GetAttr** containing the proper selector number and return type cast. You would call the macro instead of **OO_GetAttr**.

Side Effects: None

Availability: On AMS 2.00 and higher.

TI-89 / TI-92 Plus Differences: None

See Also: **OO_CondGetAttr, OO_GetAppAttr, OO_HasAttr, OO_SetAppAttr, OO_SetAttr**

Example:

```
char const *dateFormat;
dateFormat = (char const *)OO_GetAttr(MyAppObj, OO_DATE_FORMAT);

/* Better yet, call the macro generated by FDL: */
dateFormat = GetDateFormat(MyAppObj);
```

OO_HasAttr

Declaration: BOOL **OO_HasAttr** (pFrame *obj*, ULONG *selector*)

Category(ies): Apps

Description: Checks the inheritance hierarchy of object frame *obj* for the existence of attribute *selector*.

Inputs: *obj* — Object frame pointer.

selector — Attribute selector number.

Outputs: Returns TRUE if attribute *selector* exists in object frame *obj* or somewhere in its inheritance hierarchy. Otherwise returns FALSE.

Assumptions: None

Side Effects: None

Availability: On AMS 2.00 and higher.

TI-89 / TI-92 Plus Differences: None

See Also: **OO_CondGetAttr, OO_GetAppAttr, OO_GetAttr, OO_SetAppAttr, OO_SetAttr**

Example:

```
/* Does frame dispContextObj contain attribute OO_SPECIAL_FONT? */
if (OO_HasAttr(dispContextObj, OO_SPECIAL_FONT))
{
   /* Use special font */
   .
   .
   .
}
```

OO_InstallAppHook

Declaration: BOOL **OO_InstallAppHook** (AppID *appid*, pFrame *hookFrame*, pFrame * *retFrame*)

Category(ies): Apps

Description: This routine refines an application, overriding or adding attributes, by hooking a new frame at the head of the application's object frame parent hierarchy.

Many application attributes are hard-coded in Flash memory and cannot be changed — the app's table of strings, for example. A language localizer gains the effect of changing the app's string table by hooking a new frame containing the equivalent strings of a different language ahead of the app's object frame.

Multiple hooks can be installed in an app, the latest-installed having highest precedence during attribute look-up. Attribute search begins with the latest-installed hook frame and proceeds through each hook all the way back to the app's object frame until the attribute is found.

Hooks can be uninstalled in any order. See **OO_UninstallAppHook** or **OO_UninstallAppHookByName** to learn how to uninstall hooks.

Inputs:

appid — ID of application into which a new frame is to be hooked.

hookFrame — Pointer to hook frame.

retFrame — Returned pointer to RAM link frame. Keep this value around and pass it to **OO_UninstallAppHook** when you want to unhook the frame.

Outputs: Returns TRUE if hook was installed. Returns FALSE if memory for RAM link frame could not be allocated.

Assumptions: None

Side Effects: May cause heap compression.

Availability: On AMS 2.00 and higher.

TI-89 / TI-92 Plus Differences: None

See Also: **OO_InstallAppHookByName**, **OO_UninstallAppHook**, **OO_UninstallAppHookByName**

(continued)

OO_InstallAppHook *(continued)*

Example:

```
/* This single-attribute hook frame overrides the name of an app.
   The app will appear as "New App" in the [APPS] menu
*/

FRAME(NewAppName, OO_SYSTEM_FRAME, NULL, OO_APP_NAME, 1)
   ATTR(OO_APP_NAME, "New App")
ENDFRAME

pFrame hook;
AppID myappid;
   .
   .
   .
/* Get ID of the app to hook */
myappid = EV_getAppID((UCHAR const *)"MYAPP");

/* Hook new app name into app */
OO_InstallAppHook(myappid, (pFrame)&NewAppName, &hook);
```

OO_InstallAppHookByName

Declaration: BOOL **OO_InstallAppHookByName** (UCHAR const * *appname*, pFrame *hookFrame*, pFrame * *retFrame*)

Category(ies): Apps

Description: This routine refines an application, overriding or adding attributes, by hooking a new frame at the head of the application's object frame parent hierarchy.

This routine is like **OO_InstallAppHook** but takes an application name instead of an app ID as its first parameter.

Many application attributes are hard-coded in Flash memory and cannot be changed — the app's table of strings, for example. A language localizer gains the effect of changing the app's string table by hooking a new frame containing the equivalent strings of a different language ahead of the app's object frame.

Multiple hooks can be installed in an app, the latest-installed having highest precedence during attribute look-up. Attribute search begins with the latest-installed hook frame and proceeds through each hook all the way back to the app's object frame until the attribute is found.

Hooks can be uninstalled in any order. See **OO_UninstallAppHook** or **OO_UninstallAppHookByName** to learn how to uninstall hooks.

Inputs:

appname — Internal name of application into which a new frame is to be hooked.

hookFrame — Pointer to hook frame.

retFrame — Returned pointer to RAM link frame. Keep this value around and pass it to **OO_UninstallAppHookByName** or **OO_UninstallAppHook** when you want to unhook the frame.

Outputs: Returns TRUE if hook was installed. Returns FALSE if memory for RAM link frame could not be allocated or application could not be found.

Assumptions: None

Side Effects: May cause heap compression.

Availability: On AMS 2.00 and higher.

TI-89 / TI-92 Plus Differences: None

(continued)

OO_InstallAppHookByName *(continued)*

See Also: **OO_InstallAppHook, OO_UninstallAppHook, OO_UninstallAppHookByName**

Example:

```
/* This single-attribute hook frame overrides the name of an app.
   The app will appear as "New App" in the [APPS] menu
*/

FRAME(NewAppName, OO_SYSTEM_FRAME, NULL, OO_APP_NAME, 1)
   ATTR(OO_APP_NAME, "New App")
ENDFRAME

pFrame hook;
   .
   .
   .
/* Hook new name into app */
OO_InstallAppHookByName((UCHAR const *)"MYAPP", (pFrame)&NewAppName, &hook);
```

OO_InstallSystemHook

Declaration: BOOL **OO_InstallSystemHook** (pFrame *hookFrame*, pFrame * *retFrame*)

Category(ies): Apps, Operating System

Description: This routine refines the system object frame, overriding or adding attributes, by hooking a new frame at the head of the system object parent hierarchy.

Many system attributes are hard-coded in Flash memory and cannot be changed — the built-in table of strings, for example. A language localizer gains the effect of changing the system string table by hooking a new frame containing the equivalent strings of a different language ahead of the system object frame.

Multiple hooks can be installed over the system frame, the latest-installed having highest precedence during attribute look-up. Attribute search begins the with latest-installed hook frame and proceeds through each hook all the way back to the system object frame until the attribute is found.

Hooks can be uninstalled in any order. See **OO_UninstallSystemHook** to learn how to uninstall system hooks.

Inputs:

hookFrame — Pointer to hook frame.

retFrame — Returned pointer to RAM link frame. Keep this value around and pass it to **OO_UninstallSystemHook** when you want to unhook the frame.

Outputs: Returns TRUE if hook was installed. Returns FALSE if memory for RAM link frame could not be allocated.

Assumptions: None

Side Effects: May cause heap compression.

Availability: On AMS 2.00 and higher.

TI-89 / TI-92 Plus Differences: None

See Also: **OO_UninstallSystemHook**

(continued)

OO_InstallSystemHook *(continued)*

Example:

```
/* This single-attribute hook frame overrides the system date format. */

FRAME(DateHook, OO_SYSTEM_FRAME, NULL, OO_DATE_FORMAT, 1)
   ATTR(OO_DATE_FORMAT, "YYYY.MM.DD")
ENDFRAME

pFrame datehook;
   .
   .
   .
/* Hook new date format into system */
OO_InstallSystemHook((pFrame)&DateHook, &datehook);
```

OO_New

Declaration: pFrame **OO_New** (pFrame *prototype*)

Category(ies): Apps

Description: Allocates a new object frame from heap memory. Frame *prototype* is linked to the prototype hierarchy of the new frame.

Inputs: *prototype* — Pointer to prototype object. You may pass H_NULL for this parameter if you do not want to link the new object into a prototype hierarchy.

Outputs: Returns a frame pointer to the new object or H_NULL if memory for the object could not be allocated out of the heap.

Assumptions: None

Side Effects: May cause heap compression.

Availability: On AMS 2.00 and higher.

TI-89 / TI-92 Plus Differences: None

See Also: **OO_Destroy, OO_DestroyAll**

Example:

```
pFrame obj;
   .
   .
   .
obj = OO_New(H_NULL);  /* allocate a new object */
if (obj != H_NULL)
{
   .
   . /* use object */
   .
   OO_Destroy(obj);  /* free object */
}
```

OO_NextACB

Declaration: AppID **OO_NextACB** (AppID *acb*)

Category(ies): Apps

Description: Gets the ID of the next app after *acb* in the application control block list.

Inputs: *acb* — The ID of an app.

Outputs: Returns the ID of the next app after *acb* or H_NULL if *acb* is the last app in the application control block list.

Assumptions: The application control block list begins with all the built-in apps followed by Flash apps sorted in alphabetical order.

Side Effects: None

Availability: On AMS 2.00 and higher.

TI-89 / TI-92 Plus Differences: None

See Also: **OO_firstACB, OO_PrevACB**

Example:

```
AppID appid;
for (appid = OO_firstACB; appid != H_NULL; appid = OO_NextACB(appid))
{
   /* process each ACB */
   .
   .
   .
}
```

OO_PrevACB

Declaration: AppID **OO_PrevACB** (AppID *acb*)

Category(ies): Apps

Description: Gets the ID of the app before *acb* in the application control block list.

Inputs: *acb* — The ID of an app.

Outputs: Returns the ID of the app before *acb* or H_NULL if *acb* is the first app in the application control block list.

Assumptions: The application control block list begins with all the built-in apps followed by Flash apps sorted in alphabetical order.

Side Effects: None

Availability: On AMS 2.00 and higher.

TI-89 / TI-92 Plus Differences: None

See Also: **OO_firstACB, OO_NextACB**

Example:

```
AppID appid;
appid = OO_PrevACB(EV_runningApp);  /* get ID of app before me */
```

OO_SetAppAttr

Declaration: BOOL **OO_SetAppAttr** (AppID *app*, ULONG *selector*, void * *value*)

Category(ies): Apps

Description: Sets the value of an attribute in an app's object frame. Changes the value if the attribute exists or adds a new slot if the attribute does not exist.

If the app frame is marked read-only (as is the case with Flash-resident frames), a new frame will be allocated in heap memory and linked at the head of the app frame's parent hierarchy. The new attribute value is then placed in the RAM frame.

Inputs:

app — ID of application containing object frame where attribute search should begin.

selector — Attribute selector number.

value — New value for attribute. The value may be any integer or pointer which fits in 32 bits. Integer values must be cast to (void *) to avoid compiler warning messages.

Outputs: Returns TRUE if the attribute value was updated. This routine will return FALSE if it runs out of heap memory while attempting to expand the frame to add a new attribute slot or link a new frame into the app frame's parent hierarchy.

Assumptions: Rarely would you call **OO_SetAppAttr** directly. The FDL compiler (see section **7.3.3.3. Frame Description Language**) compiles attribute declarations into access macros with the call to **OO_SetAppAttr** containing the proper selector number, value and return type casts. You would call the macro instead of **OO_SetAppAttr**.

Side Effects: May cause heap compression.

Availability: On AMS 2.00 and higher.

TI-89 / TI-92 Plus Differences: None

See Also: **OO_CondGetAttr, OO_GetAppAttr, OO_GetAttr, OO_HasAttr, OO_SetAttr**

Example:

```
/* Set app's default menu */
OO_SetAppAttr(EV_runningApp, OO_APP_DEFAULT_MENU, &menu);

/* Better yet, call the macro generated by FDL: */
SetAppDefaultMenu(EV_runningApp, &menu);

/* Actually, there is a routine in the OS which does exactly this. . . . */
EV_registerMenu(&menu);
```

OO_SetAttr

Declaration: BOOL **OO_SetAttr** (pFrame *obj*, ULONG *selector*, void * *value*)

Category(ies): Apps

Description: Sets the value of an attribute in an object frame. If the frame is marked read-only, attribute slot search continues in the frame's parent object.

Inputs:

obj — Object frame where attribute search should begin.

selector — Attribute selector number.

value — New value for attribute. The value may be any integer or pointer which fits in 32 bits. Integer values must be cast to (void *) to avoid compiler warning messages.

Outputs: Returns TRUE if the attribute value was updated. This routine will return FALSE if all the frames in the object hierarchy are read-only or it runs out of heap memory while attempting to expand the frame to add a new attribute slot.

Assumptions: Rarely would you call **OO_SetAttr** directly. The FDL compiler (see section **7.3.3.3. Frame Description Language**) compiles attribute declarations into access macros with the call to **OO_SetAttr** containing the proper selector number, value and return type casts. You would call the macro instead of **OO_SetAttr**.

Side Effects: May cause heap compression.

Availability: On AMS 2.02 and higher.

TI-89 / TI-92 Plus Differences: None

See Also: **OO_CondGetAttr, OO_GetAppAttr, OO_GetAttr, OO_HasAttr, OO_SetAppAttr**

Example:

```
/* Set running app's default menu */
ACB *pacb = HeapDeref(EV_runningApp);
OO_SetAttr(pacb->appData, OO_APP_DEFAULT_MENU, &menu);

/* OO_SetAttr will give up if all frames in the app's parent hierarchy are marked
read-only. It is better to call OO_SetAppAttr: */
OO_SetAppAttr(EV_runningApp, OO_APP_DEFAULT_MENU, &menu);

/* Better yet, call the macro generated by FDL: */
SetAppDefaultMenu(EV_runningApp, &menu);

/* Actually, there is a routine in the OS which does exactly this. . . */
EV_registerMenu(&menu);
```

OO_UninstallAppHook

Declaration: BOOL **OO_UninstallAppHook** (AppID *appid*, pFrame *hookFrame*)

Category(ies): Apps

Description: This routine uninstalls an application hook previously installed with a call to **OO_InstallAppHook** or **OO_InstallAppHookByName**.

Inputs:

appid — ID of app from which to remove the hook.

hookFrame — Pointer to the frame to unhook. Use the hook frame value returned from **OO_InstallAppHook** or **OO_InstallAppHookByName**.

Outputs: Returns TRUE if the hook was found and removed from the app's list of hooks. Otherwise, returns FALSE.

Assumptions: This routine frees the memory occupied by the hook.

Side Effects: None

Availability: On AMS 2.00 and higher.

TI-89 / TI-92 Plus Differences: None

See Also: **OO_InstallAppHook, OO_InstallAppHookByName, OO_UninstallAppHookByName**

Example:

```
pFrame hook;
   .
   .
   .
/* hook contains pointer stored by call to OO_InstallAppHook */
OO_UninstallAppHook(appid, hook);
```

OO_UninstallAppHookByName

Declaration: BOOL **OO_UninstallAppHookByName** (UCHAR const * *appname*, pFrame *hookFrame*)

Category(ies): Apps

Description: This routine uninstalls an application hook previously installed with a call to **OO_InstallAppHook** or **OO_InstallAppHookByName**.

Inputs:

appname — Internal name of application from which to remove the hook.

hookFrame — Pointer to the frame to unhook. Use the hook frame value returned from **OO_InstallAppHook** or **OO_InstallAppHookByName**.

Outputs: Returns TRUE if the hook was found and removed from the app's list of hooks. Returns FALSE if the application could not be found or the hook could not be found in the application's hook list.

Assumptions: This routine frees the memory occupied by the hook.

Side Effects: None

Availability: On AMS 2.00 and higher.

TI-89 / TI-92 Plus Differences: None

See Also: **OO_InstallAppHook, OO_InstallAppHookByName, OO_UninstallAppHook**

Example:

```
pFrame hook;
   .
   .
   .
/* hook contains pointer stored by call to OO_InstallAppHookByName */
OO_UninstallAppHookByName((UCHAR const *)"MYAPP", hook);
```

OO_UninstallSystemHook

Declaration: BOOL **OO_UninstallSystemHook** (pFrame *hookFrame*)

Category(ies): Apps, Operating System

Description: This routine uninstalls a system hook previously installed with a call to **OO_InstallSystemHook**.

Inputs: *hookFrame* — Pointer to the frame to unhook. Use the hook frame value returned from **OO_InstallSystemHook**.

Outputs: Returns TRUE if the hook was found and removed from the system hook list. Otherwise, returns FALSE.

Assumptions: This routine frees the memory occupied by the hook.

Side Effects: None

Availability: On AMS 2.00 and higher.

TI-89 / TI-92 Plus Differences: None

See Also: **OO_InstallSystemHook**

Example:

```
pFrame hook;
   .
   .
   .
/* hook contains pointer stored by call to OO_InstallSystemHook */
OO_UninstallSystemHook(hook);
```

Appendix A: System Routines — Certificates

freeIdList

Declaration: void **freeIdList** (void)

Category(ies): Certificates, Link

Description: If ID list exists in memory, it is released and deleted.

Inputs: None

Outputs: None

Assumptions: None

Side Effects: None

Availability: On AMS 2.00 and higher.

TI-89 / TI-92 Plus Differences: None

See Also: **SendIdList**

Example:

```
if (DlgMessage("VAR-LINK", "Clear ID List?", PDB_YES, PDB_NO) == KB_ENTER)
      freeIdList();
```

LIO_SendIdList

Declaration: WORD **LIO_SendIdList** (WORD *DoDelete*)

Category(ies): Certificates, Link

Description: If ID list does not exist, creates one using the calculator's ID. If the ID list does exist, the calculator ID is appended. This list is then sent over the link port.

Inputs: *DoDelete* — If TRUE, ID list is deleted from memory after it is sent.

Outputs: Returns non-0 if an error occurs. Otherwise returns 0.

Assumptions: None

Side Effects: None

Availability: On AMS 2.00 and higher.

TI-89 / TI-92 Plus Differences: None

See Also: **freeIDList**

Example:

```
if(LIO_SendIdList(TRUE))
      goto RetErr;
```

Appendix A: System Routines — Data Utilities

See Also:

DataTypeNames

Declaration: BYTE * **DataTypeNames** (BYTE *Tag*)

Category(ies): Data Utilities

Description: Given a data *Tag*, return the full string that represents the type of that data. This string is localized for the current language.

Inputs: *Tag* — GDB_VAR_TAG, PIC_VAR_TAG, TEXT_VAR_TAG, DATA_VAR_TAG, MATRIX_TAG, LIST_TAG, FUNC_BEGIN_TAG, PRGM_TAG, SYSVAR_TAG, STR_DATA_TAG, GEO_FILE_TAG, GEO_MACRO_TAG, ASM_PRGM_TAG, EQUATION_TAG, GEN_DATA_TAG.

Outputs: Pointer to a static string representing the type of the *Tag* passed.

Assumptions: None

Side Effects: None

Availability: On AMS 2.00 and higher.

TI-89 / TI-92 Plus Differences: None

See Also: **SmapTypeStrings**

Example:

```
sprintf( buf, "This variable is a %s", DataTypeNames( Tag ) );
```

gen_version

Declaration: unsigned char **gen_version** (EStackIndex *i*, EStackIndex *j*)

Category(ies): Data Utilities

Description: This function is used to generate a version number to be stored in the version field of an object's symbol table entry.

Inputs:
i — EStackIndex of the object for which a version number is needed.

j — NULL_INDEX when *i* is terminated by END_OF_SEGMENT_TAG. Otherwise, EStackIndex of the byte following the last byte of *i*.

Outputs: Returns one of the following values: TV_TI_92, TV_PARM, TV_SPAM, TV_CRAM, TV_3RDPARTYAPP, TV_SCRAM.

Assumptions: Either the object indexed by *i* is terminated by an END_OF_SEGMENT_TAG, or the object is delimited by *i* and *j*.

Side Effects: None

Availability: All versions of the TI-89 / TI-92 Plus.

TI-89 / TI-92 Plus Differences: None

See Also: None

Example:

If m indexes the bolded tag in the expression x+2 and n indexes the next tag below the expression as follows
␣ X_VAR_TAG 2 1 NONNEGATIVE_INTEGER_TAG **ADD_TAG**

then

```
unsigned char ver = gen_version (m, n);
```

assigns the version number of the expression to the variable ver.

GetDataType

Declaration: short **GetDataType** (EStackIndex *i*)

Category(ies): Data Utilities

Description: Given a pointer to the data tag for a variable, return the data type value (as defined in Outputs).

Inputs: *i* — Pointer to a tag.

Outputs: AMS data type value: SDT_ASM, SDT_DATA, SDT_EXPR, SDT_FIG, SDT_FUNC, SDT_GDB, SDT_LIST, SDT_MAC, SDT_MAT, SDT_MAT, SDT_OTH, SDT_PIC, SDT_PRGM, SDT_STR, SDT_TEXT.

Assumptions: None

Side Effects: None

Availability: On AMS 2.00 and higher.

TI-89 / TI-92 Plus Differences: None

See Also: **DataTypeNames, SmapTypeStrings**

Example:

```
/* Given a pointer to a variable name (CAS format, points to zero byte terminator),
   return that variable's type in the SDT_ASM . . . SDT_TEXT format.
   Return –1 if variable not found or has no value.
*/
short getVarType( BYTE *varNamePtr ) {
   HSYM hsym;
   SYM_ENTRY *SymPtr;
   HANDLE h;

   if (hsym = VarRecall( varNamePtr, 0)) {
      SymPtr = DerefSym( hsym );
      if (h = SymPtr->hVal)
         return( GetDataType( HToESI(h) ));
   }
   return –1;
}
```

GetFuncPrgmBodyPtr

Declaration: EStackIndex **GetFuncPrgmBodyPtr** (EStackIndex *userDefTagPtr*)

Category(ies): Data Utilities

Description: Given a pointer to a USER_DEF_TAG, return the pointer to the function or program body — the parameters and flags are skipped.

Inputs: *userDefTagPtr* — Pointer to the USER_DEF_TAG of a program or function.

Outputs: Pointer to the program or function body.

Assumptions: None

Side Effects: None

Availability: On AMS 2.00 or higher.

TI-89 / TI-92 Plus Differences: None

See Also: None

Example: See **SymFindMain**.

QSysProtected

Declaration: BOOL **QSysProtected** (BYTE *Tag*)

Category(ies): Data Utilities

Description: Return TRUE if the given *Tag* is for a system protected data type.

Inputs: *Tag* — Tag byte.

Outputs: TRUE if *Tag* is that for a function, program, data-var, graph database, picture, text, assembly language program, or third-party data tag (which includes files).

Assumptions: None

Side Effects: None

Availability: All versions of the TI‑89 / TI‑92 Plus.

TI‑89 / TI‑92 Plus Differences: None

See Also: **GetDataType**

Example:

```
/* Given a pointer to a variable name (CAS format, points to zero byte
   terminator), return TRUE if O.K. to write to it
*/
getVarType( BYTE *varNamePtr ) {
   HSYM hsym;
   SYM_ENTRY *SymPtr;

   if (hsym = VarRecall( varNamePtr, 0)) {
      SymPtr = DerefSym( hsym );
      return( !QSysProtected( * HToESI(SymPtr->hVal) ));
   }  else
      return TRUE;
}
```

SmapTypeStrings

Declaration: char * **SmapTypeStrings** (short *typeNum*);

Category(ies): Data Utilities

Description: Given the numeric type number of a variable (as returned by **GetDataType**), return the short (maximum four characters) string representing the type of this variable. This is the string displayed in VAR-LINK. This string is localized for the current language.

Inputs: SDT_ASM, SDT_DATA, SDT_EXPR, SDT_FIG, SDT_FUNC, SDT_GDB, SDT_LIST, SDT_MAC, SDT_MAT, SDT_MAT, SDT_OTH, SDT_PIC, SDT_PRGM, SDT_STR, SDT_TEXT

Outputs: Pointer to a static string representing the data type of *typeNum*. Note that files (SDT_OTH) will return “OTH” and not the file type specified when the file was opened.

Assumptions: None

Side Effects: None

Availability: All versions of the TI-89 / TI-92 Plus.

TI-89 / TI-92 Plus Differences: None

See Also: **GetDataType, DataTypeNames**

Example:

```
HANDLE hDat;
EStackIndex esi;
SYM_ENTRY *SymPtr;
BYTE szBuf[18];
.
.
.
/* Assuming szBuf contains the name of the variable, this code will tack
   the 4 char (max) type onto szBuf.
*/
if (hDat = SymPtr->hVal) {
   esi = HToESI( hDat );   /* point to data type tag */
   strncpy( (char *) szBuf+13, SmapTypeStrings(GetDataType( esi )), 4 );
}
```

Appendix A: System Routines — Dialog

Dialog

Declaration: WORD **Dialog** (DIALOG * *Dlg*, short *x0*, short *y0*, char * *FieldBuf*, WORD * *OptionList*)

Category(ies): Dialog

Description: Open a dialog box and handle all keys pressed by the user until the dialog box is closed, returning any modified dialog box items in *FieldBuf* or *OptionList*. *Dlg* points to a DIALOG structure as built by the resource compiler or dynamically by **DialogNew** and **DialogAdd**.

Inputs:

Dlg — Pointer to a DIALOG structure.

x0, *y0* — Specifies the upper left corner of the dialog. If *x0* is equal to -1 then the dialog box is centered horizontally. If *y0* is equal to -1 then the dialog box is centered vertically.

FieldBuf — Points to a buffer for any edit fields in the dialog box or NULL if there are no edit fields. The indexes into *FieldBuf* are specified by each edit field.

OptionList — An array of WORDs, with an entry for each pop-up field. The initial value for each of these fields as well as the value selected by the user is stored in this array. The index into *OptionList* is specified by each pop-up field.

Note that both *FieldBuf* and *OptionList* are modified by **Dialog** whether the user presses [ENTER] to accept the dialog box or [ESC] to cancel it. Copies of either structure are not made. It is left to the caller to use the changes in these structures if the dialog box is accepted or to toss them if the dialog box is canceled.

Outputs:

KB_ENTER — User pressed [ENTER] to close dialog box.

KB_ESC — User pressed [ESC] to close dialog box.

DB_MEMFULL — Not enough memory to open the menu for the dialog box.

Assumptions: If *Dlg* points to a dynamically created dialog box, the heap block that stores the DIALOG structure must be locked.

Side Effects: May cause heap compression.

Availability: All versions of the TI-89 / TI-92 Plus.

(continued)

Dialog *(continued)*

TI-89 / TI-92 Plus Differences: None

See Also: **DialogAdd**, **DialogNew**, Resource Compiler

Example:

```
DWORD NoCallBack( WORD DlgId, DWORD Value ) {
   return TRUE;
}

void TestDialog( void ) {
   WORD opts[3];
   char buf[22];
   char outStr[256];

   strcpy( buf, "FIRST" );  /* default edit strings */
   strcpy( buf+11, "SECOND" );
   opts[0] = opts[1] = 1; /* default to 1st pop-up item */
   opts[2] = 2; /* default to 2nd pop-up item */
   if (KB_ENTER == Dialog( &tDialog, -1, -1, buf, opts )) {
      sprintf( outStr, "Edit1: %s\nEdit 2: %s\nPopup1: %d\nPopup2: %d\nPopup3: %d",
               buf, buf+11, opts[0], opts[1], opts[2] );
      DlgNotice( "tDialog", outStr );
   }
}

DIALOG tDialog, 180, 90, NoCallBack {
   SCROLL_REGION, {DF_CLR_ON_REDRAW, 12, 29}, 175,69, 3,5,  2,3, 9
   EDIT,   {DF_SCROLLABLE|DF_TAB_SPACES, 12, 30}, "EDIT1", 0, 10, 11
   EDIT,   {DF_SCROLLABLE|DF_TAB_SPACES, 12, 40}, "EDIT2", 11, 10, 11
   POPUP,     {DF_SCROLLABLE|DF_TAB_ELLIPSES, 12, 50}, "FIRST POPUP", Popup1, 0
   POPUP,     {DF_SCROLLABLE|DF_TAB_ELLIPSES, 12, 60}, "2ND POPUP", Popup2, 1
   POPUP,     {DF_SCROLLABLE|DF_TAB_ELLIPSES, 12, 70}, "3RD POPUP", Popup2, 2
   MENU,   {0, 12, 11}, MenuPages
   HEADER,   {0, 0, 0}, "DIALOG HEADER", PDB_OK, PDB_CANCEL
}

POPUP Popup1, RC_NO_IDS, 0 {
   "Item 1-1", 1
   "Item 1-2", 2
}

POPUP Popup2, RC_NO_IDS, 0 {
   "Item 2-1", 1
   "Item 2-2", 2
   "Item 2-3", 3
}

TOOLBOX MenuPages, RC_NO_IDS, 0, 120 {
   "PAGE 1", 1
   "PAGE 2", 2
   "PAGE 3", 3
}
```

DialogAdd

Declaration: HANDLE **DialogAdd** (HANDLE *dH*, WORD *Flags*, WORD *x*, WORD *y*, WORD *Type*, . . .)

Category(ies): Dialog

Description: Add an item to a dynamic dialog box. See section **11.4. Dialog Boxes** for a description of the *Flags*, *Type*, and additional parameter fields.

Inputs:

dH — HANDLE created by **DialogNew**.

Flags — Flags for new item.

x, y — Coordinates relative to dialog box of new item.

Type — One of the following items which determines the parameters following *Type*.

D_DYNPOPUP
    char **TextPtr*, HANDLE (**GetPopUp*) (WORD), WORD *oIndex*
D_EDIT_FIELD
    char **TextPtr*, WORD *bOffset*, WORD *Flen*, WORD *Dlen*
D_HEADER
    char **TextPtr*, WORD *lButton*, WORD *rButton*
D_HEDIT
    char **TextPtr*, WORD *Dlen*
D_HPOPUP
    char **TextPtr*, HANDLE *hPopUp*, WORD *oIndex*
D_MENU
    MENU **menuPtr*, WORD *MaxMenuWidth*
D_POPUP
    char **TextPtr*, MENU **PopUp*, WORD *oIndex*
D_SCROLL_REGION
    WORD *x1*, WORD *y1*, WORD *Index0*, WORD *Index1*, WORD *NumDspFields*, WORD *TotNumFields*, WORD *FieldHeight*
D_TEXT
    char **TextPtr*
D_XFLAGS
    WORD *xFlags1*, *xFlags2*, *xFlags3*, *xFlags4*

Outputs: *dH* if successful or H_NULL if not enough memory.

(continued)

DialogAdd *(continued)*

Assumptions: *dH* was created by **DialogNew**.

Side Effects: May cause heap compression.

Availability: All versions of the TI‑89 / TI‑92 Plus.

TI‑89 / TI‑92 Plus Differences: None

See Also: **DialogDo, DialogNew, Dialog**

Example: See **DialogNew**.

DialogDo

Declaration: WORD **DialogDo** (HANDLE *DialogHandle*, short *x0*, short *y0*, char * *FieldBuf*, WORD * *OptionList*)

Category(ies): Dialog

Description: Works like **Dialog** only for dynamically created dialog boxes. Instead of being passed a pointer to a dialog structure, this routine uses the handle of a dynamically created dialog box.

Inputs: *DialogHandle* — HANDLE created by **DialogNew** and modified with **DialogAdd**.

The rest of the parameters are the same as in the **Dialog** function.

Outputs: Same as **Dialog**.

Assumptions: None

Side Effects: May cause heap compression.

Availability: All versions of the TI-89 / TI-92 Plus.

TI-89 / TI-92 Plus Differences: None

See Also: **Dialog, DialogNew, DialogAdd**

Example: See **DialogNew**.

DialogNew

Declaration: HANDLE **DialogNew** (WORD *Width*, WORD *Height*, DWORD *CallBack* (WORD, DWORD));

Category(ies): Dialog

Description: Create a dynamic dialog box. Use **DialogAdd** to add items to the dialog box and **DialogDo** to execute it.

Inputs:

Width — Width in pixels of dialog box or 0 for a dynamically calculated width.

Height — Height in pixels of dialog box or 0 for a dynamically calculated height.

CallBack — Address of call-back routine (must be provided).

Outputs: HANDLE to an empty DIALOG structure that can be added to with **DialogAdd**.

Assumptions: None

Side Effects: May cause heap compression.

Availability: All versions of the TI‑89 / TI‑92 Plus.

TI‑89 / TI‑92 Plus Differences: None

See Also: **DialogAdd, DialogDo**

(continued)

DialogNew *(continued)*

Example:

```
static WORD Opts[3];
static char Buf[11];
static HANDLE hP1, hP2;

HANDLE CallBackH( WORD dlgId ) { return hP1; }

DWORD CallBackD3( WORD dlgId, DWORD Value ) {
WORD Key;

switch( dlgId ) {
   case 0:  /* Owner draw */
      Key = DlgMessage("TITLE", "Owner draw popup\nPress ESC to exit
                       dialog\nENTER to continue",
      PDB_OK, PDB_CANCEL );
      if (KB_ESC == Key)
         return DB_EXIT;
      break;
   case 1:  /* Exit if #20 selected in Opts[1] */
      if (Opts[1] == 20)
         return DB_EXIT;
      break;
   case 2:  /* Exit if "EXIT" entered */
      if (0 == strcmp("EXIT", Buf))
         return DB_EXIT;
      break;
   case 3:  /* Exit if #20 selected in Opts[2] */
      if (Opts[2] == 20)
        return DB_EXIT;
      break;
   case 4:  /* Exit if F3 pressed */
      Key = 0xFFFF & Value;
      switch( Key ) {
         case KB_F1: DlgMessage("MENU","F1 Pressed,going to entry 0",PDB_OK, 0 );
                                return 0;
         case KB_F2: DlgMessage("MENU","F2 Pressed,going to entry 1",PDB_CANCEL,0 );
                                return 1;
         case KB_F3: return DB_EXIT;
      }
      break;
   case DB_QACTIVE:
      return TRUE;
   }
return TRUE;
}
```

(continued)

DialogNew *(continued)*

```
void TestD3( void ) {
static HANDLE hD1;
static HANDLE hM1;
static WORD Key;

if (hM1 = MenuNew(0,0,0)) {
   MenuAddText( hM1, -1, "MENU1", 0, 0 );
   MenuAddText( hM1, -1, "MENU2", 0, 0 );
   MenuAddText( hM1, -1, "EXIT", 0, 0 );
   if (MenuFlags(hM1) & MF_ERROR) {
      HeapFree(hM1);
      return;
   }
}
if (hP1 = PopupNew(NULL,0)) {
   PopupAddText( hP1, -1, "POPUP ENTRY 1", 10 );
   PopupAddText( hP1, -1, "EXIT DIALOG", 20 );
   PopupAddText( hP1, -1, "POPUP ENTRY 3", 30 );
   if (MenuFlags(hP1) & MF_ERROR)
      return;
}
if (hP2 = PopupNew(NULL,0)) {
   PopupAddText( hP2, -1, "DUMMY ENTRY", 0 );
   if (MenuFlags(hP2) & MF_ERROR)
      return;
}
if (hD1 = DialogNew( 0,0, CallBackD3 )) {
   DialogAdd( hD1, DF_OWNER_DRAW, 8, 28, D_HPOPUP, "Owner draw", hP2, 0 );
   DialogAdd( hD1, 0, 8, 38, D_HPOPUP, "DB_EXIT", hP1, 1 );
   DialogAdd( hD1, 0, 8, 48, D_EDIT_FIELD, "Type 'EXIT' to quit", 0, 10, 10 );
   DialogAdd( hD1, 0, 8, 58, D_DYNPOPUP, "DYN_POPUP", &CallBackH, 2 );
   DialogAdd( hD1, 0, 0, 0, D_MENU, (MENU *) HLock(hM1), 0 );
}  else
   return;
Opts[0] = 10;
Opts[1] = 30;
Opts[2] = 30;
strcpy( Buf, "TEST" );
do {
   DialogDo( hD1, -1, -1, &Buf[0], &Opts[0] );
   Key = DlgMessage("TITLE", "Dialog ended, press ESC to completely exit or ENTER
                    to try again", PDB_OK, PDB_CANCEL );
} while (KB_ESC != Key);
DialogFree( hD1 );
PopupFree( hP1 );
PopupFree( hP2 );
HeapFree( hM1 );
}
```

DlgMessage

Declaration: WORD **DlgMessage** (const char * *Title*, const char * *Message*, WORD *lButton*, WORD *rButton*)

Category(ies): Dialog

Description: Issue a dialog with a given *Title* and a word-wrapped *Message*. The *Message* string may contain newline constants. The dialog box will be sized to fit the screen with a predefined width for the TI-89 and the TI-92 Plus.

Inputs:

Title	—	String pointer for title of dialog box (no title if NULL).
Message	—	String pointer message to be word wrapped in dialog box.
lButton, *rButton*	—	One of the predefined button constants: PDB_OK, PDB_SAVE, PDB_YES, PDB_CANCEL, PDB_NO, PDB_GOTO which define the buttons on the bottom of the dialog box. *rButton* may be 0 if only one button is needed.

Outputs:

KB_ENTER	—	User pressed ENTER to close dialog box.
KB_ESC	—	User pressed ESC to close dialog box.

Assumptions: If there is not enough memory for the dialog box, a low memory version will be used (no word wrapping); so this dialog will always succeed.

Side Effects: May cause the heap to be compressed.

Availability: All versions of the TI-89 / TI-92 Plus. However, on AMS 2.04 and higher, word wrap also occurs on commas and spaces.

TI-89 / TI-92 Plus Differences: None

See Also: **Dialog**

Example:

```
if (KB_ENTER == DlgMessage("ERROR: Writing to setup file", "Delete setup file and
                           recreate it?", PDB_YES, PDB_NO))
   FDelete( "ZSETUP" );
```

DrawStaticButton

Declaration: void **DrawStaticButton** (WINDOW * *dWin*, WORD *Type*, SWORD *x*)

Category(ies): Dialog

Description: This is a utility routine to draw dialog box style buttons at the bottom of a window.

Inputs:

dWin — Pointer to an open WINDOW.

Type — PDB_OK, PDB_SAVE, PDB_YES, PDB_CANCEL, PDB_NO, PDB_GOTO.

x — X window coordinate (buttons are always drawn at the bottom of the window).

Outputs: None

Assumptions: None

Side Effects: None

Availability: All versions of the TI‑89 / TI‑92 Plus.

TI‑89 / TI‑92 Plus Differences: None

See Also: None

(continued)

DrawStaticButton *(continued)*

Example:

```
/* The MEM screen is not a dialog box, even though it looks like one. It uses
   MenuBegin to draw its menu and GKeyIn to get keys (it is not part of the
   event loop). It uses DrawStaticButton to draw the ENTER button. It also uses
   DlgMessage to prompt the user.
*/
if (WinOpen( &w, MakeWinRect(13,8,227,120), WF_ROUNDEDBORDER |
    WF_SAVE_SCR | WF_TITLE | WF_SYS_ALLOC, XR_stringPtr(XR_MEMORY))) {
.
.
.
   DrawStaticButton( &w, PDB_OK, 5 );
   if (h = MenuBegin(&ResetMenu, 15,12, MBF_SYS_ALLOC)) {
      DrawWinBorder( &w, &w.Window );  /* fixup title line */
      do {
         Key = GKeyIn( 0l, GKF_MODAL );
         if (Key == KB_F1) {
            if ((Key = MenuKey( h, KB_F1 )) == KB_ESC)
               Key = 0;
            if (Key >= MR_RAM_ALL && Key <= MR_ALL_MEMORY) {
               i = (short) DlgMessage( XR_stringPtr(XR_RESET),
               XR_stringPtr(ResetTitles[Key-1]), PDB_YES, PDB_NO );
               if ((WORD) i != KB_ENTER)
                  continue;
               MenuEnd( h );
               WinClose( &w );
   .
   .
   .
```

ERD_dismissNotice

Declaration: void **ERD_dismissNotice** (void)

Category(ies): Dialog, Error Handling

Description: Remove dialog box displayed by previous call to **ERD_notice**.

Inputs: None

Outputs: None

Assumptions: None

Side Effects: May cause underlying windows to repaint.

Availability: On AMS 2.00 and higher.

TI-89 / TI-92 Plus Differences: None

See Also: **ERD_notice**

Example:

```
BOOL bNotice;
bNotice = ERD_notice((UCHAR *)"WARNING", (UCHAR *)"Database reorganization in
                     progress -- do not interrupt!");
  .
  . /* reorganize database */
  .
if (bNotice)
  ERD_dismissNotice();
```

ERD_notice

Declaration: BOOL **ERD_notice** (UCHAR const * *title*, UCHAR const * *msg*)

Category(ies): Dialog, Error Handling

Description: Displays text of *msg* in a pop-up window. The window stays on the screen after returning to the caller.

Use this routine to notify the user that a long-running operation has begun. Call **ERD_dismissNotice** to remove the pop-up window when the operation is complete.

Inputs: *title* — Window title.

msg — Text of message.

Outputs: Returns TRUE if the pop-up window is displayed. Returns FALSE if the window could not be opened.

Assumptions: The text of *msg* is word-wrapped as necessary to fit in the width of the message box.

Side Effects: May cause heap compression.

Availability: On AMS 2.00 and higher.

TI-89 / TI-92 Plus Differences: None

See Also: **ERD_dismissNotice**

Example:

```
BOOL bNotice;
bNotice = ERD_notice((UCHAR *)"WARNING", (UCHAR *)"Database reorganization in
                     progress -- do not interrupt!");
   .
   . /* reorganize database */
   .
if (bNotice)
   ERD_dismissNotice();
```

VarNew

Declaration: HSYM **VarNew** (BYTE * *Types* {, BYTE * *subTypeList* [ ]})

Category(ies): Dialog

Description: Creates a standard NEW dialog box (go to the Program Editor and do a F1, 3 (New . . .) as an example). The user may select from a list of types to create as well as the folder to create it in and then may type in a variable name.

Inputs:

Types — An array, terminated by zero, of the following types: GDB_VAR_TAG, PIC_VAR_TAG, TEXT_VAR_TAG, DATA_VAR_TAG, MATRIX_TAG, LIST_TAG, FUNC_BEGIN_TAG, PRGM_TAG, STR_DATA_TAG, EQUATION_TAG, GEN_DATA_TAG. Each value in the *Types* list will be presented to the user in a drop-down (unless there is only one value in the list then it will be a static field).

subTypeList (optional) — For each GEN_DATA_TAG byte in the *Types* list, there must be a string pointer in the *subTypeList* that points to the text to display in the drop down. This parameter may be left off if there are no GEN_DATA_TAGs in the *Types* list.

Outputs: The HSYM of the newly created symbol or H_NULL if the user presses ESC or there is an error creating the symbol.

Assumptions: None

Side Effects: May cause heap compression.

Availability: All versions of the TI-89 / TI-92 Plus.

TI-89 / TI-92 Plus Differences: None

See Also: **VarOpen, VarSaveAs**

(continued)

VarNew *(continued)*

Example:

```
static BYTE const Tag1[] = { GEN_DATA_TAG, GEN_DATA_TAG, 0 };
static const char * const TagS1[] = {"FILE", "DAT", NULL};
HSYM hsym;

FCreate( "f1", "FILE" );
FCreate( "f2", "FILE" );
FCreate( "d1", "DAT" );
FCreate( "d2", "DAT" );
hsym = VarSaveAs( (BYTE *) &Tag1, (BYTE *) "CURRENT", &TagS1 );
hsym = VarNew( (BYTE *) &Tag1, &TagS1 );
.
.
.
```

VarOpen

Declaration: HSYM **VarOpen** (BYTE * *Types* {, BYTE * *subTypeList* [ ]})

Category(ies): Dialog, Variables

Description: Creates a standard OPEN dialog box (go to the Program Editor and do a F1, 1 (Open . . .) as an example). The user may select from a list of types to open as well as the folder to look in and finally is presented with a drop-down of symbols in the selected folder that are of the selected type.

Inputs:

Types — An array, terminated by zero, of the following types: GDB_VAR_TAG, PIC_VAR_TAG, TEXT_VAR_TAG, DATA_VAR_TAG, MATRIX_TAG, LIST_TAG, FUNC_BEGIN_TAG, PRGM_TAG, STR_DATA_TAG, EQUATION_TAG, GEN_DATA_TAG. Each value in the *Types* list will be presented to the user in a drop-down (unless there is only one value in the list then it will be a static field).

subTypeList (optional) — For each GEN_DATA_TAG byte in the *Types* list, there must be a string pointer in the *subTypeList* that points to the text to display in the drop down. This parameter may be left off if there are no GEN_DATA_TAGs in the *Types* list.

Outputs: The HSYM of the selected symbol or H_NULL if the user presses ESC or there is an error creating the symbol.

Assumptions: None

Side Effects: May cause heap compression.

The first WORD in the global array, **VarOptList**, is set to the index of the type selected when *Types* contains multiple values. It will be one if the first entry was selected, two for the second entry, and so on. See the example below. **VarOptList** is available on AMS 2.00 and higher.

Availability: All versions of the TI-89 / TI-92 Plus.

TI-89 / TI-92 Plus Differences: None

See Also: **VarNew, VarOpen, VarSaveAs**

(continued)

VarOpen *(continued)*

Example:

```
/* This is a code fragment from the grapher that is executed when F1,1 (Open) is
   selected from the graph menu
*/
HSYM hs;
BYTE Tag, *Ptr;
BYTE SaveTypes[] = { GDB_VAR_TAG, PIC_VAR_TAG, 0 };

if (hs = VarOpen( (BYTE *) SaveTypes )) {
   Ptr = HeapDeref(DerefSym(hs)->hVal);
   Tag = (VarOptList[0] == 2 ? PIC_VAR_TAG : GDB_VAR_TAG);
   if (Tag == PIC_VAR_TAG) {
      WinBitmapPut(gr_active->grwinp, 0, 0, (BITMAP *) (Ptr + 2), A_OR );
   else
      /* . . . recall graph database . . . */
}
```

VarSaveAs

Declaration: HSYM **VarSaveAs** (BYTE * *Types*, BYTE * *Current*, {, BYTE * *subTypeList* [ ]})

Category(ies): Dialog, Variables

Description: Creates a standard SAVE COPY OF [*Current*] AS dialog box (go to the Program Editor and do a F1, 2 (Save Copy As . . .) as an example). The user may select from a list of types to save as well as the folder to save the symbol in and finally is presented with an edit box to enter the symbol name.

Inputs:

Types — An array, terminated by zero, of the following types: GDB_VAR_TAG, PIC_VAR_TAG, TEXT_VAR_TAG, DATA_VAR_TAG, MATRIX_TAG, LIST_TAG, FUNC_BEGIN_TAG, PRGM_TAG, STR_DATA_TAG, EQUATION_TAG, GEN_DATA_TAG. Each value in the *Types* list will be presented to the user in a drop-down (unless there is only one value in the list then it will be a static field).

Current — A string pointer to a string that will be placed in the title of the dialog box (NULL if no title wanted).

subTypeList (optional) — For each GEN_DATA_TAG byte in the *Types* list, there must be a string pointer in the *subTypeList* that points to the text to display in the drop down. This parameter may be left off if there are no GEN_DATA_TAGs in the *Types* list.

Outputs: The HSYM of the newly created symbol or H_NULL if the user presses ESC or there is an error creating the symbol.

Assumptions: None

Side Effects: May cause heap compression.

The first WORD in the global array, **VarOptList**, is set to the index of the type selected when *Types* contains multiple values. It will be one if the first entry was selected, two for the second entry, and so on. See the example below. **VarOptList** is available on AMS 2.00 and higher.

Availability: All versions of the TI‑89 / TI‑92 Plus.

TI‑89 / TI‑92 Plus Differences: None

See Also: **VarNew, VarOpen**

Example: See **VarNew**.

Appendix A: System Routines — Direct Floating Point Operations

See Also:

acos

Declaration: double **acos** (double *x*)

Category(ies): Direct Floating Point Operations

Description: Computes the inverse cosine of *x*.

Inputs: *x* — A double floating-point value.

Outputs: For -1 <= *x* <= 1 returns inverse cosine of *x* in radian measure.

For *x* < -1 or *x* > 1 returns a floating-point NAN.

Assumptions: None

Side Effects: None

Availability: All versions of the TI-89 / TI-92 Plus.

TI-89 / TI-92 Plus Differences: None

See Also: **asin, atan, cos, sin, tan**

Example:

acosh

Declaration: double **acosh** (double *x*)

Category(ies): Direct Floating Point Operations

Description: Computes the inverse hyperbolic cosine of *x*.

Inputs: *x* — A double floating-point value.

Outputs: For *x* >= 1 returns inverse hyperbolic cosine of *x.*

For *x* < 1 returns a floating-point NAN.

Assumptions: None

Side Effects: None

Availability: All versions of the TI-89 / TI-92 Plus.

TI-89 / TI-92 Plus Differences: None

See Also: **asinh, atanh, cosh, sinh, tanh**

Example:

asin

Declaration: double **asin** (double *x*)

Category(ies): Direct Floating Point Operations

Description: Computes the inverse sine of *x*.

Inputs: *x* — A double floating-point value.

Outputs: For *x* >= 1 returns inverse sine of *x* in radian measure.

For *x* < 1 returns a floating-point NAN.

Assumptions: None

Side Effects: None

Availability: All versions of the TI-89 / TI-92 Plus.

TI-89 / TI-92 Plus Differences: None

See Also: **acos, atan, cos, sin, tan**

Example:

asinh

Declaration: double **asinh** (double *x*)

Category(ies): Direct Floating Point Operations

Description: Computes the inverse hyperbolic sine of *x*.

Inputs: *x* — A double floating-point value.

Outputs: Returns the inverse hyperbolic sine of *x*.

Assumptions: None

Side Effects: None

Availability: All versions of the TI-89 / TI-92 Plus.

TI-89 / TI-92 Plus Differences: None

See Also: **acosh, atanh, cosh, sinh, tanh**

Example:

atan

Declaration: double **atan** (double *x*)

Category(ies): Direct Floating Point Operations

Description: Computes the inverse tangent of *x*.

Inputs: *x* — A double floating-point value.

Outputs: Returns the inverse tangent of *x* in radian measure.

Assumptions: None

Side Effects: None

Availability: All versions of the TI-89 / TI-92 Plus.

TI-89 / TI-92 Plus Differences: None

See Also: **acos, asin, cos, sin, tan**

Example:

atan2

Declaration: double **atan2** (double *y*, double *x*)

Category(ies): Direct Floating Point Operations

Description: Computes the inverse tangent of *y* / *x*.

Inputs: *y, x* — Double floating-point values.

Outputs: Returns the inverse tangent of *y* / *x* in radian measure.

Assumptions: None

Side Effects: None

Availability: All versions of the TI-89 / TI-92 Plus.

TI-89 / TI-92 Plus Differences: None

See Also: **acos, asin, atan, cos, sin, tan**

Example:

atanh

Declaration: double **atanh** (double *x*)

Category(ies): Direct Floating Point Operations

Description: Computes the inverse hyperbolic tangent of *x*.

Inputs: *x* — A double floating-point value.

Outputs: If $-1 < x < 1$, then returns inverse hyperbolic tangent of *x*.
If $x = 1$, then returns floating-point positive infinity.
If $x = -1$, then returns floating-point negative infinity.
If $x < -1$ or $x > 1$, then returns a floating-point NAN.

Assumptions: None

Side Effects: None

Availability: All versions of the TI-89 / TI-92 Plus.

TI-89 / TI-92 Plus Differences: None

See Also: **acosh, asinh, cosh, sinh, tanh**

Example:

bcdadd

Declaration: BCD16 **bcdadd** (BCD16 *a*, BCD16 *b*)

Category(ies): Direct Floating Point Operations

Description: Add two TI binary-code decimal floating-point numbers. This routine performs the same function as the C "+" operator on BCD16 values. This routine is provided as a convenience for assembly language programs.

Inputs: *a*, *b* — Two BCD16 numbers to add together.

Outputs: Returns *a* + *b* as a BCD16 number.

Assumptions: None

Side Effects: None

Availability: All versions of the TI-89 / TI-92 Plus.

TI-89 / TI-92 Plus Differences: None

See Also: **bcdbcd, bcdcmp, bcddiv, bcdlong, bcdmul, bcdneg, bcdsub**

Example:

```
bcdresult   = -10
a           = -20
b           = -30
sum         = -40
            .
            .
            .
            move.l      b(a6),-(sp)           ;push b (a 10-byte BCD16 value)
            move.l      b+4(a6),-(sp)
            move.w      b+8(a6),-(sp)
            move.l      a(a6),-(sp)           ;push a
            move.l      a+4(a6),-(sp)
            move.w      a+8(a6),-(sp)
            move.l      bcdadd(a2),a0         ;assumes a2 -> jump table
            jsr         (a0)                  ;call bcdadd(a, b)
            lea         20(sp),sp             ;pop parameters
            ; BCD16 routine value is always at -10(a6)
            move.l      bcdresult(a6),sum(a6)
            move.l      bcdresult+4(a6),sum+4(a6)
            move.w      bcdresult+8(a6),sum+8(a6)
```

bcdbcd

Declaration: BCD16 **bcdbcd** (long *a*)

Category(ies): Direct Floating Point Operations

Description: Convert an integer to BCD16 floating point. This routine performs the same function as the C cast operator "(BCD16)" on a long integer value. This routine is provided as a convenience for assembly language programs.

Inputs: *a* — A long integer.

Outputs: Returns the BCD16 floating point equivalent of *a*.

Assumptions: None

Side Effects: None

Availability: All versions of the TI-89 / TI-92 Plus.

TI-89 / TI-92 Plus Differences: None

See Also: **bcdadd, bcdcmp, bcddiv, bcdlong, bcdmul, bcdneg, bcdsub**

Example:

```
bcdresult   = -10
bcda        = -20
a           = -24
            .
            .
            .
            move.l      a(a6),-(sp)           ;push a (a long integer)
            move.l      bcdbcd(a2),a0         ;assumes a2 -> jump table
            jsr         (a0)                  ;call bcdbcd(a)
            addq.l      #4,sp                 ;pop parameters
            ; BCD16 routine value is always at -10(a6)
            move.l      bcdresult(a6),bcda(a6)
            move.l      bcdresult+4(a6),bcda+4(a6)
            move.w      bcdresult+8(a6),bcda+8(a6)
```

bcdcmp

Declaration: long **bcdcmp** (BCD16 *a*, BCD16 *b*)

Category(ies): Direct Floating Point Operations

Description: Compare two TI binary-code decimal floating-point numbers. This routine is provided as a convenience for assembly language programs.

Inputs: *a*, *b* — Two BCD16 numbers to compare.

Outputs: Returns -1 if *a* < *b*,
0 if *a* == *b*
1 if *a* > *b*.

Assumptions: None

Side Effects: None

Availability: All versions of the TI-89 / TI-92 Plus.

TI-89 / TI-92 Plus Differences: None

See Also: **bcdadd, bcdbcd, bcddiv, bcdlong, bcdmul, bcdneg, bcdsub**

Example:

```
a           = -20
b           = -30
            .
            .
            .
            move.l      b(a6),-(sp)          ;push b (a 10-byte BCD16 value)
            move.l      b+4(a6),-(sp)
            move.w      b+8(a6),-(sp)
            move.l      a(a6),-(sp)          ;push a
            move.l      a+4(a6),-(sp)
            move.w      a+8(a6),-(sp)
            move.l      bcdcmp(a2),a0        ;assumes a2 -> jump table
            jsr         (a0)                 ;call bcdcmp(a, b)
            lea         20(sp),sp            ;pop parameters
            tst.l       d0                   ;result returned in d0
            blt         less                 ;a < b --->
            bgt         greater              ;a > b --->
            .                                ;a == b
            .
            .
```

bcddiv

Declaration: BCD16 **bcddiv** (BCD16 *a*, BCD16 *b*)

Category(ies): Direct Floating Point Operations

Description: Calculate the quotient of two TI binary-code decimal floating-point numbers. This routine performs the same function as the C "/" operator on BCD16 values. This routine is provided as a convenience for assembly language programs.

Inputs: *a*, *b* — Two BCD16 numbers to divide.

Outputs: Returns *a* / *b* as a BCD16 number.

Assumptions: None

Side Effects: None

Availability: All versions of the TI-89 / TI-92 Plus.

TI-89 / TI-92 Plus Differences: None

See Also: **bcdadd, bcdbcd, bcdcmp, bcdlong, bcdmul, bcdneg, bcdsub**

Example:

```
bcdresult   = -10
a           = -20
b           = -30
quot        = -40
            .
            .
            .
            move.l      b(a6),-(sp)          ;push b (a 10-byte BCD16 value)
            move.l      b+4(a6),-(sp)
            move.w      b+8(a6),-(sp)
            move.l      a(a6),-(sp)          ;push a
            move.l      a+4(a6),-(sp)
            move.w      a+8(a6),-(sp)
            move.l      bcddiv(a2),a0        ;assumes a2 -> jump table
            jsr         (a0)                 ;call bcddiv(a, b)
            lea         20(sp),sp            ;pop parameters
            ; BCD16 routine value is always at –10(a6)
            move.l      bcdresult(a6),quot(a6)
            move.l      bcdresult+4(a6),quot+4(a6)
            move.w      bcdresult+8(a6),quot+8(a6)
```

bcdlong

Declaration: BCD16 **bcdlong** (BCD16 *a*)

Category(ies): Direct Floating Point Operations

Description: Convert a BCD16 floating-point number to an integer. This routine performs the same function as the C cast operator “(long)” on a BCD16 value. This routine is provided as a convenience for assembly language programs.

Inputs: *a* — A BCD16 floating pointer number.

Outputs: Returns the long integer equivalent of *a*.

Assumptions: None

Side Effects: None

Availability: All versions of the TI-89 / TI-92 Plus.

TI-89 / TI-92 Plus Differences: None

See Also: **bcdadd, bcdbcd, bcdcmp, bcddiv, bcdmul, bcdneg, bcdsub**

Example:

```
a           = -20
longa       = -24
            .
            .
            .
            move.l      a(a6),-(sp)          ;push a (a BCD16 floating-point number)
            move.l      a+4(a6),-(sp)
            move.w      a+8(a6),-(sp)
            move.l      bcdlong(a2),a0       ;assumes a2 -> jump table
            jsr         (a0)                 ;call bcdlong(a)
            lea         10(sp),sp            ;pop parameters
            move.l      d0,longa(sp)         ;returned long value is in D0
```

bcdmul

Declaration: BCD16 **bcdmul** (BCD16 *a*, BCD16 *b*)

Category(ies): Direct Floating Point Operations

Description: Calculate the product of two TI binary-code decimal floating-point numbers. This routine performs the same function as the C "∗" operator on BCD16 values. This routine is provided as a convenience for assembly language programs.

Inputs: *a*, *b* — Two BCD16 numbers to multiply.

Outputs: Returns *a* ∗ *b* as a BCD16 number.

Assumptions: None

Side Effects: None

Availability: All versions of the TI-89 / TI-92 Plus.

TI-89 / TI-92 Plus Differences: None

See Also: **bcdadd, bcdbcd, bcdcmp, bcddiv, bcdlong, bcdneg, bcdsub**

Example:

```
bcdresult   = -10
a           = -20
b           = -30
prod        = -40
            .
            .
            .
            move.l      b(a6),-(sp)          ;push b (a 10-byte BCD16 value)
            move.l      b+4(a6),-(sp)
            move.w      b+8(a6),-(sp)
            move.l      a(a6),-(sp)          ;push a
            move.l      a+4(a6),-(sp)
            move.w      a+8(a6),-(sp)
            move.l      bcdmul(a2),a0        ;assumes a2 -> jump table
            jsr         (a0)                 ;call bcdmul(a, b)
            lea         20(sp),sp            ;pop parameters
            ; BCD16 routine value is always at –10(a6)
            move.l      bcdresult(a6),prod(a6)
            move.l      bcdresult+4(a6),prod+4(a6)
            move.w      bcdresult+8(a6),prod+8(a6)
```

bcdneg

Declaration: BCD16 **bcdlong** (BCD16 *a*)

Category(ies): Direct Floating Point Operations

Description: Negate a BCD16 floating-point number. This routine performs the same function as the C negative operator "-" on a BCD16 value. This routine is provided as a convenience for assembly language programs.

Inputs: *a* — A BCD16 floating pointer number.

Outputs: Returns the negative of *a*.

Assumptions: None

Side Effects: None

Availability: All versions of the TI-89 / TI-92 Plus.

TI-89 / TI-92 Plus Differences: None

See Also: **bcdadd, bcdbcd, bcdcmp, bcddiv, bcdlong, bcdmul, bcdsub**

Example:

```
bcdresult   = -10
a           = -20
nega        = -30
            .
            .
            .
            move.l      a(a6),-(sp)           ;push a (a BCD16 floating-point number)
            move.l      a+4(a6),-(sp)
            move.w      a+8(a6),-(sp)
            move.l      bcdneg(a2),a0         ;assumes a2 -> jump table
            jsr         (a0)                  ;call bcdneg(a)
            lea         10(sp),sp             ;pop parameters
            ; BCD16 routine value is always at -10(a6)
            move.l      bcdresult(a6),nega(a6)
            move.l      bcdresult+4(a6),nega+4(a6)
            move.w      bcdresult+8(a6),nega+8(a6)
```

bcdsub

Declaration: BCD16 **bcdsub** (BCD16 *a*, BCD16 *b*)

Category(ies): Direct Floating Point Operations

Description: Calculate the difference between two TI binary-code decimal floating-point numbers. This routine performs the same function as the C "-" operator on BCD16 values. This routine is provided as a convenience for assembly language programs.

Inputs: *a*, *b* — Two BCD16 numbers to subtract.

Outputs: Returns *a* - *b* as a BCD16 number.

Assumptions: None

Side Effects: None

Availability: All versions of the TI-89 / TI-92 Plus.

TI-89 / TI-92 Plus Differences: None

See Also: **bcdadd, bcdbcd, bcdcmp, bcddiv, bcdlong, bcdmul, bcdneg**

Example:

```
bcdresult   = -10
a           = -20
b           = -30
diff        = -40
            .
            .
            .
            move.l      b(a6),-(sp)          ;push b (a 10-byte BCD16 value)
            move.l      b+4(a6),-(sp)
            move.w      b+8(a6),-(sp)
            move.l      a(a6),-(sp)          ;push a
            move.l      a+4(a6),-(sp)
            move.w      a+8(a6),-(sp)
            move.l      bcdsub(a2),a0        ;assumes a2 -> jump table
            jsr         (a0)                 ;call bcdsub(a, b)
            lea         20(sp),sp            ;pop parameters
            ; BCD16 routine value is always at -10(a6)
            move.l      bcdresult(a6),diff(a6)
            move.l      bcdresult+4(a6),diff+4(a6)
            move.w      bcdresult+8(a6),diff+8(a6)
```

cacos

Declaration: void **cacos** (double *x*, double *y*, double * *u*, double * *v*)

Category(ies): Direct Floating Point Operations

Description: Computes the inverse cosine of the complex number $x + y * i$.

Inputs:

x — A double floating-point value representing the real part of a complex number.

y — A double floating-point value representing the imaginary part of a complex number.

u, v — Pointers to double variables.

Outputs:

* *u* — The real part of the complex result.

* *v* — The imaginary part of the complex result.

Assumptions: None

Side Effects: None

Availability: All versions of the TI-89 / TI-92 Plus.

TI-89 / TI-92 Plus Differences: None

See Also: **casin, catan, ccos, csin, ctan**

Example:

cacosh

Declaration: void **cacosh** (double *x*, double *y*, double * *u*, double * *v*)

Category(ies): Direct Floating Point Operations

Description: Computes the inverse hyperbolic cosine of the complex number $x + y * i$.

Inputs:

x — A double floating-point value representing the real part of a complex number.

y — A double floating-point value representing the imaginary part of a complex number.

u, v — Pointers to double variables.

Outputs:

* *u* — The real part of the complex result.

* *v* — The imaginary part of the complex result.

Assumptions: None

Side Effects: None

Availability: All versions of the TI-89 / TI-92 Plus.

TI-89 / TI-92 Plus Differences: None

See Also: **casinh, catanh, ccosh, csinh, ctanh**

Example:

casin

Declaration: void **casin** (double *x*, double *y*, double * *u*, double * *v*)

Category(ies): Direct Floating Point Operations

Description: Computes the inverse sine of the complex number $x + y * i$.

Inputs:

x — A double floating-point value representing the real part of a complex number.

y — A double floating-point value representing the imaginary part of a complex number.

u, v — Pointers to double variables.

Outputs:

* *u* — The real part of the complex result.

* *v* — The imaginary part of the complex result.

Assumptions: None

Side Effects: None

Availability: All versions of the TI‑89 / TI‑92 Plus.

TI‑89 / TI‑92 Plus Differences: None

See Also: **cacos, catan, ccos, csin, ctan**

Example:

casinh

Declaration: void **casinh** (double *x*, double *y*, double * *u*, double * *v*)

Category(ies): Direct Floating Point Operations

Description: Computes the inverse hyperbolic sine of the complex number $x + y * i$.

Inputs:

x — A double floating-point value representing the real part of a complex number.

y — A double floating-point value representing the imaginary part of a complex number.

u, v — Pointers to double variables.

Outputs:

* *u* — The real part of the complex result.

* *v* — The imaginary part of the complex result.

Assumptions: None

Side Effects: None

Availability: All versions of the TI-89 / TI-92 Plus.

TI-89 / TI-92 Plus Differences: None

See Also: **cacosh, catanh, ccosh, csinh, ctanh**

Example:

catan

Declaration: void **catan** (double *x*, double *y*, double * *u*, double * *v*)

Category(ies): Direct Floating Point Operations

Description: Computes the inverse tangent of the complex number $x + y * i$.

Inputs:

x — A double floating-point value representing the real part of a complex number.

y — A double floating-point value representing the imaginary part of a complex number.

u, v — Pointers to double variables.

Outputs:

* *u* — The real part of the complex result.

* *v* — The imaginary part of the complex result.

Assumptions: None

Side Effects: None

Availability: All versions of the TI-89 / TI-92 Plus.

TI-89 / TI-92 Plus Differences: None

See Also: **cacos, casin, ccos, csin, ctan**

Example:

catanh

Declaration: void **catanh** (double *x*, double *y*, double * *u*, double * *v*)

Category(ies): Direct Floating Point Operations

Description: Computes the inverse hyperbolic tangent of the complex number $x + y * i$.

Inputs:

x — A double floating-point value representing the real part of a complex number.

y — A double floating-point value representing the imaginary part of a complex number.

u, v — Pointers to double variables.

Outputs:

* *u* — The real part of the complex result.

* *v* — The imaginary part of the complex result.

Assumptions: None

Side Effects: None

Availability: All versions of the TI-89 / TI-92 Plus.

TI-89 / TI-92 Plus Differences: None

See Also: **cacosh, casinh, ccosh, csinh, ctanh**

Example:

ccos

Declaration: void **ccos** (double *x*, double *y*, double * *u*, double * *v*)

Category(ies): Direct Floating Point Operations

Description: Computes the cosine of the complex number $x + y * i$.

Inputs:

x — A double floating-point value representing the real part of a complex number.

y — A double floating-point value representing the imaginary part of a complex number.

u, v — Pointers to double variables.

Outputs:

* *u* — The real part of the complex result.

* *v* — The imaginary part of the complex result.

Assumptions: None

Side Effects: None

Availability: All versions of the TI-89 / TI-92 Plus.

TI-89 / TI-92 Plus Differences: None

See Also: **cacos, casin, catan, csin, ctan**

Example:

ccosh

Declaration: void **ccosh** (double *x*, double *y*, double * *u*, double * *v*)

Category(ies): Direct Floating Point Operations

Description: Computes the hyperbolic cosine of the complex number $x + y * i$.

Inputs:

x — A double floating-point value representing the real part of a complex number.

y — A double floating-point value representing the imaginary part of a complex number.

u, v — Pointers to double variables.

Outputs:

* *u* — The real part of the complex result.

* *v* — The imaginary part of the complex result.

Assumptions: None

Side Effects: None

Availability: All versions of the TI-89 / TI-92 Plus.

TI-89 / TI-92 Plus Differences: None

See Also: **cacosh, casinh, catanh, csinh, ctanh**

Example:

ceil

Declaration: double **ceil** (double *x*)

Category(ies): Direct Floating Point Operations

Description: Computes the ceiling of *x*.

Inputs: *x* — A double floating-point value.

Outputs: The smallest integer >= *x*.

Assumptions: None

Side Effects: None

Availability: All versions of the TI-89 / TI-92 Plus.

TI-89 / TI-92 Plus Differences: None

See Also: **floor**

Example:

cexp

Declaration: void **cexp** (double *x*, double *y*, double * *u*, double * *v*)

Category(ies): Direct Floating Point Operations

Description: Computes the exponential function of the complex number $x + y * i$.

Inputs:
- *x* — A double floating-point value representing the real part of a complex number.
- *y* — A double floating-point value representing the imaginary part of a complex number.
- *u, v* — Pointers to double variables.

Outputs:
- * *u* — The real part of the complex result.
- * *v* — The imaginary part of the complex result.

Assumptions: None

Side Effects: None

Availability: All versions of the TI-89 / TI-92 Plus.

TI-89 / TI-92 Plus Differences: None

See Also: **cln, clog10, csqrt**

Example:

ck_valid_float

Declaration: BOOL **ck_valid_float** (BCD16 * *num*)

Category: Direct Floating Point Operations

Description: Verifies that a floating-point value is within the valid user range. The internal TI BCD floating-point format has 16 mantissa digits with an exponent range of -16384 to 16383 while a valid user float is 14 mantissa digits with an exponent range of -999 to 999. **ck_valid_float** rounds the input BCD16 value *num* to 14 digits and underflows the value to 0 if the exponent is less than -999. If *num* is NAN or the exponent is greater than 999, FALSE is returned and *num* will contain a NAN. (See section **2.9.4. Floating-Point Representations** in the TI-89 / TI-92 Plus Sierra C Assembler Reference Manual.)

Inputs: *num* — Pointer to a TI BCD floating-point value (BCD16).

Outputs: *num* is rounded to 14 mantissa digits or underflowed to 0 if necessary to insure that its contents are within the range of a user floating-point value. In this case, TRUE will be returned. If the input *num* is NAN or the exponent is greater than 999, the output value of *num* will be NAN and FALSE will be returned.

Assumptions: None

Side Effects: None

Availability: All versions of the TI-89 / TI-92 Plus.

TI-89 / TI-92 Plus Differences: None

See Also: **is_float_transfinite, push_Float**

Example:

```
/* convert an internal floating point value to the range of a user float */
if( !ck_valid_float( &temp ))   /* does round14 and underflows if necessary */
   ER_throw( ER_OVERFLOW );   /* could not convert to valid user float range */
```

cln

Declaration: void **cln** (double *x*, double *y*, double * *u*, double * *v*)

Category(ies): Direct Floating Point Operations

Description: Computes the natural logarithm of the complex number $x + y * i$.

Inputs:

x — A double floating-point value representing the real part of a complex number.

y — A double floating-point value representing the imaginary part of a complex number.

u, v — Pointers to double variables.

Outputs:

* *u* — The real part of the complex result.

* *v* — The imaginary part of the complex result.

Assumptions: None

Side Effects: None

Availability: All versions of the TI-89 / TI-92 Plus.

TI-89 / TI-92 Plus Differences: None

See Also: **cexp, clog10, csqrt**

Example:

clog10

Declaration: void **clog10** (double *x*, double *y*, double * *u*, double * *v*)

Category(ies): Direct Floating Point Operations

Description: Computes the base 10 logarithm of the complex number $x + y * i$.

Inputs:
- *x* — A double floating-point value representing the real part of a complex number.
- *y* — A double floating-point value representing the imaginary part of a complex number.
- *u, v* — Pointers to double variables.

Outputs:
- * *u* — The real part of the complex result.
- * *v* — The imaginary part of the complex result.

Assumptions: None

Side Effects: None

Availability: All versions of the TI-89 / TI-92 Plus.

TI-89 / TI-92 Plus Differences: None

See Also: **cexp, cln, csqrt**

Example:

cos

Declaration: double **cos** (double *x*)

Category(ies): Direct Floating Point Operations

Description: Computes the cosine of *x* radians.

Inputs: *x* — A double floating-point value.

Outputs: For -1.e13 < *x* < 1.e13 returns the cosine of *x* radians.
For *x* <= -1.e13 or *x* >= 1.e13 returns a floating-point NAN.

Assumptions: None

Side Effects: None

Availability: All versions of the TI-89 / TI-92 Plus.

TI-89 / TI-92 Plus Differences: None

See Also: **acos, asin, atan, sin, tan**

Example:

cosh

Declaration: double **cosh** (double *x*)

Category(ies): Direct Floating Point Operations

Description: Computes the hyperbolic cosine of *x*.

Inputs: *x* — A double floating-point value.

Outputs: The hyperbolic cosine of *x*.

Assumptions: None

Side Effects: None

Availability: All versions of the TI-89 / TI-92 Plus.

TI-89 / TI-92 Plus Differences: None

See Also: **acosh, asinh, atanh, sinh, tanh**

Example:

csin

Declaration: void **csin** (double *x*, double *y*, double * *u*, double * *v*)

Category(ies): Direct Floating Point Operations

Description: Computes the sine of the complex number $x + y * i$.

Inputs:

x — A double floating-point value representing the real part of a complex number.

y — A double floating-point value representing the imaginary part of a complex number.

u, v — Pointers to double variables.

Outputs:

* *u* — The real part of the complex result.

* *v* — The imaginary part of the complex result.

Assumptions: None

Side Effects: None

Availability: All versions of the TI-89 / TI-92 Plus.

TI-89 / TI-92 Plus Differences: None

See Also: **cacos, casin, catan, ccos, ctan**

Example:

csinh

Declaration: void **csinh** (double *x*, double *y*, double * *u*, double * *v*)

Category(ies): Direct Floating Point Operations

Description: Computes the hyperbolic sine of the complex number $x + y * i$.

Inputs:

x — A double floating-point value representing the real part of a complex number.

y — A double floating-point value representing the imaginary part of a complex number.

u, v — Pointers to double variables.

Outputs:

* *u* — The real part of the complex result.

* *v* — The imaginary part of the complex result.

Assumptions: None

Side Effects: None

Availability: All versions of the TI-89 / TI-92 Plus.

TI-89 / TI-92 Plus Differences: None

See Also: **cacosh, casinh, catanh, ccosh, ctanh**

Example:

csqrt

Declaration: void **csqrt** (double *x*, double *y*, double * *u*, double * *v*)

Category(ies): Direct Floating Point Operations

Description: Computes the square root of the complex number $x + y * i$.

Inputs:

x — A double floating-point value representing the real part of a complex number.

y — A double floating-point value representing the imaginary part of a complex number.

u, v — Pointers to double variables.

Outputs:

* *u* — The real part of the complex result.

* *v* — The imaginary part of the complex result.

Assumptions: None

Side Effects: None

Availability: All versions of the TI-89 / TI-92 Plus.

TI-89 / TI-92 Plus Differences: None

See Also: **cexp, cln, clog10**

Example:

ctan

Declaration: void **ctan** (double *x*, double *y*, double * *u*, double * *v*)

Category(ies): Direct Floating Point Operations

Description: Computes the tangent of the complex number $x + y * i$.

Inputs:

x — A double floating-point value representing the real part of a complex number.

y — A double floating-point value representing the imaginary part of a complex number.

u, v — Pointers to double variables.

Outputs:

* *u* — The real part of the complex result.

* *v* — The imaginary part of the complex result.

Assumptions: None

Side Effects: None

Availability: All versions of the TI‑89 / TI‑92 Plus.

TI‑89 / TI‑92 Plus Differences: None

See Also: **cacos, casin, catan, ccos, csin**

Example:

ctanh

Declaration: void **ctanh** (double *x*, double *y*, double * *u*, double * *v*)

Category(ies): Direct Floating Point Operations

Description: Computes the hyperbolic tangent of the complex number $x + y * i$.

Inputs:

x — A double floating-point value representing the real part of a complex number.

y — A double floating-point value representing the imaginary part of a complex number.

u, v — Pointers to double variables.

Outputs:

* *u* — The real part of the complex result.

* *v* — The imaginary part of the complex result.

Assumptions: None

Side Effects: None

Availability: All versions of the TI-89 / TI-92 Plus.

TI-89 / TI-92 Plus Differences: None

See Also: **cacosh, casinh, catanh, ccosh, csinh**

Example:

estack_number_to_Float

Declaration: Float **estack_number_to_Float** (IndexConstQuantum *i*)

Category(ies): Direct Floating Point Operations, EStack Utilities

Description: Returns the floating-point representation of *i*.

Inputs: *i* — Index of a tagged number.

Outputs: Floating-point representation of *i*.

Assumptions: *i* points to the tag of a tagged number.

Side Effects: None

Availability: On AMS 1.05 and higher.

TI-89 / TI-92 Plus Differences: None

See Also: **estack_to_float**

Example:

```
Boolean sum_has_small_numeric_term (EStackIndex i, Float tolerance)
/* i indexes a sum.
   Returns TRUE if it does not have a top-level numeric term or if its
   magnitude is < tolerance.
*/
{  i = index_numeric_term (i);
   return NULL_INDEX == i ||
          fabs (estack_number_to_Float (i)) <= tolerance;
}
```

estack_to_float

Declaration: Float **estack_to_float** (IndexConstQuantum *i*)

Category(ies): Direct Floating Point Operations, EStack Utilities

Description: Returns the Float value indexed by *i*.

Inputs: *i* — Index to a tagged float.

Outputs: Returns the Float value.

Assumptions: *i* points to the tag of a tagged float.

Side Effects: None

Availability: On AMS 2.00 and higher.

TI-89 / TI-92 Plus Differences: None

See Also: **estack_number_to_Float, estack_to_short, estack_to_ushort**

Example:

```
Boolean is_Float_exact_whole_number (IndexConstQuantum i)
/* i indexes a Float.
   Returns TRUE if it is a whole number whose magnitude is less than
   the smallest whole number that is not represented exactly.
*/
{  Float f;
   f = estack_to_float (i);
   return fabs (f) <= MAX_EXACT_FLOAT_WHOLE_NUMBER &&
          FLOAT0 == FMOD (f, FLOAT1);
}
```

exp

Declaration: double **exp** (double *x*)

Category(ies): Direct Floating Point Operations

Description: Computes the exponential function of *x*.

Inputs: *x* — A double floating-point value.

Outputs: The exponential function of *x*.

Assumptions: None

Side Effects: None

Availability: All versions of the TI-89 / TI-92 Plus.

TI-89 / TI-92 Plus Differences: None

See Also: **log, log10, pow, sqrt**

Example:

fabs

Declaration: double **fabs** (double *x*)

Category(ies): Direct Floating Point Operations

Description: Computes the absolute value of *x*.

Inputs: *x* — A double floating-point value.

Outputs: For *x* >= 0.0 returns *x*.
For *x* < 0.0 returns -*x*.

Assumptions: None

Side Effects: None

Availability: All versions of the TI-89 / TI-92 Plus.

TI-89 / TI-92 Plus Differences: None

See Also: None

Example:

floor

Declaration: double **floor** (double *x*)

Category(ies): Direct Floating Point Operations

Description: Computes the floor of *x*.

Inputs: *x* — A double floating-point value.

Outputs: The largest integer <= *x*.

Assumptions: None

Side Effects: None

Availability: All versions of the TI-89 / TI-92 Plus.

TI-89 / TI-92 Plus Differences: None

See Also: **ceil**

Example:

fmod

Declaration: double **fmod** (double *x*, double *y*)

Category(ies): Direct Floating Point Operations

Description: Computes the floating-point remainder of *x* / *y*.

Inputs: *x, y* — Double floating-point values.

Outputs: The floating-point remainder of *x* / *y*.

Assumptions: None

Side Effects: None

Availability: All versions of the TI‑89 / TI‑92 Plus.

TI‑89 / TI‑92 Plus Differences: None

See Also: **modf**

Example:

frexp10

Declaration: double **frexp10** (double *x*, int * *exp_ptr*)

Category(ies): Direct Floating Point Operations

Description: For *x* = f ∗ 10^n with 0.1 <= |f| < 1.0, returns f after storing n into * *exp_ptr*. Otherwise if *x* is any kind of floating-point zero, returns *x* and stores 0 into * *exp_ptr*.

Inputs:
x — Double.
exp_ptr — Pointer to an integer.

Outputs: None

Assumptions: None

Side Effects: None

Availability: On AMS 1.05 and higher.

TI-89 / TI-92 Plus Differences: None

See Also: Analogous to frexp (double *x*, int * *exp_ptr*) in the ANSI C math library, except using base ten rather than base two.

Example:

```
double quick_log_10_abs (double x)
/* Returns a quickly-computed rough approximation to the base-ten
   logarithm of abs(x) for nonzero x.
*/
{  int exponent;
   x = frexp10 (fabs(x), &exponent);
   return (exponent – 1) + x;
}
```

is_float_infinity

Declaration: Boolean **is_float_infinity** (Float *f*)

Category(ies): Direct Floating Point Operations

Description: Determines whether *f* represents $-\infty$, $+\infty$ or $\pm$ (unsigned) ∞.

Inputs: *f* — Float.

Outputs: Returns TRUE if *f* represents $-\infty$, $+\infty$ or $\pm$ (unsigned) ∞. Otherwise returns FALSE.

Assumptions: None

Side Effects: None

Availability: On AMS 1.05 and higher.

TI-89 / TI-92 Plus Differences: None

See Also: **is_float_signed_infinity, is_nan, is_float_unsigned_inf_or_nan, is_float_transfinite**

Example:

```
is_float_infinity (1.0/0.0);  /* Returns TRUE */
```

is_float_negative_zero

Declaration: Boolean **is_float_negative_zero** (Float *f*)

Category(ies): Direct Floating Point Operations

Description: Determines whether *f* represents 0^-.

Inputs: *f* — Float.

Outputs: Returns TRUE if *f* represents 0^-. Otherwise returns FALSE.

Assumptions: None

Side Effects: None

Availability: On AMS 1.05 and higher.

TI-89 / TI-92 Plus Differences: None

See Also: **is_float_positive_zero, is_float_unsigned_zero, is0, is_complex0**

Example:

```
is_float_negative_zero (-fabs (0.0));   /* Returns TRUE */
```

is_float_positive_zero

Declaration: Boolean **is_float_positive_zero** (Float *f*)

Category(ies): Direct Floating Point Operations

Description: Determines whether *f* represents 0+.

Inputs: *f* — Float.

Outputs: Returns TRUE if *f* represents 0+. Otherwise returns FALSE.

Assumptions: None

Side Effects: None

Availability: On AMS 1.05 and higher.

TI-89 / TI-92 Plus Differences: None

See Also: **is_float_negative_zero, is_float_unsigned_zero, is0, is_complex0**

Example:

```
is_float_positive_zero(fabs (0.0));   /* Returns TRUE */
```

is_float_signed_infinity

Declaration: Boolean **is_float_signed_infinity** (Float *f*)

Category(ies): Direct Floating Point Operations

Description: Determines whether *f* represents -∞ or +∞.

Inputs: *f* — Float.

Outputs: Returns TRUE if *f* represents -∞ or +∞. Otherwise returns FALSE.

Assumptions: None

Side Effects: None

Availability: On AMS 1.05 and higher.

TI-89 / TI-92 Plus Differences: None

See Also: **is_float_infinity, is_nan, is_float_unsigned_inf_or_nan, is_float_transfinite**

Example:

```
is_float_signed_infinity (-fabs(1.0/0.0));  /* Returns TRUE */
is_float_signed_infinity (1.0/0.0);         /* Returns FALSE */
```

is_float_transfinite

Declaration: Boolean **is_float_transfinite** (Float *f*)

Category(ies): Direct Floating Point Operations

Description: Determines whether *f* represents -∞, +∞, unsigned ∞, or NAN.

Inputs: *f* — Float.

Outputs: Returns TRUE if *f* represents -∞, +∞, unsigned ∞, or NAN. Otherwise returns FALSE.

Assumptions: None

Side Effects: None

Availability: All versions of the TI-89 / TI-92 Plus.

TI-89 / TI-92 Plus Differences: None

See Also: **is_float_signed_infinity, is_nan, is_float_unsigned_inf_or_nan, is_float_infinity**

Example:

```
is_Float_transfinite (0.0/0.0);   /* Returns TRUE) */
```

is_float_unsigned_inf_or_nan

Declaration: Boolean **is_float_unsigned_inf_or_nan** (Float *f*)

Category(ies): Direct Floating Point Operations

Description: Determines whether *f* represents unsigned ∞ or NAN.

Inputs: *f* — Float.

Outputs: Returns TRUE if *f* represents unsigned ∞ or NAN. Otherwise returns FALSE.

Assumptions: None

Side Effects: None

Availability: All versions of the TI-89 / TI-92 Plus.

TI-89 / TI-92 Plus Differences: None

See Also: **is_float_signed_infinity, is_nan, is_float_transfinite, is_float_infinity**

Example:

```
is_float_unsigned_inf_or_nan (1.0/0.0);  /* Returns TRUE */
```

is_float_unsigned_zero

Declaration: Boolean **is_float_unsigned_zero** (Float *f*)

Category(ies): Direct Floating Point Operations

Description: Determines whether *f* represents +-0.

Inputs: *f* — Float.

Outputs: Returns TRUE if *f* represents +-0. Otherwise returns FALSE. This is not the same as `is_float_negative_zero(f) || is_float_positive_zero(f)`.

Assumptions: None

Side Effects: None

Availability: All versions of the TI-89 / TI-92 Plus.

TI-89 / TI-92 Plus Differences: None

See Also: **is_float_positive_zero, is_float_negative_zero, is0, is_complex0**

Example:

```
is_float_unsigned_zero (0.0);  /* Returns TRUE */
```

is_nan

Declaration: Boolean **is_nan** (Float *f*)

Category(ies): Direct Floating Point Operations

Description: Determines whether *f* represents a NAN (this is an undefined float, such as 0./0.).

Inputs: *f* — Float.

Outputs: Returns TRUE if *f* represents a NAN. Otherwise returns FALSE.

Assumptions: None

Side Effects: None

Availability: On AMS 1.05 and higher.

TI-89 / TI-92 Plus Differences: None

See Also: **is_float_signed_infinity, is_float_infinity, is_float_unsigned_inf_or_nan, is_float_transfinite**

Examples:

```
is_nan (0.0/0.0);  /* Returns TRUE */
```

log

Declaration: double **log** (double *x*)

Category(ies): Direct Floating Point Operations

Description: Computes the natural logarithm of *x*.

Inputs: *x* — A double floating-point value.

Outputs: The natural logarithm of *x*.

Assumptions: None

Side Effects: None

Availability: All versions of the TI-89 / TI-92 Plus.

TI-89 / TI-92 Plus Differences: None

See Also: **exp, log10, pow, sqrt**

Example:

log10

Declaration: double **log10** (double *x*)

Category(ies): Direct Floating Point Operations

Description: Computes the base 10 logarithm of *x*.

Inputs: *x* — A double floating-point value.

Outputs: The base 10 logarithm of *x*.

Assumptions: None

Side Effects: None

Availability: All versions of the TI-89 / TI-92 Plus.

TI-89 / TI-92 Plus Differences: None

See Also: **exp, log, pow, sqrt**

Example:

modf

Declaration: double **modf** (double *x*, double * *int_ptr*)

Category(ies): Direct Floating Point Operations

Description: Breaks down a floating-point number into fractional and integer parts.

Inputs: *x* — A double floating-point value.

Outputs: Returns the fractional portion of *x*.
Stores the integer portion of *x* as a floating-point value at * *int_ptr*.

Assumptions: None

Side Effects: None

Availability: All versions of the TI‑89 / TI‑92 Plus.

TI‑89 / TI‑92 Plus Differences: None

See Also: **modf**

Example:

pow

Declaration: double **pow** (double *x*, double *y*)

Category(ies): Direct Floating Point Operations

Description: Computes the base *x* raised to the *y* power.

Inputs: *x, y* — Double floating-point values.

Outputs: The base *x* raised to the *y* power.

Assumptions: None

Side Effects: None

Availability: All versions of the TI-89 / TI-92 Plus.

TI-89 / TI-92 Plus Differences: None

See Also: **exp, log, log10, sqrt**

Example:

push_Float

Declaration: void **push_Float** (Float *d*)

Category(ies): Direct Floating Point Operations

Description: Pushes a tagged float onto the estack if *d* is representable as a finite Float.

If *d* is not representable as a finite Float, the most appropriate transfinite tag is pushed.

Inputs: *d* — Untagged C Float value.

Outputs: None

Assumptions: None

Side Effects: None

Availability: On AMS 1.05 and higher.

TI-89 / TI-92 Plus Differences: None

See Also: **push_Float_to_nonneg_int**

Examples:

```
void push_half (void)
/* Pushes a tagged float 0.5 if global computation mode = APPROXIMATE.
   Otherwise pushes a tagged fraction 1/2. */
{  if (IS_ARITH_APPROX)
      push_Float (FLOAT_HALF);
   else
      push_reciprocal_of_quantum (2u);
}
```

push_Float_to_nonneg_int

Declaration: void **push_Float_to_nonneg_int** (Float *f*)

Category(ies): Direct Floating Point Operations

Description: If the truncated integer part of *f* is representable as a big integer, that tagged integer value is pushed onto the estack.

Inputs: *f* — Non-negative C floating-point number.

Outputs: None

Assumptions: None

Side Effects: Throws RATIONAL_NUMERIC_OVERFLOW_ERROR if the truncated integer part of *f* is not representable as a big integer.

Availability: On AMS 1.05 and higher.

TI-89 / TI-92 Plus Differences: None

See Also: **push_Float**

Examples:

```
Boolean did_push_cnvrt_Float_to_integer (EStackIndex i)
/* i indexes a tagged float.
   If it is a whole number that is representable as an integer, pushes the
      integer equivalent then returns TRUE.
   Otherwise returns FALSE. */
{  Float f;
   f = ESTACK_TO_FLOAT (i);
   if (FLOAT0 == FMOD (f, FLOAT1))
   {  Float abs_f = fabs (f);
      if (FLOAT_TO_NON_OVERFLOW_THRESHOLD > abs_f)
      {  push_Float_to_nonneg_int (abs_f);
         if (f < FLOAT0)
            negate_top()
         return TRUE;
      }
   }
   return FALSE;
}
```

round12

Declaration: double **round12** (double *x*)

Category(ies): Direct Floating Point Operations

Description: Rounds *x* to 12 digits of precision.

Inputs: *x* — A double floating-point value.

Outputs: The value of *x* rounded to 12 digits of precision.

Assumptions: None

Side Effects: None

Availability: All versions of the TI-89 / TI-92 Plus.

TI-89 / TI-92 Plus Differences: None

See Also: **round14**

Example:

round12_err

Declaration: BCD16 **round12_err** (BCD16 *fNum*, short *errCode*)

Category(ies): Direct Floating Point Operations

Description: Verify that a BCD16 value, *fNum*, is in the valid user range for floating point values. This routine will round to 12 digits and underflow to zero if the exponent is < -999. If the number is NAN or the exponent is > 999, then it will throw the error code passed in *errCode*.

Inputs:

fNum — Floating point number to check and round.

errCode — Error to throw if number passed is not a number or its exponent is greater than 999.

Outputs: The value of *fNum* rounded to 12 digits of precision.

Assumptions: None

Side Effects: May throw the error code passed in *errCode*.

Availability: All versions of the TI-89 / TI-92 Plus.

TI-89 / TI-92 Plus Differences: None

See Also: **round12, round14, ER_throw**

(continued)

round12_err *(continued)*

Example: The zoom trig on the grapher sets up the range variables depending on the degrees/radians setting and the number of pixels in the X direction of the graph as shown in this example. The xscl factor is rounded to 14 digits using **round14** but the xmin and xmax variables are rounded to 12 digits using **round12_err**.

```
void SysVarStore( short sysNum, BCD16 value )
{  Access_AMS_Global_Variables;
   EStackIndex saveTop = top_estack;
   BYTE tag[2];

   tag[1] = SYSVAR_TAG;
   tag[0] = sysNum;
   push_Float( value );
   VarStore( tag+1, STOF_ESI, 0, top_estack );
   top_estack = saveTop;
}

void GZ_Trig( void )
{  Access_AMS_Global_Variables;
   short xp;
   BCD16 *fr, TempF;

   fr = gr_active->rngp;
   xp = gr_active->xmaxpix;
   if (IS_DEGREES) {
      SysVarStore( SV_DELTAX, 7.5 );
      SysVarStore( SV_XSCL, FLOATTAB[FPI_90] );
   }  else {
      SysVarStore( SV_DELTAX, FLOATTAB[FPI_PIDIV24] );
      SysVarStore( SV_XSCL, round14(FLOATTAB[FPI_PIDIV2]) );
   }
   TempF = -(fr[GR_DELTAX] * xp) / FLOATTAB [FPI_2];
   SysVarStore( SV_XMIN, round12_err( TempF, ER_RANGE ) );
   SysVarStore( SV_XMAX, round12_err(fr[GR_DELTAX] * xp + TempF, ER_RANGE) );
   SysVarStore( SV_YMIN, -4.0 );
   SysVarStore( SV_YMAX, 4.0 );
   SysVarStore( SV_YSCL, 0.5 );
   GT_Regraph();
}
```

round14

Declaration: double **round14** (double *x*)

Category(ies): Direct Floating Point Operations

Description: Rounds *x* to 14 digits of precision.

Inputs: *x* — A double floating-point value.

Outputs: The value of *x* rounded to 14 digits of precision.

Assumptions: None

Side Effects: None

Availability: All versions of the TI-89 / TI-92 Plus.

TI-89 / TI-92 Plus Differences: None

See Also: **round12**

Example:

sin

Declaration: double **sin** (double *x*)

Category(ies): Direct Floating Point Operations

Description: Computes the sine of *x* radians.

Inputs: *x* — A double floating-point value.

Outputs: For -1.e13 < *x* < 1.e13 returns the sine of *x* radians.
For *x* <= -1.e13 or *x* >= 1.e13 returns a floating-point NAN.

Assumptions: None

Side Effects: None

Availability: All versions of the TI-89 / TI-92 Plus.

TI-89 / TI-92 Plus Differences: None

See Also: **acos, asin, atan, cos, tan**

Example:

sinh

Declaration: double **sinh** (double *x*)

Category(ies): Direct Floating Point Operations

Description: Computes the hyperbolic sine of *x*.

Inputs: *x* — A double floating-point value.

Outputs: The hyperbolic sine of *x*.

Assumptions: None

Side Effects: None

Availability: All versions of the TI-89 / TI-92 Plus.

TI-89 / TI-92 Plus Differences: None

See Also: **acosh, asinh, atanh, cosh, tanh**

Example:

sqrt

Declaration: double **sqrt** (double *x*)

Category(ies): Direct Floating Point Operations

Description: Computes the square root of *x*.

Inputs: *x* — A double floating-point value.

Outputs: The square root of *x*.

Assumptions: None

Side Effects: None

Availability: All versions of the TI-89 / TI-92 Plus.

TI-89 / TI-92 Plus Differences: None

See Also: **exp, log, pow, sqrt**

Example:

tan

Declaration: double **tan** (double *x*)

Category(ies): Direct Floating Point Operations

Description: Computes the tangent of *x* radians.

Inputs: *x* — A double floating-point value.

Outputs: For -1.e13 < *x* < 1.e13 returns the tangent of *x* radians.
For *x* <= -1.e13 or *x* >= 1.e13 returns a floating-point NAN.

Assumptions: None

Side Effects: None

Availability: All versions of the TI-89 / TI-92 Plus.

TI-89 / TI-92 Plus Differences: None

See Also: **acos, asin, atan, cos, sin**

Example:

tanh

Declaration: double **tanh** (double *x*)

Category(ies): Direct Floating Point Operations

Description: Computes the hyperbolic tangent of *x*.

Inputs: *x* — A double floating-point value.

Outputs: The hyperbolic tangent of *x*.

Assumptions: None

Side Effects: None

Availability: All versions of the TI-89 / TI-92 Plus.

TI-89 / TI-92 Plus Differences: None

See Also: **acosh, asinh, atanh, cosh, sinh**

Example:

Appendix A: System Routines — Display

ClientToScr

Declaration: void **ClientToScr** (const SCR_RECT * *csr*, const SCR_RECT * *sr*, SCR_RECT * *retScrRect*)

Category(ies): Display

Description: Add the corresponding x and y values in two SCR_RECTs, *csr* and *sr*; and return the result in *retScrRect*. This is usually used to convert a WINDOW based SCR_RECT to screen-based coordinates by adding a WINDOW based SCR_RECT to the WINDOW's client region.

Inputs:
- *csr* — First SCR_RECT.
- *sr* — Second SCR_RECT.

Outputs: *retScrRect* — Sum of *csr* and *sr* (coordinates added together).

Assumptions: None

Side Effects: None

Availability: On AMS 2.00 and higher.

TI-89 / TI-92 Plus Differences: None

See Also: **ScrToWin**

Example: See **WinHome**.

display_statements

Declaration: HANDLE **display_statements** (EStackIndex *i*, Boolean *replace_newlines*, Boolean *full_precision*)

Category(ies): Display

Description: Converts the external-tokenized form of an expression, statement, or group of statements to linear ASCII text form.

Inputs:

i — EStackIndex of external-tokenized expression, statement, or group of statements.

replace_newlines — TRUE to replace '\r' characters with ':'; FALSE to leave '\r' characters as-is.

full_precision — TRUE to display floating-point numbers with full available precision. FALSE to display floating-point numbers with precision determined by MODE screen settings.

Outputs: Returns the HANDLE to a heap packet which contains the ASCII text result; returns H_NULL if memory full.

Assumptions: None

Side Effects: May cause heap compression.

Availability: All versions of the TI‑89 / TI‑92 Plus.

TI‑89 / TI‑92 Plus Differences: None

See Also: **Parse1DExpr**

Example:

```
/* Assume i is the EStackIndex of the tagged floating-point number 3.1415926535898
   and that the current MODE setting for Display Digits is FLOAT 6, then */

HANDLE h;
h = display_statements (i, TRUE, TRUE);

/* returns a HANDLE to a memory block containing the full precision
   ASCII text 3.1415926535898. However, */

h = display_statements (i, TRUE, FALSE);

/* returns a HANDLE to a memory block containing the FLOAT 6 precision
   ASCII text 3.14159. */
```

DrawStrWidth

Declaration: WORD **DrawStrWidth** (const char * *Str*, BYTE *FontType*)

Category(ies): Display

Description: Return the length of the given string in pixels. For the 8x10 and 6x8 fonts this is just 8/6 times the length of the string; but the 4x6 font is proportional.

Inputs: *Str* — Pointer to string to return the length of.

FontType — F_4x6, F_6x8, F_8x10

Outputs: Length of string in pixels for the given font.

Assumptions: None

Side Effects: None

Availability: All versions of the TI‑89 / TI‑92 Plus.

TI‑89 / TI‑92 Plus Differences: None

See Also: **DrawStrWidthP**

Example:

```
WORD sLen;
sLen = DrawStrWidth("Test string", F_8x10 );   /* sLen = 88 */
sLen = DrawStrWidth("Test string", F_6x8 );    /* sLen = 66 */
sLen = DrawStrWidth("Test string", F_4x6 );    /* sLen = 36 */
```

DrawStrWidthP

Declaration: WORD **DrawStrWidthP** (const char * *Str*, short *strLen*, BYTE *FontType*)

Category(ies): Display

Description: Return the length of the given string in pixels. For the 8x10 and 6x8 fonts this is just 8/6 times the length of the string; but the 4x6 font is proportional. This is the same as **DrawStrWidth** only the caller passes the length of the string versus passing a zero-byte terminated string.

Inputs:

Str	—	Pointer to string to return the length of (not required to be zero-byte terminated).
strLen	—	Length of string.
FontType	—	F_4x6, F_6x8, F_8x10

Outputs: Length of string in pixels for the given font.

Assumptions: None

Side Effects: None

Availability: On AMS 2.00 and higher.

TI‑89 / TI‑92 Plus Differences: None

See Also: **DrawStrWidth**

Example:

```
WORD sLen;
sLen = DrawStrWidthP("Test string", 11, F_4x6 );   /* sLen = 36 */
```

Parms2D

Declaration: void **Parms2D** (EStackIndex *i*, WORD * *Width*, WORD * *Depth*, WORD * *Height*)

Category(ies): Display

Description: Return the width, depth, and height of a 2D expression.

Inputs:
i — EStackIndex of 2D expression (output of **Parse2DExpr** or **Parse2DMultiExpr**).

Outputs:
Width — Pointer to width in pixels.

Depth — Pointer to depth in pixels.

Height — Pointer to height in pixels.

Note that *Depth* + *Height* is equal to the total height of the 2D expression. Every 2D expression has an imaginary center line above which is the height and below (counting the line) is the depth.

Assumptions: Assumes the input came from **Parse2DExpr** or **Parse2DMultiExpr**.

Side Effects: None

Availability: All versions of the TI‑89 / TI‑92 Plus.

TI‑89 / TI‑92 Plus Differences: None

See Also: **Parse2DExpr, Parse2DMultiExpr**

Example: See **Print2DExpr**.

Parse1DExpr

Declaration: HANDLE **Parse1DExpr** (EStackIndex *i*, Boolean *full_precision*, unsigned short *width*)

Category(ies): Display

Description: Converts the external-tokenized form of an expression to linear ASCII text form.

Inputs:

i — EStackIndex of external-tokenized expression.

full_precision — TRUE to display floating-point numbers with full available precision; FALSE to display floating-point numbers with precision determined by MODE screen settings.

width — Maximum width of text result; 0 indicates no width restriction.

Outputs: Returns the HANDLE to a heap packet which contains the ASCII text result; returns H_NULL if memory full.

When necessary, symbolic expressions are truncated to *width*-1 characters and terminated with an ellipsis character (. . .). When *i* indexes a tagged floating-point number, the number is rounded to fit in *width* characters. When the number cannot be rounded to fit in *width* characters, an ellipsis character (. . .) is returned.

Assumptions: None

Side Effects: May cause heap compression.

Availability: All versions of the TI‑89 / TI‑92 Plus.

TI‑89 / TI‑92 Plus Differences: None

See Also: **display_statements**

(continued)

Parse1DExpr *(continued)*

Example:

```
/* Assume i is the EStackIndex of the tagged floating-point number 314.15926535898
   and that the current MODE setting for Display Digits is FLOAT 6, then */

HANDLE h;
h = Parse1DExpr (i, TRUE, 4);

/* returns a HANDLE to a memory block containing width restricted ASCII text 314. */

h = display_statements (i, TRUE, 3);

/* returns a HANDLE to a memory block containing the width restricted ASCII
   text . . . , because the number cannot be represented in 3 characters. */
```

Parse2DExpr

Declaration: EStackIndex **Parse2DExpr** (EStackIndex *i*, BOOL *FullPrec*)

Category(ies): Display

Description: Parse a CAS expression on the estack into a boxed RPN representing the 2D (graphical, as opposed to linear text) output of the RPN.

Inputs:

i — EStackIndex of CAS expression to convert to 2D.

FullPrec — If TRUE, floating-point numbers are converted to full float precision. Otherwise, they are printed with the current precision set by the MODE screen.

Outputs: Return the EStackIndex of the boxed RPN. If there is not enough memory then a special symbol is pushed onto the estack to signify this.

Assumptions: None

Side Effects: May compress the heap. May throw the following errors: ER_MEMORY (very low memory) and ER_BREAK (ON pressed while converting).

Availability: All versions of the TI-89 / TI-92 Plus.

TI-89 / TI-92 Plus Differences: None

See Also: **Parse2DMultiExpr, Print2DExpr**

Example: See **Print2DExpr**.

Parse2DMultiExpr

Declaration: EStackIndex **Parse2DMultiExpr** (HANDLE *hSrc*, BOOL *FullPrec*)

Category(ies): Display

Description: Parse a CAS expression on the estack into a 2D expression (graphical, as opposed to linear text) output of the input. The expression to be parsed may contain multiple expressions as follows (where * means 0 or more): END_OF_SEGMENT_TAG { expr SEPARATOR_TAG } * expr.

Compare this to **Parse2DExpr** which only parses one expression.

Inputs:

hSrc — HANDLE of data containing the CAS expression to convert to 2D (it will be locked and unlocked as needed).

FullPrec — If TRUE, floating-point numbers are converted to full float precision otherwise they are printed with the current precision set by the MODE screen.

Outputs: Return the EStackIndex of the boxed RPN. If there is not enough memory then a special symbol is pushed onto the estack to signify this.

Assumptions: None

Side Effects: May compress the heap. May throw the following errors: ER_MEMORY (very low memory) and ER_BREAK (ON pressed while converting).

Availability: All versions of the TI-89 / TI-92 Plus.

TI-89 / TI-92 Plus Differences: None

See Also: **Parse2DExpr, Print2DExpr**

Example: See **Print2DExpr** and substitute **Parse2DMultiExpr** for **Parse2DExpr**.

Print2DExpr

Declaration: void **Print2DExpr** (EStackIndex *i*, WINDOW * *Win*, SWORD *xOffset*, SWORD *yOffset*)

Category(ies): Display

Description: Print the 2D output from **Parse2DExpr** or **Parse2DMultiExpr** to a window (with defined offsets).

Inputs:

i	—	EStackIndex of 2D expression (output of **Parse2DExpr** or **Parse2DMultiExpr**).
Win	—	WINDOW structure to write to.
xOffset, *yOffset*	—	x and y offsets of where to start drawing in the window.

Outputs: Draws to *Win.*

Assumptions: Assumes the input came from **Parse2DExpr** or **Parse2DMultiExpr**.

Side Effects: Throws an ILLEGAL_TAG_ERROR if any invalid tags in the input.

Availability: All versions of the TI-89 / TI-92 Plus.

TI-89 / TI-92 Plus Differences: None

See Also: **Parse2DMultiExpr, Parse2DExpr**

Example: See **WinScrollH** for an example of a scrollable 2D viewer.

```
WORD Width, Depth, Height, Key;
short y;
EStackIndex origEStack, i2D;
BYTE Buf[128];

WinClr( &appW );
y = 0;   /* appW setup when app started */
strcpy( (char *) Buf, "((1-X)^2/(1+X))" );   /* Buf could be input by user */
origEStack = top_estack;   /* save top of  estack */
TRY
   push_quantum( END_OF_SEGMENT_TAG );   /* mark end of expression */
   push_parse_text( Buf );   /* tokenized text */
   i2D = Parse2DExpr( top_estack, 1 );   /* convert to display rpn */
   Parms2D( i2D, &Width, &Depth, &Height );   /* get parameters of display rpn */
   y += Height + 2;   /* update y cursor by height of expression */
   Print2DExpr( i2D, &appW, 0, y );   /* print to our window */
   GKeyIn( NULL, 0 );   /* wait */
   y += Depth;   /* update y cursor by depth */
ONERR
   top_estack = origEStack;   /* restore estack if error */
   PASS;   /* and pass error on up the chain */
ENDTRY
top_estack = origEStack;   /* restore estack */
```

sf_width

Declaration: UCHAR **sf_width** (UCHAR *ch*)

Category(ies): Display

Description: Return the width in pixels of a small font character.

Inputs: *ch* — Character to get width of.

Outputs: Width of given character in the small font.

Assumptions: Note that the small font (4x6) is the only proportional font; all the other fonts (large and huge) are fixed width.

Side Effects: None

Availability: On AMS 2.00 and higher.

TI-89 / TI-92 Plus Plus Differences: The small font is generally used more on the TI-89 than on the TI-92 Plus. It allows for more characters to fit on a line since it is proportional.

See Also: **DrawStrWidth, DrawStrWidthP**

Example: The **DrawStrWidth** function uses **sf_width** to compute the width of small (4x6) font characters.

```
/* Return the length of the given string in pixels.
   For the 8x10/6x8 font this is just 8/6 times the length of the string,
   but the 4x6 font is proportional.
*/
WORD DrawStrWidth( const char *Str, BYTE FontType )
{  register BYTE c;
   register WORD Width;

   switch( FontType ) {
      case F_4x6:
         Width = 0;
         while (c = *Str++)
            Width += sf_width (c);     /* proportional font */
         return Width;
      case F_6x8:  return (6 * strlen((char *) Str));
      case F_8x10: return (strlen(Str) << 3);
   }
}
```

Appendix A: System Routines — Error Handling

See Also:

clear_error_context

Declaration: void **clear_error_context** (void)

Category(ies): Error Handling

Description: Resets the internal variables associated with error handling. This subroutine is called when an error throw is caught and not passed.

Inputs: None

Outputs: None

Assumptions: None

Side Effects: None

Availability: On AMS 2.00 and higher.

TI-89 / TI-92 Plus Differences: None

See Also: None

Example:

```
/* This subroutine is called when an error throw is caught and not passed.
   For example, when the calculator user presses the ON key, an ER_BREAK is thrown.
   The following code segment passes all other codes but clears the ER_BREAK and
   continues processing.
*/
TRY
   /* code that may cause an error throw */
   .
   .
   .
ONERR
   if (errCode != ER_BREAK)
      PASS;
   clear_error_context ( );
ENDTRY
```

ER_catch

Declaration: SINT **ER_catch** (ER_ENVIRONMENT *throwBuf*)

Category(ies): Error Handling

Description: This routine is used in the implementation of the **TRY** macro. It pushes the current execution context on the exception stack. See chapter **10. Error Handling** for a complete discussion of throwing and catching exceptions.

Inputs: *throwBuf* — Saved execution context.

Outputs: Error code.

Assumptions: None

Side Effects: None

Availability: All versions of the TI‑89 / TI‑92 Plus.

TI‑89 / TI‑92 Plus Differences: None

See Also: **ER_success, TRY**

Example: Do not code a call to **ER_catch** yourself — use the **TRY** macro instead.

ER_success

Declaration: void **ER_success** (void)

Category(ies): Error Handling

Description: This routine is used in the implementation of the **ONERR** and **FINALLY** macros. It pops the exception stack. See chapter **10. Error Handling** for a complete discussion of throwing and catching exceptions.

Inputs: None

Outputs: None

Assumptions: None

Side Effects: None

Availability: All versions of the TI‑89 / TI‑92 Plus.

TI‑89 / TI‑92 Plus Differences: None

See Also: **ER_catch, FINALLY, ONERR**

Example: Do not code a call to **ER_success** yourself — use the **ONERR** or **FINALLY** macros instead.

ER_throwFrame

Declaration: void **ER_throwFrame** (int *errStringNum*, pFrame *appFrame*)

Category(ies): Error Handling

Description: This routine throws an exception to be caught in the **ONERR** or **FINALLY** section of an enclosing **TRY** block.

See chapter **10. Error Handling** for a complete discussion of throwing and catching exceptions.

Inputs:

errStringNum — Application error number signifying cause of exception. This should be an integer *n* in the range OO_FIRST_APP_STRING to OO_FIRST_APP_STRING+0x6FF. The system error handler displays the text of string OO_FIRST_STRING+*n* from your app's frame.

appFrame — Pointer to application's frame. This parameter should be the variable containing the pointer to the frame described in section **7.3.1.2 Pointer to FRAME**.

Outputs: None. This routine never returns — execution resumes at the **ONERR** or **FINALLY** section of the enclosing **TRY** block.

Assumptions: None

Side Effects: None

Availability: On AMS 2.00 and higher.

TI-89 / TI-92 Plus Differences: None

See Also: **ER_throw, ER_throwVar, FINALLY, ONERR, TRY**

(continued)

ER_throwFrame *(continued)*

Example:

```
/* App error numbers begin with OO_FIRST_APP_STRING */
#define ER_COUNT_TOO_LONG (OO_FIRST_APP_STRING+1)
FRAME(appObj, OO_SYSTEM_FRAME, ...)
   .
   .
   .
   /* String numbers begin with OO_FIRST_STRING */
   ATTR(OO_FIRST_STRING+ER_COUNT_TOO_LONG,"COUNT MUST BE <= 1024")
ENDFRAME

pFrame pAppObj = (pFrame)&appObj;
   .
   .
   .
unsigned long count = estack_to_ulong(top_estack);
if (count > 1024)
   ER_throwFrame(ER_COUNT_TOO_LONG, pAppObj);  /* Throw app error number */
```

ER_throwVar

Declaration: void **ER_throwVar** (SINT *errorCode*)

Category(ies): Error Handling

Description: This routine throws an exception to be caught in the **ONERR** or **FINALLY** section of an enclosing **TRY** block. See chapter **10. Error Handling** for a complete discussion of throwing and catching exceptions.

Inputs: *errorCode* — Error number signifying cause of exception. This can be a predefined error number (see “ER_” macros) or an application-specific error number with corresponding error string in the app’s frame.

Outputs: None. This routine never returns — execution resumes at the **ONERR** or **FINALLY** section of the enclosing **TRY** block.

Assumptions: Do not throw errors while processing events CM_START, CM_ACTIVATE, CM_FOCUS, CM_UNFOCUS, CM_DEACTIVATE, CM_QUIT, CM_WPAINT, CM_INSTALL, CM_UNINSTALL, CM_PACK, or CM_UNPACK. See **EV_errorCode** and section **10.2 Delayed Error Messages** for an alternative to throwing errors.

Side Effects: None

Availability: All versions of the TI‑89 / TI‑92 Plus.

TI‑89 / TI‑92 Plus Differences: None

See Also: **ER_throw, ER_throwFrame, EV_errorCode, FINALLY, ONERR, TRY**

Example:

```
TRY
   HANDLE h;
   h = HeapAlloc(A_BIG_BUFFER);
   if (h == H_NULL)
      ER_throwVar(ER_MEMORY);
   .
   .
   .
   HeapFree(h)
ONERR
   /* Do something about low memory condition */
   .
   .
   .
ENDTRY
```

ERD_dialog

Declaration: BOOL **ERD_dialog** (SSHORT *errno*, BOOL *bGoto*)

Category(ies): Error Handling

Description: Looks up and displays text of error message for error number *errno* in a dialog box. This routine waits for the user to press ESC or ENTER to dismiss the dialog box.

The error dialog box always displays an [ESC=CANCEL] button. Additionally, the [Enter=GOTO] button is displayed if *bGoto* is TRUE. This routine does not perform the actual transfer to the program editor if the user presses ENTER.

This routine is the core of the user interface to the system error handler. It is rarely necessary to call this routine directly. **ER_throwVar** is usually adequate to handle error conditions.

Inputs:

errno — Number of system error message.

bGoto — TRUE means add [Enter=GOTO] button to dialog box. FALSE means omit the [Enter=GOTO] button.

Outputs: Returns TRUE if the [Enter=GOTO] button is visible and the user pressed ENTER. Otherwise returns FALSE.

Assumptions: *errno* must be the number of a system error — do not use this routine to display application error messages.

Side Effects: May cause heap compression.

Availability: All versions of the TI-89 / TI-92 Plus.

TI-89 / TI-92 Plus Differences: None

See Also: **ER_throwVar**

Example:

```
ERD_dialog(ER_MEMORY, FALSE);   /* display ERROR: MEMORY dialog box */
/* Execution returns here after user presses [ENTER] or [ESC] */
```

find_error_message

Declaration: const unsigned char * **find_error_message** (int *error_code*)

Category(ies): Error Handling

Description: Returns a pointer to the message associated with a specified error_code.

Inputs: *error_code* — An error code.

Outputs: A pointer to the text message associated with the specified error code. If the input is not a valid error code, then the routine returns a pointer to the string “Unknown ERROR code”.

Assumptions: None

Side Effects: None

Availability: On AMS 1.05 and higher.

TI-89 / TI-92 Plus Differences: None

See Also: None

Example:

```
unsigned char * cp = find_error_message (ER_DATATYPE);
```

Assigns to cp a pointer to the string “Data type”.

Appendix A: System Routines — EStack Arithmetic

See Also:

add_to_top

Declaration: void **add_to_top** (EStackIndex *i*)

Category(ies): EStack Arithmetic

Description: Replaces the expression on the top of the estack with the internally-simplified sum of the top plus the expression indexed by *i*. If one operand is a scalar and the other is a square matrix, the scalar is distributed only over the diagonal of the matrix.

Inputs: *i* — Index to an internally-simplified algebraic expression, algebraic comparison, or aggregate thereof.

Outputs: None

Assumptions: *i* and **top_estack** point to the top tag of internally-simplified algebraic expressions, algebraic comparisons, or aggregates thereof.

Side Effects: May cause heap compression or throw an error.

Availability: On AMS 2.02 and higher.

TI-89 / TI-92 Plus Differences: None

See Also: **push_sum, replace_top2_with_sum, push_difference, subtract_from_top, replace_top2_with_difference, add1_to_top, subtract1_from_top, push_arg_plus_1, push_arg_minus_1**

Example:

```
void add1_to_top (void)
/* Add 1 or 1.0 the top expression, depending on global computation_mode. */
{  add_to_top (IS_ARITH_APPROX ? Float1Index : Integer1Index);
}
```

add1_to_top

Declaration: void **add1_to_top** (void)

Category(ies): EStack Arithmetic

Description: Replaces the expression on the top of the estack with the internally-simplified sum of the top plus one. If IS_ARITH_APPROX is true, a floating-point one (1.0) is added. Otherwise, a tagged integer one (1) is added. If the top of the estack is a square matrix, the one is added only to the diagonal of the matrix.

Inputs: None

Outputs: None

Assumptions: The top of the estack is an internally-simplified algebraic expression, algebraic comparison, or aggregate thereof.

Side Effects: May cause heap compression or throw an error.

Availability: On AMS 2.02 and higher.

TI‑89 / TI‑92 Plus Differences: None

See Also: **push_sum, replace_top2_with_sum, push_difference, subtract_from_top, replace_top2_with_difference, add_to_top, subtract1_from_top, push_arg_plus_1, push_arg_minus_1**

Example:

```
void push_unit_step (EStackIndex i)
/* Pushes (sign(i) + 1)/2 onto the estack. */
{  Access_AMS_Global_Variables;
   push_sign (i);
   add1_to_top ();
   i = top_estack;
   push_reciprocal_of_quantum (2u);
   replace_top2_with_prod (i);
}
```

can_be_approxed

Declaration: Boolean **can_be_approxed** (EStackIndex *i*, Boolean *allow_complex*)

Category(ies): EStack Arithmetic

Description: Determines if an expression can be approxed.

Inputs:

i — Index of the top tag of an expression.

allow_complex — TRUE if and only if nonreal numbers are allowed in expression *i*.

Outputs: Returns TRUE if *allow_complex* is FALSE and the expression indexed by *i* can be approximated to a (perhaps transfinite) number or a list thereof or if *allow_complex* is TRUE, and the expression indexed by *i* can be approximated to a (perhaps transfinite) complex number or a list thereof. Otherwise returns FALSE.

Assumptions: *i* points to an expression in the estack or some other locked block.

Side Effects: None

Availability: On OS 1.05 and higher.

TI-89 / TI-92 Plus Differences: None

See Also: **is_constant**

(continued)

can_be_approxed *(continued)*

Example:

```
void push_colnorm (EStackIndex matrix_idx)
/*  Pushes the largest of the sums of the absolute values of the elements
    in each column of the approximatable matrix indexed by matrix_idx.
*/
{  Access_AMS_Global_Variables;
   EStackIndex old_top = top_estack;
   EStackIndex i, j;
   if (is_matrix (matrix_idx))
   {  if (! can_be_approxed (matrix_idx, TRUE))
         ER_THROW (ER_DOMAIN);
      i = matrix_idx - 1u;
      push0 ();
      while (END_TAG != ESTACK (i))
      {  j = top_estack;
         push_abs (i);
         replace_top2_with_sum (j);
         i = next_expression_index (i);
      }
      j = top_estack;
      push_max1 (j);
      delete_between (old_top, j);
   }
   else
   {  /* error - data type */
      ER_throw( ER_DATATYPE );
   }
}
```

compare_complex_magnitudes

Declaration: int **compare_complex_magnitudes** (IndexConstQuantum *j*, IndexConstQuantum *k*)

Category(ies): EStack Arithmetic

Description: Compares the magnitude of the complex number indexed by *j* with the complex number indexed by *k*.

Inputs: *j*, *k* — Indexes of tagged real or unreal numbers.

Outputs: Returns 1 if the complex number indexed by *j* has a greater magnitude than the complex number indexed by *k*. Returns -1 if the complex number indexed by *j* has a greater magnitude than the complex number indexed by *k*. Returns 0 if the complex number indexed by *j* has the same magnitude as the complex number indexed by *k*. Remember that complex numbers include real numbers. Avoids square roots even if *j* and/or *k* index nonreal numbers.

Assumptions: *j* and *k* point to the tags of tagged real or unreal numbers.

Note that PI_TAG and INFINITY_TAG are not considered numbers here.

Side Effects: Temporarily attempts to push items onto the estack.

Availability: On AMS 1.05 and higher.

TI-89 / TI-92 Plus Differences: None

See Also: **compare_numbers, compare_Floats, compare_expressions**

Examples:

```
Boolean has_larger_complex_magnitude (EStackIndex i, EStackIndex j)
{  return compare_complex_magnitudes(j,k) > 0;
}
```

compare_Floats

Declaration: int **compare_Floats** (IndexConstQuantum *i*, IndexConstQuantum *j*)

Category(ies): EStack Arithmetic

Description: Compares tthe tagged floats indexed by *i* and *j*.

Inputs: *i, j* — Indexes to tagged floats.

Outputs: Returns 0 if the tagged floats indexed by *i* and *j* are equal, 1 if the tagged float indexed by *i* is larger, or -1 if the tagged float indexed by *j* is larger.

Assumptions: *i* and *j* point to the tags of tagged floats.

Side Effects: None

Availability: On AMS 1.05 and higher.

TI-89 / TI-92 Plus Differences: None

See Also: **compare_numbers, compare_complex_magnitudes, compare_expressions**

Examples:

```
push_Float (3.7);
j = top_estack;
push_Float (5.2);
i = top_estack;
compare_Floats (i, j);  /* Returns 1 */
```

compare_numbers

Declaration: int **compare_numbers** (EStackIndex *i*, EStackIndex *j*)

Category(ies): EStack Arithmetic

Description: Compares the two numbers indexed by *i* and *j*.

Inputs: *i, j* — Indexes to tagged numbers.

Outputs: Returns 0 if the numbers indexed by *i* and *j* are equal, even if one is Float and one is rational. Returns a positive integer if the number indexed by *i* is greater than the number indexed by *j*. Returns a negative integer if the number indexed by *i* is less than the number indexed by *j*. For this function, all zeros are considered equal: 0 = +0 = -0 = +0.0 = -0.0 = 0.0.

Assumptions: *i* and *j* point to the numeric tags of tagged numbers. Note that PI_TAG and IM_RE_TAG are not considered numeric tags.

Side Effects: Might temporarily push items onto the estack.

Availability: On AMS 2.02 and higher.

TI-89 / TI-92 Plus Differences: None

See Also: **compare_Floats, compare_complex_magnitudes, compare_expressions**

Examples:

```
/* Return j if it indexes a larger magnitude number than k indexes.
   Otherwise return k: */
return compare_numbers(j,k) > 0 ? j : k;
```

did_push_cnvrt_Float_to_integer

Declaration: Boolean **did_push_cnvrt_Float_to_integer** (EStackIndex *i*)

Category(ies): EStack Arithmetic

Description: If the fractional part of the float indexed by *i* is any zero and the number is representable as a big integer, pushes the tagged big-integer equivalent.

Inputs: *i* — Index of a tagged float.

Outputs: Returns TRUE if a tagged big-integer is pushed. Otherwise returns FALSE.

Assumptions: *i* points to the top tag of a tagged float.

Side Effects: None

Availability: On AMS 1.05 and higher.

TI-89 / TI-92 Plus Differences: None

See Also: **is_whole_number, is_Float_exact_whole_number, push_cnvrt_integer_if_whole_nmb, push_floor, push_ceiling, push_integer_part**

Example:

```
void push_cnvrt_integer_if_whole_nmb (EStackIndex i)
/* i indexes an expression.
   If it is a whole number that is representable as an integer, pushes the
   integer equivalent. Otherwise pushes expression i.
*/
{  if (FLOAT_TAG == ESTACK (i) && did_push_cnvrt_Float_to_integer (i))
      ;
   else
      push_expression (i);
}
```

divide_top

Declaration: void **divide_top** (EStackIndex *i*)

Category(ies): EStack Arithmetic

Description: Replaces the top expression on the estack with the internally-simplified ratio of the top divided by the expression indexed by *i*.

Inputs: *i* — Index to an internally-simplified algebraic expression, algebraic comparison, or aggregate thereof.

Outputs: None

Assumptions: *i* and **top_estack** point to the top tags of algebraic expressions, algebraic comparisons, or aggregates thereof.

Side Effects: None

Availability: On AMS 2.02 and higher.

TI-89 / TI-92 Plus Differences: None

See Also: **push_ratio, replace_top2_with_ratio, push_reciprocal, replace_top_with_reciprocal**

Example:

```
void push_coef_list (EStackIndex i, EStackIndex ki)
/* i indexes a polynomial in kernel ki.
   Pushes a tail of its coefficients (including zero-coefficients),
   with the constant coefficient deepest.
*/
{  Access_AMS_Global_Variables;
   EStackIndex j, ans;
   push_quantum (END_TAG);
   ans = top_estack;
   push_expression (i);
   i = top_estack;
   while (! is_independent_of (i, ki))
      {  push_substitute_simplify (i, ki, Integer0Index);
         j = top_estack;
         push_difference (i, j);
         divide_top (ki);
         ans = j - deleted_between (ans, i);
         i = top_estack;
      }
}
```

get_lb

Declaration: float **get_lb** (EStackIndex *var*)

Category(ies): EStack Arithmetic

Description: If global **NG_such_that_index** includes *var* >= expression, and approx (expression) -> float, returns that float. Otherwise if global **NG_such_that_index** includes *var* > expression, and approx (expression) -> float, returns that float − epsilon, where epsilon = 1E-38 if approx (x) == 0.0 or 8 ∗ FLOAT_EPSILON ∗ ABS (Float) if approx (x) != 0.0. Otherwise returns FLOAT_MAX.

Inputs: *var* — Index of a variable.

Outputs: None

Assumptions: None

Side Effects: None

Availability: On AMS 2.02 and higher.

TI-89 / TI-92 Plus Differences: None

See Also: **get_ub**

Example:

```
Int var_gt_eq_other_const (EStackIndex ki, EStackIndex j)
/* ki indexes a kernel and j indexes a number.
   Using NG_such_that_index:
      returns 1 if deduces the variable indexed by ki > the number indeded by j,
      returns 0 if deduces the variable indexed by ki = the number indeded by j,
      returns -1 otherwise.
*/
{  Access_AMS_Global_Variables;
   Float fki = get_lb (ki);
   if (fki < FLOAT_MAX)
      {  ki = top_estack;
         push_approx (j);
         j = top_estack;
         if (FLOAT_TAG == ESTACK (j))
           {  Float fj = ESTACK_TO_FLOAT (j);
              top_estack = ki;
              return fki < fj ? -1 : fki > fj;
           }
      }
   return -1;
}
```

get_ub

Declaration: float **get_ub** (EStackIndex *var*)

Category(ies): EStack Arithmetic

Description: If global **NG_such_that_index** includes *var* <= expression, and approx (expression) -> float, returns that float. Otherwise if global **NG_such_that_index** includes *var* < expression, and approx (expression) -> float, returns that float + epsilon, where epsilon = 1E-38 if approx (x) == 0.0 and 8 * FLOAT_EPSILON * ABS (Float) if approx (x) != 0.0. Otherwise returns FLOAT_MIN.

Inputs: *var* — Index to a variable.

Outputs: None

Assumptions: None

Side Effects: None

Availability: On AMS 2.02 and higher.

TI-89 / TI-92 Plus Differences: None

See Also: **get_lb**

Example:

```
Int var_lt_eq_other_const (EStackIndex ki, EStackIndex j)
/* ki indexes a kernel and j indexes a number.
   Using NG_such_that_index:
      returns 1 if deduces the variable indexed by ki < the number indexed by j,
      returns 0 if deduces the variable indexed by ki = the number indexed by j,
      returns -1 otherwise.
*/
{  Access_AMS_Global_Variables;
   Float fki = get_ub (ki);
   if (fki > -FLOAT_MAX)
      {  ki = top_estack;
         push_approx (j);
         j = top_estack;
         if (FLOAT_TAG == ESTACK (j))
            {  Float fj = ESTACK_TO_FLOAT (j);
               top_estack = ki;
               return fki > fj ? -1 : fki < fj;
            }
      }
   return -1;
}
```

integer_non_unknown

Declaration: int **integer_non_unknown** (EStackIndex *i*)

Category(ies): EStack Arithmetic

Description: Determines if the expression indexed by *i* can have whole-number values.

Inputs: *i* — Index to an algebraic expression.

Outputs: Returns 1 if the function determines that the expression indexed by *i* can have only whole-number values. Returns -1 if the function determines that the expression cannot have a whole-number value. Returns 0 if the function cannot determine either of the above. The argument expression does not have to be numeric.

Assumptions: *i* indexes the top tag of an algebraic expression.

Side Effects: None

Availability: On AMS 2.02 and higher.

TI-89 / TI-92 Plus Differences: None

See Also: **is_whole_number, is_Float_exact_whole_number**

Examples:

```
Boolean is_even_expression (EStackIndex i)
/* Returns TRUE if the expression indexed by i is recognized as even. */
{  Access_AMS_Global_Variables;
   EStackIndex old_top = top_estack;
   Boolean b;
   push_reciprocal_of_quantum (2u);
   times_top (i);
   b = integer_non_unknown (top_estack) > 0;
   top_estack = old_top;
   return b;
}
```

is_cFloat_agg

Declaration: Boolean **is_cFloat_agg** (EStackIndex *i*)

Category(ies): EStack Arithmetic

Description: Tests whether every element of a scalar or aggregate is a float or complex float.

Inputs: *i* — EStackIndex of a scalar expression, a list, or a matrix in internal tokenized form.

Outputs: Returns TRUE if every element of the input is a float or complex float in internal tokenized form.

Assumptions: None

Side Effects: None

Availability: On AMS 2.00 and higher.

TI-89 / TI-92 Plus Differences: None

See Also: **is_complex_Float, is_complex_number**

(continued)

is_cFloat_agg *(continued)*

Example:

If j indexes the bolded tag in the following estack representation of the number 0
0 **NONNEGATIVE_INTEGER_TAG**

then
```
is_cFloat_agg (j);
```

returns TRUE.

If j indexes the bolded tag in the following estack representation of the number $2 - 3i$
2 1 NONNEGATIVE_INTEGER_TAG 3 1 NEGATIVE_INTEGER_TAG **IM_RE_TAG**

then
```
is_cFloat_agg (j);
```

returns TRUE.

If j indexes the bolded tag in the following estack representation of the list {0, 2-3i, x}
END_TAG X_VAR_TAG 2 1 NONNEGATIVE_INTEGER_TAG 3 1 NEGATIVE_INTEGER_TAG
0 NONNEGATIVE_INTEGER_TAG **LIST_TAG**

then
```
is_cFloat_agg (j);
```

returns FALSE, because x is not a float or complex float.

If j indexes the bolded tag in the following external tokenized form of the number $2 - 3i$
2 1 NONNEGATIVE_INTEGER_TAG 3 1 NONNEGATIVE_INTEGER_TAG I_TAG
MULTIPLY_TAG **SUBTRACT_TAG**

then
```
is_cFloat_agg (j);
```

returns FALSE, because the input is not in an internal tokenized form.

is_complex_Float

Declaration: Boolean **is_complex_Float** (IndexConstQuantum *i*)

Category(ies): EStack Arithmetic

Description: Determines whether the expression indexed by *i* is a complex float.

Inputs: *i* — Index to an expression.

Outputs: Returns TRUE if *i* indexes a FLOAT_TAG or an IM_RE_TAG on top of two expressions having float tags. Otherwise returns FALSE.

Assumptions: *i* indexes the top tag of an expression.

Side Effects: None

Availability: On AMS 2.02 and higher.

TI-89 / TI-92 Plus Differences: None

See Also: **is_complex_number**

Examples:

```
Boolean is_cFloat_agg (EStackIndex i)
/* Returns TRUE if the expression indexed by i contains only floats
   and/or complex floats. Otherwise returns FALSE. */
{  return is_complex_Float(i) ||
          LIST_TAG == ESTACK(i) && all_tail (&is_cFloat_agg, i-1u);
}
```

is_complex0

Declaration: Boolean **is_complex0** (IndexConstQuantum *i*)

Category(ies): EStack Arithmetic

Description: Determines whether *i* indexes a real or unreal signed or unsigned zero.

Inputs: *i* — Index of an expression.

Outputs: Returns TRUE if *i* indexes a real or unreal signed or unsigned zero. Otherwise, returns FALSE.

Assumptions: *i* indexes the top tag of an expression.

Side Effects: None

Availability: On AMS 1.05 and higher.

TI-89 / TI-92 Plus Differences: None

See Also: **is0**

Example:

```
Boolean is_complex0_or_aggregate_thereof (EStackIndex i)
{  return LIST_TAG == ESTACK (i) ?
         all_tail (is_complex0_or_aggregate_thereof, i-1u) :
         is_complex0 (i);
}
```

is_complex_number

Declaration: Boolean **is_complex_number** (IndexConstQuantum *i*)

Category(ies): EStack Arithmetic

Description: Determines whether *i* indexes a complex number.

Inputs: *i* — Index of an expression.

Outputs: Returns TRUE if *i* indexes a numeric tag or an IM_RE_TAG on top of two expressions having numeric tags. Otherwise returns FALSE.

Assumptions: *i* indexes the top tag of an expression.

Side Effects: None

Availability: On AMS 1.05 and higher.

TI-89 / TI-92 Plus Differences: None

See Also: **is_complex_float**

Examples:

```
push_Float (3.7);
is_complex_number (top_estack);  /* Returns TRUE */
real_part = top_estack;
push_quantum_as_nonnegative_int (5u);
replace_top2_with_imre (real_part);
is_complex_number (top_estack);  /* Returns TRUE */
push_quantum (PI_TAG);
is_complex_number (top_estack);  /* Returns FALSE */
```

is_constant

Declaration: Boolean **is_constant** (EStackIndex *i*)

Category(ies): EStack Arithmetic

Description: Determines whether the expression indexed by *i* is a constant.

Inputs: *i* — Index to an expression.

Outputs: Returns TRUE if the expression indexed by *i* is free of all variables. Otherwise returns FALSE. Examples include expressions composed of real or unreal numbers, π, e, transfinites, true, false, and aggregates thereof.

Assumptions: *i* indexes the top tag of an expression.

Side Effects: None

Availability: On AMS 2.02 and higher.

TI-89 / TI-92 Plus Differences: None

See Also: **can_be_approxed, is_complex_number, is_complex_float**

Examples:

```
Boolean has_constant_term (EStackIndex i)
/* Returns TRUE if and only if the expression indexed by i has a zero
      top-level constant term. */
{  while (! is0 (i))
         if (is_constant (lead_term_index (i)))
            return TRUE;
         else
            i = reductum_index (i);
   return FALSE;
}  /* end has_constant_term */
```

is_Float_exact_whole_number

Declaration: Boolean **is_Float_exact_whole_number** (IndexConstQuantum *i*)

Category(ies): EStack Arithmetic

Description: Determines whether the float indexed by *i* is a whole number whose magnitude is <= MAX_EXACT_FLOAT_WHOLE_NUMBER.

Inputs: *i* — Index of a tagged float.

Outputs: Returns TRUE if the float indexed by *i* is a whole number whose magnitude is <= MAX_EXACT_FLOAT_WHOLE_NUMBER. Otherwise returns FALSE.

Assumptions: *i* points to the FLOAT_TAG of a tagged float.

Side Effects: None

Availability: On AMS 1.05 and higher.

TI-89 / TI-92 Plus Differences: None

See Also: **is_whole_number**

Examples:

```
Float gcdExactWholeFloats(IndexConstQuantum i, IndexConstQuantum j)
/* i and j index exact whole Floats.
   Returns their greatest common divisor, computed by Euclid's algorithm. */
{  Float f3, f1, f2;
   if (FLOAT_TAG == ESTACK (i) && FLOAT_TAG == ESTACK (j) &&
         is_Float_exact_whole_number (i) && is_Float_exact_whole_number (j))
   {
      f1 = fabs (ESTACK_TO_FLOAT (i));
      f2 = fabs (ESTACK_TO_FLOAT (j));
      if (f1 < f2)
         {  f3 = f1;                          /* Swap f1 and f2 */
            f1 = f2;
            f2 = f3;
         }
      while (f2 > FLOAT0)
         {  f3 = fmod (f1, f2);
            f1 = f2;
            f2 = f3;
         }
      return f1;
   }
   else
      ER_throw( ER_DOMAIN);
}
```

is_minus1

Declaration: Boolean **is_minus1** (IndexConstQuantum *i*)

Category(ies): EStack Arithmetic

Description: Determines whether *i* indexes a tagged big integer -1 or tagged float -1.

Inputs: *i* — Index of an expression.

Outputs: Returns TRUE if *i* indexes a tagged big integer -1 or tagged float -1. Otherwise returns FALSE.

Assumptions: *i* indexes the top tag of an expression.

Side Effects: None

Availability: On AMS 2.02 and higher.

TI-89 / TI-92 Plus Differences: None

See Also: **is0, is1**

Example:

```
Boolean is_plus_or_minus_one (IndexConstQuantum i)
/* i indexes an expression. Returns TRUE if the expression is an integer
   or Float 1 or -1. Otherwise returns FALSE.
*/
{  return is1 (i) || is_minus1 (i);
}
```

is_pos_int_and_eq_quantum

Declaration: Boolean **is_pos_int_and_eq_quantum** (IndexConstQuantum *i*, Quantum *q*)

Category(ies): EStack Arithmetic

Description: Determines whether the expression indexed by *i* is a positive big integer whose magnitude equals *q*.

Inputs: *i* — Index of an expression.

q — Nonzero quantum.

Outputs: Returns TRUE if the expression indexed by *i* is a positive big integer whose magnitude equals *q*. Otherwise, returns FALSE.

Assumptions: *i* points to the top tag of an expression, and *q* is nonzero.

Side Effects: None

Availability: On AMS 2.02 and higher.

TI-89 / TI-92 Plus Differences: None

See Also: None

Example:

```
push_quantum_as_nonnegative_int (5u);
is_pos_int_and_eq_quantum (top_estack,5u);
   /* Returns TRUE */

push_Float (5.0);
is_pos_int_and_eq_quantum (top_estack,5u);
   /* Returns FALSE */
```

is_reciprocal_of_quantum

Declaration: Boolean **is_reciprocal_of_quantum** (IndexConstQuantum *i*, Quantum *q*)

Category(ies): EStack Arithmetic

Description: Determines whether the expression indexed by *i* is a fraction that is the reciprocal of the nonzero quantum *q*.

Inputs: *i* — Index of an expression.

q — Nonzero quantum.

Outputs: Returns TRUE if the expression indexed by *i* is a fraction that is the reciprocal of the nonzero quantum *q*. Otherwise, returns FALSE.

Assumptions: *i* points to the top tag of an expression.

q is a nonzero quantum.

Side Effects: None

Availability: On AMS 2.02 and higher.

TI-89 / TI-92 Plus Differences: None

See Also: None

Example:

```
push_reciprocal_of_quantum (5u);
is_reciprocal_of_quantum (top_estack, 5u);  /* Returns TRUE */
```

is_whole_number

Declaration: Boolean **is_whole_number** (IndexConstQuantum *i*)

Category(ies): EStack Arithmetic

Description: Determines whether the expression indexed by *i* is a big integer, a float whose fractional part is 0.0, or any real zero.

Inputs: *i* — Index of an expression.

Outputs: Returns TRUE if the expression indexed by *i* is a big integer, a float whose fractional part is 0.0, or any real zero. Otherwise, returns FALSE.

Assumptions: *i* points to the top tag of an expression.

Side Effects: None

Availability: All versions of the TI‑89 / TI‑92 Plus.

TI‑89 / TI‑92 Plus Differences: None

See Also: **is_Float_exact_whole_number**

Example:

```
is_whole_number(i) -> TRUE if i indexes a tagged float 1.2345678901234e30
```

is0

Declaration: Boolean **is0** (IndexConstQuantum *i*)

Category(ies): EStack Arithmetic

Description: Determines whether *i* indexes any tagged big-rational or tagged float zero.

Inputs: *i* — Pointer to an expression.

Outputs: Returns TRUE if *i* indexes any tagged big-rational or tagged float zero. Otherwise, returns FALSE.

Assumptions: *i* indexes the top tag of an expression.

Side Effects: None

Availability: All versions of the TI‑89 / TI‑92 Plus.

TI‑89 / TI‑92 Plus Differences: None

See Also: **Is_complex0, is1, is_minus1**

Example:

```
push_Float (0.0);
is0(top_estack);   /* Returns TRUE */
```

is1

Declaration: Boolean **is1** (IndexConstQuantum *i*)

Category(ies): EStack Arithmetic

Description: Determines whether *i* indexes a tagged big integer 1 or tagged float 1.

Inputs: *i* — Indexes an expression.

Outputs: Returns TRUE if *i* indexes a tagged big integer 1 or tagged float 1. Otherwise, returns FALSE.

Assumptions: *i* points to the top tag of an expression.

Side Effects: None

Availability: On AMS 1.05 and higher.

TI-89 / TI-92 Plus Differences: None

See Also: **is0, is_minus1**

Example:

```
EStackIndex index_nonnegative_factor (EStackIndex i)
/* i indexes a term.
   Returns NULL_INDEX if it cannot determine that one of its explicit syntactic
   factors is nonnegative.
   Otherwise returns i or the index of its 1st remaining factors for which
   the lead factor has been determined to be nonnegative.
*/
{  while (! is1 (i))
      if (is_nonnegative (lead_factor_index (i)))
         return i;
      else
         i = remaining_factors_index (i);
   return NULL_INDEX;
}
```

negate_top

Declaration: void **negate_top** (void)

Category(ies): EStack Arithmetic

Description: Replaces the expression indexed by **top_estack** with its internally-simplified negative.

Inputs: None

Outputs: None

Assumptions: **top_estack** points to the top tag of an internally-simplified algebraic expression, algebraic comparison, or aggregate thereof.

Side Effects: None

Availability: On AMS 2.02 and higher.

TI-89 / TI-92 Plus Differences: None

See Also: **push_negate_quantum_as_negint**

Example:

```
void push_ceiling (EStackIndex i)
/* i indexes an internally-simplified algebraic expression or an aggregate thereof.
   Pushes the equivalent expression (- floor (-i)) onto the estack.
*/
{  Access_AMS_Global_Variables;
   EStackIndex old_top = top_estack;
   push_negate (i);
   i = top_estack;
   push_floor (i);
   delete_between (old_top, i);
   negate_top ();
}
```

push_arg_minus_1

Declaration: void **push_arg_minus_1** (EStackIndex *i*)

Category(ies): EStack Arithmetic

Description: Pushes onto the estack the internally-simplified difference of the expression *i* minus 1.0 if IS_ARITH_APPROX, or the expression *i* minus 1 otherwise. If *i* indexes a square matrix, the one is subtracted only from the diagonal of the matrix.

Inputs: *i* — Index to an internally-simplified algebraic expression, algebraic comparison, or aggregate thereof.

Outputs: None

Assumptions: *i* indexes the top tag of an internally-simplified algebraic expression, algebraic comparison, or aggregate thereof.

Side Effects: None

Availability: On AMS 2.02 and higher.

TI-89 / TI-92 Plus Differences: None

See Also: **push_sum, replace_top2_with_sum, push_difference, subtract_from_top, replace_top2_with_difference, add1_to_top, subtract1_from_top, push_arg_plus_1, add_to_top**

Example:

```
Boolean is_odd_expression (EStackIndex i)
/* Returns TRUE if the expression indexed by i is recognized as odd. */
{  Access_AMS_Global_Variables;
   EStackIndex old_top;
   Boolean b;
   if (SIGN_TAG == ESTACK (i))
      return IS_DOMAIN_REAL && is_real (i - 1u);
   old_top = top_estack;
   push_arg_minus_1 (i);
   i = top_estack;
   push_reciprocal_of_quantum (2u);
   replace_top2_with_prod (i);
   b = integer_non_unknown (top_estack) > 0;
   top_estack = old_top;
   return b;
}
```

push_arg_plus_1

Declaration: void **push_arg_plus_1** (EStackIndex *i*)

Category(ies): EStack Arithmetic

Description: Pushes onto the estack the internally-simplified sum of the expression *i* plus 1.0 if IS_ARITH_APPROX, or the expression *i* plus 1 otherwise. If *i* indexes a square matrix, the one is added only to the diagonal of the matrix.

Inputs: *i* — Index to an internally-simplified algebraic expression, algebraic comparison, or aggregate thereof.

Outputs: None

Assumptions: *i* points to the top tag of an internally-simplified algebraic expression, algebraic comparison, or aggregate thereof.

Side Effects: None

Availability: On AMS 2.02 and higher.

TI‑89 / TI‑92 Plus Differences: None

See Also: **push_sum, replace_top2_with_sum, push_difference, subtract_from_top, replace_top2_with_difference, add1_to_top, subtract1_from_top, add_to_top, push_arg_minus_1**

Example:

```
push_quantum (8u);  /* push variable x */
push_arg_plus_1 (top_estack);  /* Pushes x + 1 or x + 1.0 */
```

push_difference

Declaration: void **push_difference** (EStackIndex *i*, EStackIndex *j*)

Category(ies): EStack Arithmetic

Description: Pushes onto the estack the internally-simplified difference of the expression indexed by *i* minus the expression indexed by *j*. If one operand is a scalar and the other is a square matrix, the scalar is distributed only over the diagonal of the matrix.

Inputs: *i, j* — Index of top tags of internally-simplified algebraic expressions, algebraic comparisons, or aggregates thereof.

Outputs: None

Assumptions: None

Side Effects: None

Availability: On AMS 2.02 and higher.

TI-89 / TI-92 Plus Differences: None

See Also: **push_sum, replace_top2_with_sum, add_to_top, subtract_from_top, replace_top2_with_difference, add1_to_top, subtract1_from_top, push_arg_plus_1, push_arg_minus_1**

Example:

```
void push_coef_list (EStackIndex i, EStackIndex ki)
/* i indexes a polynomial in the variable or kernel indexed by ki.
   Pushes a tail of its coefficients (including zero-coefficients),
   with the constant coefficient deepest.
*/
{  Access_AMS_Global_Variables;
   EStackIndex j, ans;
   push_quantum (END_TAG);
   ans = top_estack;
   push_expression (i);
   i = top_estack;
   while (! is_independent_of (i, ki))
   {  push_substitute_simplify (i, ki, Integer0Index);
      j = top_estack;
      push_difference (i, j);
      divide_top (ki);
      ans = j - deleted_between (ans, i);
      i = top_estack;
   }
}
```

push_gcd_numbers

Declaration: void **push_gcd_numbers** (EStackIndex *i*, EStackIndex *j*)

Category(ies): EStack Arithmetic

Description: Pushes the gcd of the expressions indexed by *i* and *j* onto the estack. Pushes 1.0 if either argument is a Float for which **is_Float_exact_whole_number** (. . .) returns FALSE.

> **NOTE:** gcd(0, k) = abs(k)
> gcd(m/n, p/q) = gcd(m, n)/lcm(n, q)
> gcd (-0, 0) = gcd (+0, 0) = gcd (0, -0) = gcd (0, +0) = +0

Inputs: *i*, *j* — Indexes to the numeric tag of tagged numbers.

Outputs: None

Assumptions: None

Side Effects: None

Availability: On AMS 2.02 and higher.

TI-89 / TI-92 Plus Differences: None

See Also: **push_gcd_then_cofactors**

Examples:

```
push_reciprocal_of_quantum (2u);
fraction_half = top_estack;
push_Float (2.0);
float2 = top_estack;
push_gcd_numbers (fraction_half, float2);  /* Pushes 1/2 */
push_negate_quantum_as_negint (2u);
push_gcd_numbers (Integer0Index, top_estack);  /* Pushes 2 */
```

push_is_prime

Declaration: void **push_is_prime** (EStackIndex *i*)

Category(ies): EStack Arithmetic

Description: Pushes TRUE if the expression indexed by *i* is determined to be a prime exact whole number, such as 2, 2., 3, 3., 5, 5., etc. Pushes FALSE if it is any other number or a transfinite tag. Otherwise throws ER_DOMAIN.

> **NOTE:** Floats > MAX_EXACT_FLOAT_WHOLE_NUMBER are whole numbers, but not exact whole numbers.

Inputs: *i* — Index to the top tag of an expression.

Outputs: None

Assumptions: None

Side Effects: None

Availability: On AMS 2.02 and higher.

TI-89 / TI-92 Plus Differences: None

See Also: **push_factor**

Example:

```
push_Float (2.0);
push_is_prime (top_estack);          /* Pushes TRUE_TAG */
push_is_prime (Integer1Index);       /* Pushes FALSE_TAG */
push_negate_quantum_as_negint (2u);
push_is_prime (top_estack);          /* Pushes FALSE_TAG */
push_reciprocal_of_quantum (2u);
push_is_prime (top_estack);          /* Pushes FALSE_TAG */
```

push_minus_recip_of_quantum

Declaration: void **push_minus_recip_of_quantum** (Quantum *q*)

Category(ies): EStack Arithmetic

Description: Pushes tagged fraction -1/*q* onto the estack.

Inputs: *q* — Quantum.

Outputs: None

Assumptions: *q* != 0 and *q* != 1.

Side Effects: None

Availability: On AMS 2.02 and higher.

TI-89 / TI-92 Plus Differences: None

See Also: **push_reciprocal_of_quantum, push_quantum_pair_as_pos_frac**

Example:

```
void push_minus_pi_on_quantum (Quantum q)
/* Pushes -pi/q onto the estack, where q >= 2. */
{  push_minus_recip_of_quantum (q);
   push_quantum (PI_TAG);
   push_quantum (MULTIPLY_TAG);
}
```

push_negate

Declaration: void **push_negate** (EStackIndex *i*)

Category(ies): EStack Arithmetic

Description: Pushes onto the estack the internally-simplified negative of the expression indexed by *i*.

Inputs: *i* — Index of the top tag of an internally-simplified algebraic expression, algebraic comparison, or aggregate thereof.

Outputs: None

Assumptions: None

Side Effects: None

Availability: On AMS 2.02 and higher.

TI-89 / TI-92 Plus Differences: None

See Also: **negate_top**

Example:

```
Boolean is_symmetric (EStackIndex i, EStackIndex var)
/* Returns TRUE if it can determine that the expression indexed by i is symmetric
   with respect to the variable indexed by var. Otherwise returns FALSE.
*/
{  Access_AMS_Global_Variables;
   EStackIndex old_top = top_estack;
   push_negate (var);
   push_substitute_simplify (i, var, top_estack);
   subtract_from_top (i);
   if (is0 (top_estack))
      {  top_estack = old_top;
         return TRUE;
      }
   top_estack = old_top;
   return FALSE;
}
```

push_negate_quantum_as_negint

Declaration: void **push_negate_quantum_as_negint** (Quantum *q*)

Category(ies): EStack Arithmetic

Description: Pushes nonzero *q* onto the estack as a tagged negative big integer.

Inputs: *q* — Quantum.

Outputs: None

Assumptions: *q* is nonzero.

Side Effects: None

Availability: On AMS 2.00 and higher.

TI-89 / TI-92 Plus Differences: None

See Also: **push_quantum_as_nonnegative_int, push_reciprocal_of_quantum, push_minus_recip_of_quantum, push_quantum_pair_as_pos_frac**

Example:

```
void push_quadratic_discriminant (EStackIndex a, EStackIndex b, EStackIndex c)
/* Pushes onto the estack  b^2 - 4 a c. */
{  Access_AMS_Global_Variables;
   push_negate_quantum_as_negint (4u);
   times_top (a);
   times_top (c);
   a = top_estack;

   push_square (b);
   replace_top2_with_sum (a);
}
```

push_pi

Declaration: void **push_pi** (void)

Category(ies): EStack Arithmetic

Description: Pushes a tagged float 3.14159 . . . if IS_ARITH_APPROX. Otherwise pushes a PI_TAG.

Inputs: None

Outputs: None

Assumptions: None

Side Effects: None

Availability: On AMS 2.02 and higher.

TI-89 / TI-92 Plus Differences: None

See Also: **push_pi_on_quantum**

Example:

```
push_pi ();
push_cos (top_estack);  /* Pushes IS_ARITH_APPROX ? –1.0 : -1 */
```

push_pi_on_quantum

Declaration: void **push_pi_on_quantum** (Quantum *q*)

Category(ies): EStack Arithmetic

Description: Pushes π/*q* onto the estack, with *q* >= 2. The result is a tagged float if IS_ARITH_APPROX. Otherwise the result is a symbolic expression.

Inputs: *q* — Quantum.

Outputs: None

Assumptions: *q* >= 2

Side Effects: None

Availability: On AMS 2.02 and higher.

TI-89 / TI-92 Plus Differences: None

See Also: **push_pi**

Example:

```
/*
** Title: push_radians
** Description: pushes the result of converting the specified value
**              (interpreted as radians) to the currently selected measure
** Input: estack index i of the radian value
** Output: pushes the value converted to the currently selected angle measure
*/
void push_radians (EStackIndex i)
{  Access_AMS_Global_Variables;
   if (! is_free_of_tag (i, IM_RE_TAG))
      ER_THROW (ER_DOMAIN);
   if (IS_RADIANS)
      push_expression (i);
   else
   {  push_pi_on_quantum (180);
      push_ratio (i, top_estack);
      delete_expression (next_expression_index (top_estack));
   }
}
```

push_product

Declaration: void **push_product** (EStackIndex *i*, EStackIndex *j*)

Category(ies): EStack Arithmetic

Description: Pushes onto the estack the internally-simplified product of the expressions indexed by *i* and *j*. If both operands are conforming matrices, it is the matrix product rather than an element-wise product.

Inputs: *i, j* — Indexes to the top tags of internally-simplified algebraic expressions, algebraic comparisons, or aggregates thereof.

Outputs: None

Assumptions: None

Side Effects: None

Availability: On AMS 2.02 and higher.

TI-89 / TI-92 Plus Differences: None

See Also: **times_top, replace_top2_with_prod, push_negate, negate_top**

Example:

```
void push_inner_prod (EStackIndex i, EStackIndex j)
/* Pushes the inner product of two equal-length lists.
   Unlike push_dotproduct() it does not apply conj to the second arg.
*/
{  Access_AMS_Global_Variables;
   EStackIndex k;
   --i;
   --j;
   push0 ();
   while (END_TAG != ESTACK (i))
   {  k = top_estack;
      push_product (i, j);
      replace_top2_with_sum (k);
      i = next_expression_index (i);
      j = next_expression_index (j);
   }
}
```

push_quantum_as_nonnegative_int

Declaration: void **push_quantum_as_nonnegative_int** (Quantum *q*)

Category(ies): EStack Arithmetic

Description: Pushes *q* onto the estack as a tagged non-negative big integer.

Inputs: *q* — Quantum.

Outputs: None

Assumptions: None

Side Effects: None

Availability: On AMS 2.02 and higher.

TI-89 / TI-92 Plus Differences: None

See Also: **push_negate_quantum_as_negint, push_reciprocal_of_quantum, push_minus_recip_of_quantum, push_quantum_pair_as_pos_frac**

Example:

```
void push_arclen (EStackIndex i, EStackIndex vi, EStackIndex j, EStackIndex k)
/* j and k index expressions, vi indexes a variable, and i indexes an expression
   simplified through variable vi.
   Pushes onto the estack the arc displacement of expression i with respect
   to vi going from j through k.
*/
{  Access_AMS_Global_Variables;
   EStackIndex m,
               old_top = top_estack;
   push_quantum_as_nonnegative_int (2u);
   m = top_estack;
   push_1st_derivative (i, vi);
   replace_top2_with_pow (m);
   add1_to_top ();
   i = top_estack;
   push_sqrt (i);
   delete_between (old_top, i);
   i = top_estack;
   push_def_int (i, vi, j, k);
   delete_between (old_top, i);
}
```

push_quantum_pair_as_pos_frac

Declaration: void **push_quantum_pair_as_pos_frac** (Quantum *num*, Quantum *den*)

Category(ies): EStack Arithmetic

Description: Pushes *num* / *den* onto the estack as a positive fraction.

Inputs: *num*, *den* — Quantums.

Outputs: None

Assumptions: *num* != 0; *den* > 1; gcd (*num*, *den*) = 1.

Side Effects: None

Availability: On AMS 2.02 and higher.

TI-89 / TI-92 Plus Differences: None

See Also: **push_quantum_as_nonnegative_int, push_reciprocal_of_quantum, push_minus_recip_of_quantum, push_negate_quantum_as_negint**

Example:

```
push_quantum_pair_as_pos_frac (2u, 3u);  /* Pushes tagged faction 2/3 */
```

push_ratio

Declaration: void **push_ratio** (EStackIndex *i*, EStackIndex *j*)

Category(ies): EStack Arithmetic

Description: Pushes onto the estack the internally-simplified ratio of expression *i* divided by expression *j*.

Inputs: *i, j* — Indexes to top tags of internally-simplified algebraic expressions, algebraic comparisons, or aggregates thereof.

Outputs: None

Assumptions: None

Side Effects: None

Availability: On AMS 2.02 and higher.

TI-89 / TI-92 Plus Differences: None

See Also: **divide_top, replace_top2_with_ratio, push_reciprocal, replace_top_with_reciprocal**

Example:

```
EStackIndex index_push_monic_or_prim_pair (EStackIndex i, EStackIndex j)
/* If i indexes a Float, pushes 1.0 then the ratio of expressions j and i.
   Otherwise pushes i/gcd(i,j) then j/gcd(i,j).
   In either case, returns the index of the deeper pushed value.
*/
{  Access_AMS_Global_Variables;
   EStackIndex k;
   if (FLOAT_TAG == ESTACK (i))
      {  push_expression (Float1Index);
         k = top_estack;
         push_ratio (j, i);
      }
   else
      {  EStackIndex old_top = top_estack;
         EStackDisplacement del =
            deleted_between (old_top, push_gcd_then_cofactors (j, i, &k));
         k -= del;
      }
   return k;
}
```

push_reciprocal

Declaration: void **push_reciprocal** (EStackIndex *i*)

Category(ies): EStack Arithmetic, Math

Description: Pushes the internally-simplified reciprocal of the expression indexed by *i* onto the estack.

Inputs: *i* — Index of the top tag of an internally-simplified algebraic expression, algebraic comparison, or aggregate thereof.

Outputs: None

Assumptions: None

Side Effects: None

Availability: On AMS 2.02 and higher.

TI-89 / TI-92 Plus Differences: None

See Also: **divide_top, replace_top2_with_ratio, push_ratio, replace_top_with_reciprocal**

Example:

```
void replace_top_with_reciprocal (void)
/* Replaces the top expression on the estack with its reciprocal. */
{  Access_AMS_Global_Variables;
   EStackIndex old_top = top_estack;
   push_reciprocal (old_top);
   delete_expression (old_top);
}
```

push_reciprocal_of_quantum

Declaration: void **push_reciprocal_of_quantum** (Quantum *q*)

Category(ies): EStack Arithmetic

Description: Pushes tagged fraction 1/*q* onto the estack.

Inputs: *q* — Quantum.

Outputs: None

Assumptions: *q* > 1.

Side Effects: None

Availability: On AMS 2.02 and higher.

TI-89 / TI-92 Plus Differences: None

See Also: **push_negate_quantum_as_negint, push_quantum_as_nonnegative_int, push_minus_recip_of_quantum, push_quantum_pair_as_pos_frac**

Example:

```
void push_percent (EStackIndex i)
/* i indexes an algebraic or aggregate expression.
   Pushes i divided by 100.
*/
{  if (is_units_term (i))
      ER_THROW (ER_INVALID_USE_OF_UNITS);
   push_reciprocal_of_quantum (100u);
   times_top (i);
}
```

push_sum

Declaration: void **push_sum** (EStackIndex *i*, EStackIndex *j*)

Category(ies): EStack Arithmetic

Description: Pushes onto the estack the internally-simplified sum of the expressions indexed by *i* and *j*. If one operand is a scalar and the other is a square matrix, the scalar is distributed only over the diagonal of the matrix.

Inputs: *i, j* — Indexes to the top tags of internally-simplified algebraic expressions, algebraic comparisons, or aggregates thereof.

Outputs: None

Assumptions: None

Side Effects: None

Availability: On AMS 2.02 and higher.

TI-89 / TI-92 Plus Differences: None

See Also: **replace_top2_with_sum, push_difference, subtract_from_top, replace_top2_with_difference, push_arg_plus_1, add1_to_top, subtract1_from_top, add_to_top, push_arg_minus_1**

Example:

```
void push_increment_degree (EStackIndex power, EStackIndex inc)
/* Pushes onto the estack factor_base (power) ^ (inc + factor_deg(power)) */
{  push_sum (inc, factor_exponent_index (power));
   raise_to_top (factor_base_index (power));
}
```

push0

Declaration: void **push0** (void)

Category(ies): EStack Arithmetic

Description: Pushes a tagged float 0.0 if IS_ARITH_APPROX. Otherwise pushes a tagged big-integer 0.

Inputs: None

Outputs: None

Assumptions: None

Side Effects: None

Availability: On AMS 2.00 and higher.

TI-89 / TI-92 Plus Differences: None

See Also: **push1, push_pi, push_pi_on_quantum**

Example:

```
void push_dense_poly_eval (EStackIndex i, EStackIndex j)
/* i indexes a list of polynomial coefficients in descending order of the
   exponents of a variable, and j indexes a value for that variable.
   Pushes the value of the polynomial onto the estack.
*/
{  if (LIST_TAG == ESTACK(i))
   {  --i;
      push0 ();
      while (END_TAG != ESTACK(i))
      {  times_top (j);
         add_to_top (i);
         i = next_expression_index (i);
      }
   }
   else if (is_constant (i))
   {   ER_THROW (ER_DOMAIN);
   }
   else
   {  push_expression (j);
      push_expression (i);
      push_quantum (DENSE_POLY_EVAL_TAG);
   }
}
```

push1

Declaration: void **push1** (void)

Category(ies): EStack Arithmetic

Description: Pushes a tagged float 1.0 if IS_ARITH_APPROX. Otherwise pushes a tagged big-integer 1.

Inputs: None

Outputs: None

Assumptions: None

Side Effects: None

Availability: On AMS 2.00 and higher.

TI-89 / TI-92 Plus Differences: None

See Also: **push0, push_pi, push_pi_on_quantum**

Example:

```
void push_identity_mat (EStackIndex dim_exp_idx)
/*  dim_exp_idx indexes an integer in the range 1 through 32767.
    Pushes that size identity matrix onto the estack, using tagged float
    rather than tagged integer elements if IS_ARITH_APPROX.
*/
{  long  dimension, i, j;
   dimension = GetValue (dim_exp_idx, 1, 32767);
   push_quantum (END_TAG);
   for (i = 0 ; i < dimension; i++)
   {  push_quantum (END_TAG);
      for (j = 0 ; j < dimension; j++)
         if (j == i)
            push1 ();
         else push0 ();
      push_quantum (LIST_TAG);
   }
   push_quantum (LIST_TAG);
   return;
}
```

replace_top_with_reciprocal

Declaration: void **replace_top_with_reciprocal** (void)

Category(ies): EStack Arithmetic, Math

Description: Replaces the top of the estack with its internally-simplified reciprocal.

Inputs: None

Outputs: None

Assumptions: The top expression on the estack is the top tag of an internally-simplified algebraic expression or comparison.

Side Effects: None

Availability: On AMS 2.02 and higher.

TI-89 / TI-92 Plus Differences: None

See Also: **divide_top, replace_top2_with_ratio, push_ratio, push_reciprocal**

Example:

```
void push_anti_deriv_powprod_to_frac (EStackIndex i, EStackIndex k)
/* k indexes an expression of the form (bz + c)^n, with b possibly 1,
   c possibly 0, and n possibly 1.
   i indexes an expression of the form (a * k)^p, with p fractional and
   a possibly 1.
   Pushes corresponding anti-derivative with respect to z onto estack.
*/
{  Access_AMS_Global_Variables;
   EStackIndex old_top = top_estack;
   push_product (POWER_EXPONENT_INDEX (i), factor_exponent_index (k));
   if (is_minus1 (top_estack))
      {  /* int((a(bz + c)^n)^(-1/n),z) -> (a(bz + c)^n)^-(1/n) (bz + c) ln(bz + c) */
         top_estack = old_top;
         push_ln (factor_base_index (k));
         times_top (factor_base_index (k));
         times_top (i);
      }
   else  {  /* int ((az^n)^p, z) -> (a(bz + c)^n)^p (bz + c)/(np + 1) */
            add1_to_top ();
            replace_top_with_reciprocal ();
            times_top (factor_base_index (k));
            times_top (i);
         }
} /* end push_anti_deriv_powprod_to_frac */
```

replace_top2_with_difference

Declaration: void **replace_top2_with_difference** (EStackIndex *i*)

Category(ies): EStack Arithmetic

Description: Replaces the top two expressions of the estack with the internally-simplified difference of expression *i* minus the top expression. If one operand is a scalar and the other is a square matrix, the scalar is distributed only over the diagonal of the matrix.

Inputs: *i* — Index to the top tag of the deeper of the top two expressions of the estack. These top two expressions are internally-simplified algebraic expressions, algebraic comparisons, or aggregates thereof.

Outputs: None

Assumptions: None

Side Effects: None

Availability: On AMS 2.02 and higher.

TI-89 / TI-92 Plus Differences: None

See Also: **replace_top2_with_sum, push_sum, subtract_from_top, replace_top2_with_difference, push_arg_plus_1, add1_to_top, subtract1_from_top, add_to_top, push_arg_minus_1**

Example:

```
push_quantum_as_nonnegative_int (3u);
coefficient = top_estack;
push_estack (8u);   /* Push variable x */
replace_top2_with_prod (coefficient);
minuend = top_estack;   /* 3 * x  */
push_estack (8u);   /* Push variable x */
replace_top2_with_difference (minuend);  /* top_estack -> 2 * x  */
```

replace_top2_with_prod

Declaration: void **replace_top2_with_prod** (EStackIndex *i*)

Category(ies): EStack Arithmetic

Description: Replaces the top two expressions of the estack with their internally-simplified product. If both operands are conforming matrices, it is the matrix product rather than an element-wise product.

Inputs: *i* — Index to the top tag of the deeper of the top two expressions of the estack. These top two expressions are internally-simplified algebraic expressions, algebraic comparisons, or aggregates thereof.

Outputs: None

Assumptions: None

Side Effects: None

Availability: On AMS 2.02 and higher.

TI-89 / TI-92 Plus Differences: None

See Also: **times_top, push_product, push_negate, negate_top**

Example:

```
Boolean is_odd_expression (EStackIndex i)
/* Returns TRUE if the expression indexed by i is recognized as odd. */
{  Access_AMS_Global_Variables;
   EStackIndex old_top;
   Boolean b;
   if (SIGN_TAG == ESTACK (i))
      return IS_DOMAIN_REAL && is_real (i - 1u);
   old_top = top_estack;
   push_arg_minus_1 (i);
   i = top_estack;
   push_reciprocal_of_quantum (2u);
   replace_top2_with_prod (i);
   b = integer_non_unknown (top_estack) > 0;
   top_estack = old_top;
   return b;
}
```

replace_top2_with_ratio

Declaration: void **replace_top2_with_ratio** (EStackIndex *i*)

Category(ies): EStack Arithmetic

Description: Replaces the top two expressions on the estack with the internally-simplified (expression *i*)/(top expression).

Inputs: *i* — Index to the top tag of the deeper of the top two expressions of the estack. These top two expressions are internally-simplified algebraic expressions, algebraic comparisons, or aggregates thereof.

Outputs: None

Assumptions: None

Side Effects: None

Availability: On AMS 2.02 and higher.

TI-89 / TI-92 Plus Differences: None

See Also: **divide_top, push_divide, push_reciprocal, replace_top_with_reciprocal**

Example:

```
void push_log_gen (EStackIndex i, EStackIndex j)
/* Pushes onto the estack the logarithm to the base indexed by j of
   expression indexed by i.
*/
{  Access_AMS_Global_Variables;
   if (j)
   {  push_ln (i);
      i = top_estack;
      push_ln (j);
      replace_top2_with_ratio (i);
   }
   else
      push_log10 (i);
}
```

replace_top2_with_sum

Declaration: void **replace_top2_with_sum** (EStackIndex *i*)

Category(ies): EStack Arithmetic

Description: Replaces the top two expressions of the estack with their internally-simplified sum. If one operand is a scalar and the other is a square matrix, the scalar is distributed only over the diagonal of the matrix.

Inputs: *i* — Index to the top tag of the deeper of the top two expressions of the estack. These top two expressions are internally-simplified algebraic expressions, algebraic comparisons, or aggregates thereof.

Outputs: None

Assumptions: None

Side Effects: None

Availability: On AMS 2.02 and higher.

TI-89 / TI-92 Plus Differences: None

See Also: **push_sum, push_difference, subtract_from_top, replace_top2_with_difference, push_arg_plus_1, add1_to_top, subtract1_from_top, add_to_top, push_arg_minus_1**

Example:

```
void push_quadratic_discriminant (EStackIndex a, EStackIndex b, EStackIndex c)
/* Pushes onto the estack  b^2 - 4 a c. */
{  Access_AMS_Global_Variables;
   EStackIndex old_top = top_estack;
   push_negate_quantum_as_negint (4u);
   times_top (a);
   times_top (c);
   a = top_estack;

   push_square (b);
   replace_top2_with_sum (a);
}
```

subtract_from_top

Declaration: void **subtract_from_top** (EStackIndex *i*)

Category(ies): EStack Arithmetic

Description: Replaces the top expression with the internally-simplified difference of the expression *i* minus the expression indexed by *i*. If one operand is a scalar and the other is a square matrix, the scalar is distributed only over the diagonal of the matrix.

Inputs: *i* — Index to the top tag of an internally-simplified algebraic expression, algebraic comparison, or aggregate thereof.

Outputs: None

Assumptions: *i* and **top_estack** point to the top tags of internally-simplified algebraic expressions, algebraic comparisons, or aggregates thereof.

Side Effects: None

Availability: On AMS 2.02 and higher.

TI-89 / TI-92 Plus Differences: None

See Also: **push_sum, replace_top2_with_sum, add_to_top, push_difference, replace_top2_with_difference, add1_to_top, subtract1_from_top, push_arg_plus_1, push_arg_minus_1**

Example:

```
void push_div_dif_1f (EStackIndex i, EStackIndex vi, EStackIndex j)
/* avgRC:  Pushes onto the estack the forward 1st difference of the expression
           indexed by i, with respect to the variable indexed by vi, using
           the expression indexed by j as the step size.
*/
{  Access_AMS_Global_Variables;
   EStackIndex old_top = top_estack;
   push_expression (vi);
   add_to_top (j);
   push_substitute_simplify (i, vi, top_estack);
   subtract_from_top (i);   /* Rely on algebra to map over aggregates */
   i = top_estack;
   push_ratio (i, j);
   delete_between (old_top, i);
}
```

subtract1_from_top

Declaration: void **subtract1_from_top** (void)

Category(ies): EStack Arithmetic

Description: Replaces the top expression of the estack with the internally-simplified difference of the top minus 1.0 if IS_ARITH_APPROX, and the top minus 1 otherwise. If the top of the estack is a square matrix, the one is subtracted only from the diagonal of the matrix.

Inputs: None

Outputs: None

Assumptions: The top of the estack is an internally-simplified algebraic expression, algebraic comparison, or aggregate thereof.

Side Effects: None

Availability: On AMS 2.02 and higher.

TI-89 / TI-92 Plus Differences: None

See Also: **push_sum, replace_top2_with_sum, add_to_top, push_difference, replace_top2_with_difference, add1_to_top, subtract_from_top, push_arg_plus_1, push_arg_minus_1**

Example:

```
push_Float (5.7);
subtract1_from_top ();  /* pushes 4.7 */
```

times_top

Declaration: void **times_top** (EStackIndex *i*)

Category(ies): EStack Arithmetic

Description: Replaces the top expression on the estack with the internally-simplified product of the top times the expression indexed by *i*. If both operands are conforming matrices, the matrix product rather than an element-wise product is computed.

Inputs: *i* — Index of the top tag of an internally-simplified algebraic expression, algebraic comparison, or aggregate thereof.

Outputs: None

Assumptions: *i* and **top_estack** index the top tags of an internally-simplified algebraic expressions, algebraic comparisons, or aggregates thereof.

Side Effects: None

Availability: On AMS 2.02 and higher.

TI-89 / TI-92 Plus Differences: None

See Also: **push_product, replace_top2_with_prod, push_negate, negate_top**

Example:

```
void push_r_cis (EStackIndex r, EStackIndex t)
/* r and t index real expressions.
   Pushes r cos(t) + #i r sin(t) onto the estack.
*/
{  Access_AMS_Global_Variables;
   EStackIndex k;
   push_cos (t);
   times_top (r);
   k = top_estack;
   push_sin (t);
   times_top (r);
   replace_top2_with_imre (k);
}
```

Appendix A: System Routines — EStack Utilities

See Also:

check_estack_size

Declaration: void **check_estack_size** (EStackDisplacement *d*)

Category(ies): EStack Utilities

Description: This function is used to insure that at least *d* unused Quantums are available on the estack. For example, all of the push_ . . . procedures directly or indirectly call **check_estack_size**.

Inputs: *d* — The desired minimum number of unused Quantums on the estack.

Outputs: None

Assumptions: None

Side Effects: Might use **HeapRealloc** to make *d* unused Quantums available on the estack. Throws an **ESTACK_OVERFLOW_ERROR** if there are less than *d* unused Quantums available on the estack and **HeapRealloc** is unable to make that amount available.

Availability: On AMS 1.05 and higher.

TI-89 / TI-92 Plus Differences: None

See Also: None

Example:

```
void push_between (IndexConstQuantum low, IndexConstQuantum high)
/* Pushes quantums indexed from low + 1 through high onto the estack. */
{  Access_AMS_Global_Variables;
   OS_CHECK;
   check_estack_size ((EStackDisplacement)(high - low));

   (void)memcpy (& ESTACK (top_estack + 1u), & ESTACK (low + 1u),
                        (high - low) * BYTES_PER_QUANTUM);
   top_estack += high - low;
}
```

delete_between

Declaration: void **delete_between** (EStackIndex *low*, EStackIndex *high*)

Category(ies): EStack Utilities

Description: Deletes the quantums occupying from index *low* + 1 through *high* by copying down the quantums from *high* + 1 through global **top_estack**, which is then reduced by the number of deleted quantums.

Inputs: *low, high* — Indexes into the estack.

Outputs: None

Assumptions: **bottom_estack** <= *low* and *low* <= *high* and *high* <= **top_estack**.

Side Effects: **top_estack** is reduced by the number of deleted quantums.

Availability: On AMS 1.05 and higher.

TI-89 / TI-92 Plus Differences: None

See Also: **deleted_between, delete_expression, deleted_expression, move_between_to_top, moved_between_to_top**

Example:

```
void replace_top2_with_sum (EStackIndex i)
/* i indexes next-to-top expression.
   Replaces the top two expressions with their sum.
*/
{  Access_AMS_Global_Variables;
   EStackIndex j = top_estack;
   push_sum (i, j);
   delete_between (next_expression_index (i), j);
}
```

delete_expression

Declaration: void **delete_expression** (EStackIndex *i*)

Category(ies): EStack Utilities

Description: Deletes the estack expression indexed by *i*. Global **top_estack** is reduced by the number of deleted quantums.

Inputs: *i* — Index of the top (highest address) tag of an estack expression.

Outputs: None

Assumptions: None

Side Effects: **top_estack** is reduced by the number of deleted quantums.

Availability: On AMS 1.05 and higher.

TI-89 / TI-92 Plus Differences: None

See Also: **deleted_between, delete_between, deleted_expression, move_between_to_top, moved_between_to_top**

Example:

```
void replace_top_with_reciprocal (void)
/* Replaces the top expression on the estack with its reciprocal. */
{  Access_AMS_Global_Variables;
   EStackIndex old_top = top_estack;
   push_reciprocal (old_top);     /* find reciprocal of top expression */
   delete_expression (old_top);   /* remove original expression */
}
```

deleted_between

Declaration: EStackDisplacement **deleted_between** (EStackIndex *low*, EStackIndex *high*)

Category(ies): EStack Utilities

Description: Deletes the quantums occupying from index *low* + 1 through *high* by copying down the quantums from *high* + 1 through global **top_estack**, which is then reduced by the number of deleted quantums. Returns the number of deleted quantums.

Inputs: *low, high* — Indexes into the estack.

Outputs: Returns the number of deleted quantums.

Assumptions: **bottom_estack** <= *low* and *low* <= *high* and *high* <= **top_estack**.

Side Effects: **top_estack** is reduced by the number of deleted quantums.

Availability: On AMS 1.05 and higher.

TI-89 / TI-92 Plus Differences: None

See Also: **delete_between, delete_expression, deleted_expression, move_between_to_top, moved_between_to_top**

Example:

```
EStackDisplacement moved_between_to_top (EStackIndex i, EStackIndex j )
/* Copies elements from i + 1 through j to top of estack.
   Then deletes elements from i + 1 through j.
   Then returns the number of quantums deleted.
*/
{  push_between (i, j);
   return deleted_between (i, j);
}
```

deleted_expression

Declaration: EStackDisplacement **deleted_expression** (EStackIndex *i*)

Category(ies): EStack Utilities

Description: Deletes the estack expression indexed by *i*. Global **top_estack** is reduced by the number of deleted quantums. Returns the number of deleted quantums.

Inputs: *i* — Index of the top (highest address) tag of an estack expression.

Outputs: Number of deleted quantums.

Assumptions: None

Side Effects: **top_estack** is reduced by the number of deleted quantums.

Availability: On AMS 1.05 and higher.

TI-89 / TI-92 Plus Differences: None

See Also: **deleted_between, delete_between, delete_expression, move_between_to_top, moved_between_to_top**

Example:

```
EStackIndex push_cofactors (EStackIndex i, EStackIndex j)
/* i and j index internally-simplified algebraic expressions.
   Let gcd denote their greatest common divisor.
   Pushes onto the estack (expression j)/gcd then (expression i)/gcd,
   then returns the index of the deeper pushed cofactor.
*/
{  i = push_gcd_then_cofactors (i, j, &j);
   return j – deleted_expression (i);
}
```

estack_to_short

Declaration: short **estack_to_short** (EStackIndex *i*, signed short * *result*)

Category(ies): EStack Utilities, EStack Arithmetic

Description: Converts a tagged whole number to a C signed short.

Inputs:
i — Index of the tag of a tagged integer or tagged float whole number.

result — Points to a signed short for storing the short result.

Outputs: If successful, returns 1 after storing the signed short value in *result*. If the argument is a valid type having magnitude too large for a signed short, returns 0 after storing the correctly signed -32768 or 32767 in *result*. If the argument is invalid, returns -1 after storing 0 in *result*.

Assumptions: None

Side Effects: None

Availability: On AMS 1.05 and higher.

TI-89 / TI-92 Plus Differences: None

See Also: **estack_to_ushort, estack_to_float, estack_number_to_Float**

Example:

```
push_quantum_as_nonnegative_int (5u);
estack_to_short(top_estack, &ans);    /* Stores 5 in ans then returns 1 */
push_ushort_to_integer (65535);
estack_to_short(top_estack, &ans);    /* Stores 32767 in ans then returns 0 */
estack_to_short(FloatPiIndex, &ans);  /* Stores 0 in ans then returns –1 */
```

estack_to_ushort

Declaration: short **estack_to_ushort** (EStackIndex *i*, unsigned short * *result*)

Category(ies): EStack Utilities, EStack Arithmetic

Description: Converts a tagged whole number to a C unsigned short.

Inputs:
i — Index of the tag of a tagged integer or tagged float whole number.

result — Points to an unsigned short for return value.

Outputs: If successful, returns 1 and unsigned short via *result*. If the argument is a valid type, but too large for unsigned short, returns 0 and stores 65535 in *result*. If the argument is invalid, returns -1 and stores 0 in *result*.

Assumptions: None

Side Effects: None

Availability: On AMS 1.05 and higher.

TI-89 / TI-92 Plus Differences: None

See Also: **estack_to_short**

Example:

```
void push_char (EStackIndex i)
/* i indexes a tagged whole number from 0 through 255 or an aggregate thereof.
   Pushes the corresponding string or aggregate thereof, except char(0) -> "".
*/
{   unsigned short c;
    if (IS_NUMBER_TAG(ESTACK(i)) && is_whole_number(i) &&
        1 == estack_to_ushort(i, &c) && c < 256 )
    {   push_quantum (0);
        if (c)
            push_quantum ((Quantum) c);
        push_quantum (0u);
        push_quantum (STR_DATA_TAG);
    }
    else if (LIST_TAG == ESTACK(i))
    {   map_tail (push_char, i-1);
        push_quantum (LIST_TAG);
    }
    else ER_THROW (ER_DOMAIN);
}
```

GetValue

Declaration: long **GetValue** (EStackIndex *indx*, long *rLow*, long *rHigh*)

Category(ies): EStack Utilities, EStack Arithmetic

Description: Get a value from the estack in the range *rLow* to *rHigh*. Throw an ER_DOMAIN error if not in range. Throw an ER_DATATYPE error if the estack does not contain an integer or a number that can be converted to an integer.

Inputs:
indx — Indexes the location of the estack to retrieve the value from.
rLow — The low range of accepted values.
rHigh — The high range of accepted values.

Outputs: Value in the range *rLow* to *rHigh*.

Assumptions: None

Side Effects: May throw an ER_DOMAIN or ER_DATATYPE error.

Availability: On AMS 1.05 and higher.

TI-89 / TI-92 Plus Differences: None

See Also: **estack_to_short, estack_to_ushort**

Example: This example is used by the graphing commands in TI-BASIC to get the user supplied screen attribute value. It returns the attribute value after mapping it from a user supplied integer in the range -1 to +1.

```
/* Get a valid attribute value (0,1,-1) and return its corresponding
   screen attribute (A_REVERSE, A_NORMAL, A_XOR).
*/
short GetAttr( EStackIndex i )
{  Access_AMS_Global_Variables;
   SWORD RetInt;

   RetInt = GetValue( i, -1, 1 );
   if (RetInt == -1)
      return A_XOR;
   else if (RetInt == 0)
      return A_REVERSE;
   else
      return A_NORMAL;
}
```

move_between_to_top

Declaration: void **move_between_to_top** (EStackIndex *i*, EStackIndex *j*)

Category(ies): EStack Utilities

Description: Copies elements from (*i* + 1) through *j* to top of estack if space is available or can be made available. Elements (*i* + 1) through *j* are then deleted.

Inputs: *i*, *j* — Indexes into the estack.

Outputs: None

Assumptions: **bottom_estack** <= *i* and *i* <= *j* and *j* <= **top_estack**.

Side Effects: The estack temporarily grows even though there is no net change in **top_estack**. Consequently, an ESTACK_OVERFLOW_ERROR may be thrown or heap compression may have occurred.

Availability: On AMS 1.05 and higher.

TI-89 / TI-92 Plus Differences: None

See Also: **moved_between_to_top**

Example:

```
old_top = top_estack;
push_Float (3.7);
f = top_estack;
push_quantum_as_nonnegative_int (7u);
move_between_to_top (old_top, f);
   /* Now the float 3.7 is on top of the integer 7. */
```

moved_between_to_top

Declaration: EStackDisplacement **moved_between_to_top** (EStackIndex *i*, EStackIndex *j*)

Category(ies): EStack Utilities

Description: Copies elements from (*i* + 1) through *j* to top of estack if space is available or can be made available, then returns the number of elements copied after deleting elements (*i* + 1) through *j*.

Inputs: *i, j* — Indexes into the estack.

Outputs: Returns the number of quantums deleted.

Assumptions: **bottom_estack** <= *i* and *i* <= *j* and *j* <= **top_estack**.

Side Effects: The estack temporarily grows even though there is no net change in **top_estack**. Consequently, an ESTACK_OVERFLOW_ERROR may be thrown or heap compression may have occurred.

Availability: On AMS 1.05 and higher.

TI-89 / TI-92 Plus Differences: None

See Also: **move_between_to_top**

Example:

```
old_top = top_estack;
push_Float (3.7);
f = top_estack;
push_quantum_as_nonnegative_int (7u);
foo = top_estack;
push_quantum (8u);  /* Push variable x */
delta = moved_between_to_top (old_top, f);  /* Now f is on top */
foo -= delta;    /* Adjust index of foo */
x = top_estack – delta;  /* Adjust index of variable x */
```

next_expression_index

Declaration: EStackIndex **next_expression_index** (EStackIndex *i*)

Category(ies): EStack Utilities

Description: Returns the index of the next expression deeper than the one indexed by *i*.

Inputs: *i* — Index of the top (highest address) tag of an expression.

Outputs: Index of the quantum just below (lower address) the entire expression indexed by *i*.

Assumptions: None

Side Effects: Throws an ILLEGAL_TAG_ERROR, if *i* or one of its subexpressions does not contain a valid tag. This indicates a malformed estack or an *i* that is pointing to an END_TAG, into the middle of a number, or into the middle of a variable name, for example.

Availability: All versions of the TI-89 / TI-92 Plus.

TI-89 / TI-92 Plus Differences: None

See Also: None

Example:

```
/*******************************************************/
/* Title: remaining_element_count                      */
/* Description: Get number of expressions in tail.     */
/* Input: estack index i                               */
/*    i indexes a tail of expressions terminated       */
/*    by END_TAG.                                      */
/* Output: unsigned number of expressions in the tail  */
/*******************************************************/
ELEMENT_COUNT remaining_element_count (EStackIndex i) {
   ELEMENT_COUNT ans = 0u;
   while (ESTACK (i) != END_TAG) {
      ++ans;
      i = next_expression_index (i);
   }
   return (ans);
}
```

push_between

Declaration: void **push_between** (IndexConstQuantum *low*, IndexConstQuantum *high*)

Category(ies): EStack Utilities

Description: Pushes quantums indexed from (*low* + 1) through *high* onto the estack if space is available or can be made available.

Inputs: *low* — Index one quantum below the lowest quantum of the data.

high — Indexes the top tag of the data.

Outputs: None

Assumptions: *low* ≤ *high* — Both *low* and *high* point to the estack or the same locked block.

Side Effects: Throws ESTACK_OVERFLOW_ERROR if there is not enough space left on the estack. May cause heap compression. Throws ER_BREAK error if the ON key has been pressed to interrupt execution.

Availability: On AMS 1.05 and higher.

TI-89 / TI-92 Plus Differences: None

See Also: **push_expression, push_quantum**

Example:

```
void move_between_to_top (EStackIndex i, EStackIndex j)
/* Copies elements from i + 1 through j to top of estack,
   then deletes elements from i + 1 through j.
*/
{  push_between (i, j);
   delete_between (i, j);
}
```

push_expression

Declaration: void **push_expression** (EStackIndex *i*)

Category(ies): EStack Utilities

Description: Pushes onto the estack a copy of the expression indexed by *i*.

Inputs: *i* — Indexes the top tag of an expression.

Outputs: None

Assumptions: *i* points to the estack or some other locked block.

Side Effects: May cause heap compression or throw an error.

Availability: On AMS 2.00 and higher.

TI-89 / TI-92 Plus Differences: None

See Also: **push_between, push_quantum**

Example:

```
void push_var_kern_tail (EStackIndex i)
/* i indexes an expression.  Pushes onto the estack a tail of all its variables,
   with the most main variable deepest.
*/
{  EStackIndex vi = main_gen_var_index (i);
   push_quantum (END_TAG);
   while (vi)
      {  push_expression (vi);
         vi = next_var_or_kernel_index (i, vi);
      }
}
```

push_Float_to_rat

Declaration: void **push_Float_to_rat** (EStackIndex *i*)

Category(ies): EStack Utilities, EStack Arithmetic

Description: Pushes UNDEFINED_TAG if the float is a NAN, MINUS_INFINITY_TAG if the float represents -∞, PLUS_INFINITY_TAG if the float represents +∞, or PLUS_OR_MINUS_INFINITY_TAG if the float represents +-∞. Otherwise pushes the equivalent tagged bignum, to within a relative error given by **RAtionalize_tol**.

Inputs: *i* — Index to the tag of a tagged float.

Outputs: None

Assumptions: *i* points to the estack or some other locked block.

Side Effects: May cause heap compression or throw an error.

Availability: On AMS 1.05 and higher.

TI-89 / TI-92 Plus Differences: None

See Also: **push_Float_to_nonneg_int, did_push_cnvrt_Float_to_integer**

Example:

```
void push_round_Float (EStackIndex i)
/* i indexes a Float. Pushes onto the estack the float obtained by rationalizing
   then converting back to Float.
*/
{  Access_AMS_Global_Variables;
   EStackIndex old_top = top_estack;
   Float old_RAtionalize_tol = RAtionalize_tol;

   RAtionalize_tol = CUBE_ROOT_FLOAT_EPSILON;

   push_Float_to_rat (i);
   i = top_estack;

   RAtionalize_tol = old_RAtionalize_tol;

   PUSH_NUMBER_TO_FLOAT (i);
   delete_between (old_top, i);
}
```

push_long_to_integer

Declaration: void **push_long_to_integer** (long *x*)

Category(ies): EStack Utilities, EStack Arithmetic

Description: Pushes a tagged bignum integer whose value is the signed value of *x*.

Inputs: *x* — Long integer.

Outputs: None

Assumptions: None

Side Effects: May cause heap compression or throw an error.

Availability: On AMS 2.00 and higher.

TI-89 / TI-92 Plus Differences: None

See Also: **push_quantum_as_nonnegative_int, push_ulong_to_integer, push_ushort_to_integer**

Example:

```
push_long_to_integer ((long)(-2u));  /* Pushes a tagged integer –2 */
```

push_quantum

Declaration: void **push_quantum** (Quantum *q*)

Category(ies): EStack Utilities, Token Operations

Description: Pushes quantum *q* onto the estack if space is available or can be made available.

Inputs: *q* — Quantum.

Outputs: None

Assumptions: None

Side Effects: Throws ESTACK_OVERFLOW_ERROR if there is not enough space left on the estack. May cause heap compression.

Availability: On AMS 1.05 and higher.

TI-89 / TI-92 Plus Differences: None

See Also: **push_expression, push_between**

Example:

```
push_quantum(END_TAG);  /* Push a 10-element list of zeros on the estack */
for (i = 0 ; i < 10; i++)
   push0 ();
push_quantum (LIST_TAG);
```

push_ulong_to_integer

Declaration: void **push_ulong_to_integer** (unsigned long *x*)

Category(ies): EStack Utilities, EStack Arithmetic

Description: Pushes a non-negative tagged bignum integer whose value is *x*.

Inputs: *x* — Unsigned long integer.

Outputs: None

Assumptions: None

Side Effects: May cause heap compression or throw an error.

Availability: On AMS 2.00 and higher.

TI-89 / TI-92 Plus Differences: None

See Also: **push_quantum_as_nonnegative_int, push_long_to_integer, push_ushort_to_integer**

Example:

```
push_ulong_to_integer(123456789u);   /* Pushes the tagged integer 123456789 */
```

push_ushort_to_integer

Declaration: void **push_ushort_to_integer** (unsigned short *tq*)

Category(ies): EStack Utilities, EStack Arithmetic

Description: Pushes *tq* onto the estack as a tagged non-negative bignum integer.

Inputs: *tq* — Unsigned short integer.

Outputs: None

Assumptions: None

Side Effects: May cause heap compression or throw an error.

Availability: On AMS 2.00 and higher.

TI-89 / TI-92 Plus Differences: None

See Also: **push_quantum_as_nonnegative_int, push_long_to_integer, push_ulong_to_integer**

Example:

```
push_ushort_to_integer (655535u);  /* Pushes the tagged integer 655535 */
```

reset_estack_size

Declaration: void **reset_estack_size** (EStackDisplacement *size*)

Category(ies): EStack Utilities

Description: Changes the expression stack to the specified size.

Inputs: *size* — Amount of space to give the expression stack.

Outputs: None

Assumptions: If *size* is smaller than the current amount of used space, then the estack is set to the current amount of used space instead of *size*.

Side Effects: May throw a memory error if *size* is more space than can be allocated.

Availability: On AMS 1.05 and higher.

TI-89 / TI-92 Plus Differences: None

See Also: **bottom_estack, top_estack, estack_max_index**

Example:

```
reset_estack_size (INITIAL_ESTACK_MAX_COUNT);
```

Resets the expression stack to its default size. This is the most common use of **reset_estack_size**.

Appendix A: System Routines — Expression Evaluation / Algebraic Simplification

See Also:

ForceFloat

Declaration: BCD16 **ForceFloat** (EStackIndex *i*)

Category(ies): Expression Evaluation, Floating Point Operations

Description: If the estack expression pointed to by *i* is a floating-point value, it is returned. Otherwise **NG_approxESI** is used to try to force it to a floating-point value. Either way, the result returned is rounded to 14 places before it is returned.

Inputs: *i* — estack pointer of expression to convert to float.

Outputs: BCD16 (double) value of estack expression.

Assumptions: None

Side Effects: None

Availability: On AMS 2.00 and higher.

TI-89 / TI-92 Plus Differences: None

See Also: **NG_approxESI**

Example:

```
Access_AMS_Global_Variables;
EStackIndex origEStack;
char Buf[128];
BCD16 floatVal;

WinClr( &appW );
strcpy( (char *) Buf, "100/201" );
origEStack = top_estack;
TRY
   push_quantum( END_OF_SEGMENT_TAG );
   push_parse_text( (BYTE *) Buf );
   floatVal = ForceFloat( top_estack );
   sprintf( Buf, "%f", floatVal );
   WinStr( &appW, Buf );
   GKeyIn( NULL, 0 );
ONERR
   top_estack = origEStack;
   PASS;
ENDTRY
top_estack = origEStack;
```

NG_approxESI

Declaration: void **NG_approxESI** (EStackIndex *esi*)

Category(ies): Expression Evaluation / Algebraic Simplification

Description: Forces the simplification of an expression in APPROX mode.

Inputs: *esi* — EStackIndex of the expression to be simplified.

Outputs: Pushes onto the estack the external tokenized result of fully simplifying the input expression in APPROX mode.

Assumptions: None

Side Effects: May cause estack expansion, heap compression, or throw errors associated with APPROX simplification of the input expression.

Availability: All versions of the TI-89 / TI-92 Plus.

TI-89 / TI-92 Plus Differences: None

See Also: **NG_rationalESI, NG_execute**

Example:

If j indexes the bolded tag in the following external tokenized form of the expression 3/2 + √(-9)
3 1 NONNEGATIVE_INTEGER_TAG 2 1 NONNEGATIVE_INTEGER_TAG DIVIDE_TAG 9 1 NONNEGATIVE_INTEGER_TAG CHS_TAG SQRT_TAG **ADD_TAG**

then

```
NG_approxESI (j);
```

pushes the external tokenized form of 1.5 + 3.*i* onto the estack such that **top_estack** points to the bolded tag.
0x40 0x00 0x15 0x00 0x00 0x00 0x00 0x00 0x00 FLOAT_TAG 0x40 0x00 0x30 0x00 0x00 0x00 0x00 0x00 0x00 FLOAT_TAG I_TAG MULTIPLY_TAG **ADD_TAG**

NG_execute

Declaration: void **NG_execute** (HANDLE *hExpr*, BOOL *bApprox*)

Category(ies): Expression Evaluation / Algebraic Simplification

Description: Executes one or more expressions or statements.

Inputs:

hExpr — HANDLE of memory containing expression(s) or statement(s) to execute.

bApprox — NG_APPROXIMATE approximates the result.

NG_DONT_APPROXIMATE uses the current mode setting.

Outputs: Depending upon the input, may push evaluation result on estack in external tokenized form. The evaluation of an algebraic expression or a function leaves a result on the estack. The execution of commands or programs does not leave a result on the estack.

Assumptions: None

Side Effects: Clears error context, resets control flags, may cause estack expansion, heap compression, or throw errors.

Availability: All versions of the TI-89 / TI-92 Plus.

TI-89 / TI-92 Plus Differences: None

See Also: None

Example:

If h is the HANDLE of memory containing a tokenized form of the statement

Define f(x) = sin(x)

then

```
NG_execute (h, NG_DONT_APPROXIMATE);
```

would define the function f in the current symbol table and would not leave any result on the estack.

NG_rationalESI

Declaration: void **NG_rationalESI** (EStackIndex *esi*)

Category(ies): Expression Evaluation / Algebraic Simplification

Description: Forces the simplification of an expression in EXACT mode.

Inputs: *esi* — EStackIndex of the expression to be simplified.

Outputs: Pushes onto the estack the external tokenized result of fully simplifying the input expression in EXACT mode.

Assumptions: None

Side Effects: May cause estack expansion, heap compression, or throw errors associated with EXACT simplification of the input expression.

Availability: All versions of the TI-89 / TI-92 Plus.

TI-89 / TI-92 Plus Differences: None

See Also: **NG_approxESI, NG_execute**

Example:

If j indexes the bolded tag in the following external tokenized form of the expression 1.5 + √(-9)
0x40 0x00 0x15 0x00 0x00 0x00 0x00 0x00 0x00 FLOAT_TAG 9 1 NONNEGATIVE_INTEGER_TAG CHS_TAG SQRT_TAG **ADD_TAG**

then

```
NG_rationalESI (j);
```

pushes the external tokenized form of 3/2 + 3*i* onto the estack such that **top_estack** points to the bolded tag.
2 1 3 1 POSITIVE_FRACTION_TAG 3 1 NONNEGATIVE_INTEGER_TAG I_TAG MULTIPLY_TAG **ADD_TAG**

push_approx

Declaration: void **push_approx** (EStackIndex *i*)

Category(ies): Expression Evaluation / Algebraic Simplification

Description: Executes SET_ARITH_APPROX, then pushes a corresponding simplified version of the expression onto the estack. The arithmetic mode is restored to what it was upon entry to this subroutine.

Inputs: *i* — Index to an expression.

Outputs: None

Assumptions: *i* points to the top tag of an expression that does not have to be already internally simplified, but can be.

Side Effects: None

Availability: On AMS 2.01 and higher.

TI-89 / TI-92 Plus Differences: None

See Also: **estack_number_to_Float**

Example:

```
push_quantum_as_nonnegative_int (2u);
exponent = top_estack;
push_quantum (8u);                        /* Push variable x */
replace_top2_with_pow (exponent);
power = top_estack;                       /* x^2 */
push_quantum (PI_TAG);
replace_top2_with_prod (power);
product = top_estack;                     /* pi * x^2 */
push_quantum_as_nonnegative_int (5u);     /* pi * x^2 + 5 */
push_approx (top_estack);                 /* 3.1415926536898 * x^2 + 5.0 */
   /* Note: Whole-number exponents such as 2 are not approximated. */
```

push_equals

Declaration: void **push_equals** (EStackIndex *i*, EStackIndex *j*)

Category(ies): Expression Evaluation / Algebraic Simplification

Description: Pushes onto the estack the internally-simplified equivalent of the equality (expression *i*) = (expression *j*).

To accommodate stepping through equation solving, the simplification is merely a comparison of the two sides for the default IS_AUTO_SOLVE_OFF.

If the result is neither a TRUE_TAG nor a FALSE_TAG, the result could be true for some values of the variables therein but false for other values. Alternatively the simplification might not be powerful enough to determine that the result is always true or always false.

Inputs: *i*, *j* — Indices of the top tags of an internally-simplified expressions, strings, or aggregates thereof.

Outputs: None

Assumptions: None

Side Effects: None

Availability: On AMS 2.02 and higher.

TI-89 / TI-92 Plus Differences: None

See Also: **push_not_equals, push_greater_than, push_less_than, push_greater_than_or_equals, push_less_than_or_equals,**

Example:

```
push_quantum_as_nonnegative_int (3u);
right_side = top_estack;
push_quantum (8u);                          /* Push variable x */
push_equals (top_estack, right_side);       /* push x = 3 */
```

push_greater_than

Declaration: void **push_greater_than** (EStackIndex *i*, EStackIndex *j*)

Category(ies): Expression Evaluation / Algebraic Simplification

Description: Pushes onto the estack the internally-simplified equivalent of the inequality (expression *i*) > (expression *j*).

To accommodate stepping through inequality solving, the simplification is merely a comparison of the two sides for the default IS_AUTO_SOLVE_OFF.

If the result is neither a TRUE_TAG nor a FALSE_TAG, the result could be true for some values of the variables therein but false for other values. Alternatively the simplification might not be powerful enough to determine that the result is always true or always false.

Inputs: *i*, *j* — Indices of the top tags of internally-simplified strings, algebraic expressions, or aggregates thereof.

Outputs: None

Assumptions: None

Side Effects: None

Availability: On AMS 2.02 and higher.

TI-89 / TI-92 Plus Differences: None

See Also: **push_not_equals, push_equals, push_less_than, push_greater_than_or_equals, push_less_than_or_equals**

Example:

```
push_Float (2.3);
j = top_estack;
push_Float (3.5);
push_greater_than (top_estack, j);   /* Pushes TRUE_TAG */
```

push_greater_than_or_equals

Declaration: void **push_greater_than_or_equals** (EStackIndex *i*, EStackIndex *j*)

Category(ies): Expression Evaluation / Algebraic Simplification

Description: Pushes onto the estack the internally-simplified equivalent of the inequality (expression *i*) ≥ (expression *j*).

To accommodate stepping through inequality solving, the simplification is merely a comparison of the two sides for the default IS_AUTO_SOLVE_OFF.

If the result is neither a TRUE_TAG nor a FALSE_TAG, the result could be true for some values of the variables therein but false for other values. Alternatively the simplification might not be powerful enough to determine that the result is always true or always false.

Inputs: *i*, *j* — Indices of the top tags of internally-simplified strings, algebraic expressions, or aggregates thereof.

Outputs: None

Assumptions: None

Side Effects: None

Availability: On AMS 2.02 and higher.

TI-89 / TI-92 Plus Differences: None

See Also: **push_not_equals, push_greater_than, push_less_than, push_equals, push_less_than_or_equals**

Example:

```
push_Float (2.3);
j = top_estack;
push_Float (3.5);
push_greater_than_or_equals (top_estack, j);   /* Pushes TRUE_TAG */
```

push_internal_simplify

Declaration: void **push_internal_simplify** (EStackIndex *i*)

Category(ies): Expression Evaluation / Algebraic Simplification

Description: The low-level entry point to the simplifier accepts all valid tokenized expressions and produces an internal tokenized expression.

Inputs: *i* — EStackIndex of tokenized expression.

Outputs: Pushes onto the expression stack the internal tokenized form of the result of evaluating / simplifying the input.

Assumptions: Input must be a logical, Boolean, or algebraic expression or relation. Use **push_simplify_statements** to process TI-BASIC commands.

Side Effects: May expand expression stack, cause heap compression, or throw an error.

Availability: On AMS 2.01 and higher.

TI-89 / TI-92 Plus Differences: None

See Also: **push_simplify_statements, push_simplify, replace_top_with_post_simplified**

Example:

If i is the EStackIndex of the external-tokenized form of the complex expression a + b ∗ *i*, which is A_VAR_TAG B_VAR_TAG I_TAG MULTIPLY_TAG ADD_TAG, then

```
push_internal_simplify (i);
```

pushes the internal-tokenized form of the expression, which is A_VAR_TAG B_VAR_TAG IM_RE_TAG.

push_less_than

Declaration: void **push_less_than** (EStackIndex *i*, EStackIndex *j*)

Category(ies): Expression Evaluation / Algebraic Simplification

Description: Pushes onto the estack the internally-simplified equivalent of the inequality (expression *i*) < (expression *j*).

To accommodate stepping through inequality solving, the simplification is merely a comparison of the two sides for the default IS_AUTO_SOLVE_OFF.

If the result is neither a TRUE_TAG nor a FALSE_TAG, the result could be true for some values of the variables therein but false for other values. Alternatively the simplification might not be powerful enough to determine that the result is always true or always false.

Inputs: *i*, *j* — Indices of the top tags of internally-simplified strings, algebraic expressions, or aggregates thereof.

Outputs: None

Assumptions: None

Side Effects: None

Availability: On AMS 2.02 and higher.

TI-89 / TI-92 Plus Differences: None

See Also: **push_not_equals, push_equals, push_greater_than, push_greater_than_or_equals, push_less_than_or_equals**

Example:

```
push_Float (2.3);
j = top_estack;
push_Float (3.5);
push_less_than (top_estack, j);   /* Pushes FALSE_TAG */
```

push_less_than_or_equals

Declaration: void **push_less_than_or_equals** (EStackIndex *i*, EStackIndex *j*)

Category(ies): Expression Evaluation / Algebraic Simplification

Description: Pushes onto the estack the internally-simplified equivalent of the inequality (expression *i*) ≤ (expression *j*).

Inputs: *i*, *j* — Indices of the top tags of internally-simplified strings, algebraic expressions, or aggregates thereof.

Outputs: None

Assumptions: None

Side Effects: None

Availability: On AMS 2.02 and higher.

TI-89 / TI-92 Plus Differences: None

See Also: **push_not_equals, push_equals, push_less_than, push_greater_than_or_equals, push_greater_than**

Example:

```
push_Float (2.3);
j = top_estack;
push_Float (3.5);
push_less_than_or_equals (top_estack, j);   /* Pushes FALSE_TAG */
```

push_not_equals

Declaration: void **push_not_equals** (EStackIndex *i*, EStackIndex *j*)

Category(ies): Expression Evaluation / Algebraic Simplification

Description: Pushes onto the estack the internally-simplified equivalent of the inequality (expression *i*) ≠ (expression *j*).

To accommodate stepping through inequality solving, the simplification is merely a comparison of the two sides for the default IS_AUTO_SOLVE_OFF.

If the result is neither a TRUE_TAG nor a FALSE_TAG, the result could be true for some values of the variables therein but false for other values. Alternatively the simplification might not be powerful enough to determine that the result is always true or always false.

Inputs: *i*, *j* — Indices of the top tags of an internally-simplified expressions, strings, or aggregates thereof.

Outputs: None

Assumptions: None

Side Effects: None

Availability: On AMS 2.02 and higher.

TI-89 / TI-92 Plus Differences: None

See Also: **push_equals, push_greater_than, push_less_than, push_greater_than_or_equals, push_less_than_or_equals**

Example:

```
push_quantum_as_nonnegative_int (3u);
right_side = top_estack;
push_quantum (8u);   /* Push variable x */
push_not_equals (top_estack, right_side);   /* Push x ≠ 3 */
```

push_simplify

Declaration: void **push_simplify** (EStackIndex *k*)

Category(ies): Expression Evaluation / Algebraic Simplification

Description: Pushes onto the estack in external Polish form a simplified version of the expression indexed by *k*.

Inputs: *k* — Indexes an expression that can be in either external Polish form or internally-simplified Polish form.

Outputs: None

Assumptions: None

Side Effects: None

Availability: On AMS 2.00 and higher.

TI-89 / TI-92 Plus Differences: None

See Also: **push_internal_simplify, replace_top_with_post_simplified**

Example:

```
push_Float (3.7);
push_Float (2.3);
/* Note: SUBTRACT_TAG does not occur in internally-simplified expressions */
push_quantum (SUBTRACT_TAG);
push_simplify (top_estack);   /* Pushes tagged float 1.4 onto the estack. */
```

push_simplify_statements

Declaration: void **push_simplify_statements** (EStackIndex *i*)

Category(ies): Expression Evaluation / Algebraic Simplification

Description: The top level entry point to the simplifier accepts all valid tokenized input and produces an external tokenized output.

Inputs: *i* — EStackIndex of tokenized input.

Outputs: Pushes onto the expression stack the external tokenized form of the result of evaluating / simplifying the input.

Assumptions: None

Side Effects: May expand expression stack, cause heap compression, or throw an error.

Availability: On AMS 2.00 and higher.

TI-89 / TI-92 Plus Differences: None

See Also: **push_simplify, push_internal_simplify**

Example:

```
void evaluate_string(unsigned char *i)
/* Given a text string, convert it to tokenized form, and then, evaluate/simplify it.

   input:  i – text string
   output: return one result (if any) on top of expression stack
*/
{  Access_AMS_Global_Variables;
   EStackIndex oldtop = top_estack;

   push_parse_text (i);                        /* convert string to tokenized form */
   i = top_estack;                             /* save top of tokenized form */
   push_simplify_statements (i);               /* evaluate/simplify */
   if (top_estack != old_top)                  /* if eval left a result */
      i = next_expression_index (top_estack);  /* point below result */
   delete_between (old_top, i);                /* clean up all but result */
}
```

replace_top_with_post_simplified

Declaration: void **replace_top_with_post_simplified** (EStackIndex *old_top*)

Category(ies): Expression Evaluation / Algebraic Simplification

Description: Replaces the internal-tokenized expression on top of the stack with its external-tokenized form.

Inputs: *old_top* — EStackIndex of position below the internal-tokenized expression on top of the expression stack.

Outputs: None

Assumptions: None

Side Effects: May expand expression stack, cause heap compression, or throw an error.

Availability: On AMS 2.03 and higher.

TI-89 / TI-92 Plus Differences: None

See Also: **push_simplify_statements, push_simplify, push_internal_simplify**

Example:

If the internal-tokenized form of the complex expression a + b ∗ *i*, which is A_VAR_TAG B_VAR_TAG IM_RE_TAG, is on top of the expression stack, then

```
replace_top_with_post_simplified (next_expression_index (top_estack));
```

replaces that expression with its external-tokenized form, which is A_VAR_TAG B_VAR_TAG I_TAG MULTIPLY_TAG ADD_TAG.

Appendix A: System Routines — Files

FAccess

Declaration: WORD **FAccess** (char * *fileName*, WORD *mode*, char * *typeName*)

Category(ies): Files

Description: Check to see if a file can be opened for a given mode without modifying the file. The file, if it exists, must have the same type as *typeName*.

Inputs:

fileName	—	String name of file to check.
mode	—	FM_READ_ACCESS or FM_WRITE_ACCESS.
typeName	—	1 . . . 4 character string of file type.

Outputs:

FS_OK	—	The file can be opened in the given mode.
FS_ERROR	—	The file cannot be opened for the specified mode (may be locked or is not a third-party data-type).
FS_NOT_FOUND	—	*fileName* does not exist.
FS_BAD_NAME	—	*fileName* is invalid.

Assumptions: The file, if it exists, must have the same type as *typeName.*

Side Effects: None

Availability: On AMS 2.00 and higher.

TI-89 / TI-92 Plus Differences: None

See Also: **FOpen, FCreate**

Example:

```
if (FS_ERROR == FAccess("ZTEMP", FM_WRITE_ACCESS, "TYPE")) {
   Disp("ERROR: Can not write to data file");
   return;
}
```

FClose

Declaration: WORD **FClose** (FILES * *fsPtr*)

Category(ies): Files

Description: Close a file. This is required for files opened in WRITE mode. For files in READ mode just updates the *fsPtr*->fileMode and *fsPtr*->fileStatus fields.

Inputs: *fsPtr* — Pointer to a FILES structure that was previously opened with **FOpen**.

Outputs: Return FS_OK if file successfully closed. There is extra overhead required for a FILE that is always kept around by the FILE system and so closing a FILE will only return an error if the file or the heap has been damaged.

Assumptions: None

Side Effects: None

Availability: On AMS 2.00 and higher.

TI-89 / TI-92 Plus Differences: None

See Also: **FAccess, FOpen, FStatus**

Example:

```
FILE f1;
if (FS_OK == FOpen("AF1", &f1, FM_WRITE, "APP1" )) {
   FWrite( "LINE 1\n", 7, &f1 );
   FWrite( "LINE 2\n", 7, &f1 );
   FClose( &f1 );
   WinStr( &appW, "AF1.APP1 created\n");
}  else
   WinStr( &appW, "AF1 Failed!\n");
```

FCreate

Declaration: WORD **FCreate** (char * *fileName*, char * *typeName*)

Category(ies): Files

Description: Create an empty file. Normally only needed if multiple files must be simultaneously opened in write mode. Return value same as **FOpen**. *typeName* is the same value normally passed to **FOpen**.

Inputs:

fileName — String pointer to name of file to create.

typeName — Must point to a string of, at most, four characters that describes the file type (FS_ERROR is returned if it does not).

Outputs: Same return value as **FOpen**.

Assumptions: Same as **FOpen**.

Side Effects: May cause heap compression or invalidate any current HSYMs.

Availability: On AMS 2.00 and higher.

TI-89 / TI-92 Plus Differences: None

See Also: **FClose, FOpen**

Example:

```
static char fType = "DAT";
FILES f1, f2;
if (FS_OK == FCreate("f1", fType) && FS_OK == FCreate("f2",fType)) {
   if (FS_OK == FOpen( "f1", &f1, FM_WRITE, fType )) {
      if (FS_OK == FOpen( "f2", &f2, FM_WRITE, fType )) {
         writeToFile( &f1 );
         readFromOtherFile();
         writeToFile( &f2 );
         writeToFile( &f1 );
         FClose( &f2 );
      }
      FClose( &f1 );
   }
}
```

FDelete

Declaration: WORD **FDelete** (const char * *fileName*)

Category(ies): Files

Description: Deletes a file.

Inputs:	*fileName*	—	String pointer to name of file to delete.
Outputs:	FS_OK	—	File deleted.
	FS_BAD_NAME	—	Bad filename.
	FS_ERROR	—	File not deleted because it is locked, in use, a folder, in Flash, or it does not exist.

Assumptions: None

Side Effects: May invalidate any current HSYMs.

Availability: On AMS 2.00 and higher.

TI‑89 / TI‑92 Plus Differences: None

See Also: None

Example:

```
/* Create a temporary file, use the file, then delete it. */
FILE f1;
if (FS_OK == FOpen("ZTEMPF", &f1, FM_WRITE, "APP1" )) {
   useTempFile( &f1 );
   FClose( &f1 );
   FDelete("ZTEMPF");
}
```

FEof

Declaration: BOOL **FEof** (FILES * *fsPtr*)

Category(ies): Files

Description: Returns TRUE if a FILE is at the end.

Inputs: *fsPtr* — Pointer to a FILES structure opened with **FOpen**.

Outputs: TRUE if a file is at the End of File mark. FALSE otherwise.

Assumptions: Note that this is not an error condition (like returned by **FStatus**) and can be changed not only by reading from a file but by changing the current file position (see example).

Side Effects: None

Availability: On AMS 2.00 and higher.

TI-89 / TI-92 Plus Differences: None

See Also: **FOpen, FSetPos, FGetPos**

Example:

```
FILE f1;
if (FS_OK == FOpen("APPDATA", &f1, FM_READ, "APP1" )) {
   while (!FEof(&f1)) {
      c = FGetC(&f1);
      WinChar( w, c );
   }
   FSetPos( &f1, 0 );  /* rewind file, FEof should now return FALSE */
   if (FEof(&f1))
      Disp("FILE must have been empty");
}
```

FFindFirst

Declaration: SYM_ENTRY * **FFindFirst** (WORD *Options*, char * *typeName*, char * *folderName*)

Category(ies): Files

Description: Find the first symbol of FILE type *typeName* and return a SYM_ENTRY pointer to it (NULL if none found). Use **FFindNext** to find subsequent entries, NULL returned after the last entry is found.

Inputs:

Options — FO_NONE — *folderName* is a string pointer to the folder to search.

FO_RECURSE — Search all folders (*folderName* is ignored).

typeName — Must point to a string of, at most, four characters that describes the file type.

Outputs: SYM_ENTRY pointer to first file with type matching *typeName.*

The *Name* field of the SYM_ENTRY contains the 1 . . . 8 character string of the symbol for the file being searched for. If using FO_RECURSE, then calling **SymFindFoldername** will return the folder of the symbol just found.

Assumptions: None

Side Effects: SYM_ENTRY pointers are only valid until heap compression is done or another symbol is added to the symbol table.

Availability: On AMS 2.00 and higher.

TI-89 / TI-92 Plus Differences: None

See Also: **FFindNext, SymFindFoldername**

Example:

```
SYM_ENTRY sePtr;
Disp ("ALL Files of type FTYPE2\n");
if (sePtr = FFindFirst(FO_RECURSE, FTYPE2, NULL)) {
   Disp(sePtr->Name);
   while (sePtr = FFindNext())
      Disp(sePtr->Name);
}
Disp("All Files of type FTYPE2 in folder: FOLD1\n");
if (sePtr = FFindFirst(FO_NONE, FTYPE2, "fold1")) {
   Disp(sePtr->Name);
   while (sePtr = FFindNext())
      Disp(sePtr->Name);
}
```

FFindNext

Declaration: SYM_ENTRY * **FFindNext** (void)

Category(ies): Files

Description: Find the next symbol as setup by a call to **FFindFirst** and return a SYM_ENTRY pointer to it (NULL if no more symbols found).

Inputs: None

Outputs: SYM_ENTRY pointer to file with type matching that as setup by a previous call to **SymFindFirst**. If using FO_RECURSE, then calling **SymFindFoldername** will return the folder of the symbol just found.

Assumptions: **FFindFirst** must have been called previously to find the first symbol.

Side Effects: SYM_ENTRY pointers are only valid until heap compression is done or another symbol is added to the symbol table.

Availability: On AMS 2.00 and higher.

TI‑89 / TI‑92 Plus Differences: None

See Also: **FFindFirst, SymFindFoldername**

Example: See **FFindFirst**.

FGetC

Declaration: WORD **FGetC** (FILES * *fsPtr*)

Category(ies): Files

Description: Read a BYTE from an open file (may be in either READ or WRITE mode) and return the BYTE or FS_EOF if the end of file was reached.

Inputs: *fsPtr* — Pointer to FILES structure previously opened with **FOpen**.

Outputs: Next BYTE in file or FS_EOF if reached end of file.

Assumptions: The FILES structure pointed to by *fsPtr* must have been opened with **FOpen**.

Side Effects: None

Availability: On AMS 2.00 and higher.

TI-89 / TI-92 Plus Differences: None

See Also: **FOpen, FRead, FSetPos**

Example:

```
FILE f1;
if (FS_OK == FOpen("APPDATA", &f1, FM_READ, "APP1" )) {
   while (!FEof(&f1)) {
      c = FGetC(&f1);
      WinChar( w, c );
   }
}
```

FGetPos

Declaration: FSWORD **FGetPos** (FILES * *fsPtr*)

Category(ies): Files

Description: Return the current file position (where the next read or write would occur) for a FILE.

Inputs: *fsPtr* — Pointer to FILES structure previously opened with **FOpen**.

Outputs: Current file position.

Assumptions: The FILES structure pointed to by *fsPtr* must have been previously opened with **FOpen**.

Side Effects: None

Availability: On AMS 2.00 and higher.

TI-89 / TI-92 Plus Differences: None

See Also: **FOpen, FSetPos, FEof**

Example:

```
FILES f1;
FSWORD fPos;
static char FTYPE2 = "DAT";

if (FS_OK == FOpen( "f3", &f1, FM_WRITE, FTYPE2 )) {
   FWrite( "abc", 3, &f1 );
   x = FGetPos( &f1 );  /* x should have value of 3 */
}
```

FGetSize

Declaration: FSWORD **FGetSize** (FILES * *fsPtr*)

Category(ies): Files

Description: Return the number of bytes currently stored in an opened FILE.

Inputs: *fsPtr* — Pointer to FILES structure previously opened with **FOpen**.

Outputs: Number of bytes in the opened file.

Assumptions: The FILES structure pointed to by *fsPtr* must have been previously opened with **FOpen** (either read or write mode).

Side Effects: None

Availability: On AMS 2.00 and higher.

TI-89 / TI-92 Plus Differences: None

See Also: **FOpen, FSetSize**

Example:

```
FILES f1;
FSWORD x;
if (FS_OK == FOpen("f1", &f1, FM_WRITE, "DAT"))  {
   FWrite( "START", 5, &f1 );
   x = FGetSize( &f1 );                        /* x == 5 */
   FWrite( "123", 3, &f1 );
   x = FGetSize( &f1 );                        /* x == 8 */
   FSetSize( &f1, 7 );                         /* truncate file */
   x = FGetSize( &f1 );                        /* x == 7 */
   FClose(&f1);
}
```

FOpen

Declaration: WORD **FOpen** (const char * *fileName*, FILES * *fsPtr*, WORD *mode*, char * *typeName*)

Category(ies): Files

Description: Open a file for a specific mode.

Inputs:

fileName — String pointer to name of file to open.

fsPtr — Pointer to a structure of type FILES.

mode —
FM_READ
File must exist and can only be read.
FM_WRITE
Open file for writing and erase its contents. Create file if it does not exist.
FM_APPEND
If file exists, open it for writing at the end of the file. If it does not exist, then create it and open it for writing.

typeName — Must point to a string of, at most, four characters that describes the file type (FS_ERROR is returned if it does not). If there is an existing file with the same name as *fileName* then the types must match.

Outputs:

FS_OK — The file was opened for the specified mode.

FS_ERROR — The file cannot be opened for the specified mode (may be locked or is not a third-party data-type).

FS_BAD_NAME — *fileName* is invalid.

FS_MEMORY — Not enough memory.

The return value is also stored in *fsPtr* -> fileStatus.
The FILES pointer may then be used in subsequent file routines.

Assumptions: Filenames are not tokenized variable names (as required by the symbol table code) but rather a string of characters. They must not be reserved names. If a filename does not have a folder name then it will be stored in the current folder. Internally, FILEs are stored as third-party data-types GEN_DATA_TAG). They will show up to the user in the VAR-LINK screen as the type specified when the FILE was opened (up to four letters).

(continued)

FOpen *(continued)*

Assumptions: ***(continued)***

When a file is opened with **FOpen** in FM_WRITE or FM_APPEND mode the associated variable is locked and inaccessible by any other routines in the system. It must be closed with **FClose** to return the variable to not in-use mode, to write the file type and the GEN_DATA_TAG, and to close the associated buffer. For FILEs opened in FM_READ mode, the **FClose** will merely update the mode of the file in the FILES structure to closed and clear the associated error status.

There is no separate mode to open a file for both reading and writing. However, if a file is opened in FM_APPEND mode the contents of the file are not erased and the file may be subsequently positioned to any location in the file (random access) and either read from or written to.

For all access modes the given name must not be a system variable. For FM_READ/APPEND the name must be a third-party data-type. For FM_WRITE the file must not exist or if it does it must be a third-party data-type and the variable must not be locked or in-use.

Note that for files in READ mode **FClose** merely updates the *fsPtr*->fileMode field to indicate it is closed. For WRITE mode it must be called to update information in the file needed by the system. While a file is in WRITE mode it is marked as in-use so that no other application, or the system, will try to access it (it will not be visible in the VAR-LINK screen also). Files in WRITE or APPEND mode may be accessed randomly by using **FSetPos**.

> **NOTE:** Any number of files may be simultaneously opened in READ mode. If multiple files are opened simultaneously in WRITE (or APPEND) mode than all but the first file must already exist before they are opened. Use the **FCreate** routine if they need to be created as empty files before they are opened with **FOpen**.

Side Effects: May cause heap compression or invalidate any current HSYMs.

Availability: On AMS 2.00 and higher.

TI-89 / TI-92 Plus Differences: None

See Also: **FAccess, FClose, FRead, FSetPos, FStatus, FWrite**

(continued)

FOpen *(continued)*

Example:

```
BOOL LocAppViewer( AppID appID, BYTE *type, WINDOW *w, HSYM hSym )
{  char fName[MAX_SYM_LEN];
   FILES f1;
   WORD c;

   WinFont( w, F_8x10 );
   if (0 == stricmp((char *) type,"APP1")) {
      if (HSYMtoName( hSym, (BYTE *) fName )) {
         if (FS_OK == FOpen(fName, &f1, FM_READ, "APP1" )) {
            while (!FEof(&f1)) {
               c = FGetC(&f1);
               WinChar( w, c );
            }
         }
      }
      return TRUE;
   }
}
```

FPutC

Declaration: WORD **FPutC** (short *byte*, FILES * *fsPtr*)

Category(ies): Files

Description: Write a byte to a file opened in WRITE mode.

Inputs: *byte* — BYTE to write.

fsPtr — Pointer to a FILES structure previously opened with **FOpen**.

Outputs: Return FS_OK if successful. FS_ERROR if file is not opened for write mode or FS_MEMORY if the system is out of memory.

> **NOTE:** Any error will cause the file status to be set to FS_ERROR so that multiple writes may be performed without checking the return status as long as an **FStatus** is done at the end to make sure all of the writes were successful.

Assumptions: The FILES structure pointed to by *fsPtr* must have been opened with **FOpen** in write mode.

Side Effects: May cause heap compression.

Availability: On AMS 2.00 and higher.

TI-89 / TI-92 Plus Differences: None

See Also: **FOpen, FRead, FSetPos, FStatus**

Example:

```
FILE f1;
if (FS_OK == FOpen("APPDATA", &f1, FM_WRITE, "APP1" )) {
   /* Can check each write */
   for (int j = 1; j <= 9; j++)
      if (FS_OK != FPutC( '2', &f1)) {
         Disp("ERROR writing to file");
         return;
      }
   /* Or can check FStatus after all writes */
   FPutC( 1, &f1 );
   FPutC( 2, &f1 );
   if (FS_OK != FStatus( &f1 )) {
      Disp("ERROR writing to file");
      return;
   }
}
```

FRead

Declaration: WORD **FRead** (void * *buffer*, FSWORD *bytesToRead*, FILES * *fsPtr*)

Category(ies): Files

Description: Read *bytesToRead* BYTEs from an open file (may be in either READ or WRITE mode) into *buffer*.

Inputs:

buffer	—	Pointer to a buffer of at least *bytesToRead* bytes in length.
bytesToRead	—	Number of bytes to read.
fsPtr	—	Pointer to FILES structure previously opened with **FOpen**.

Outputs:

FS_OK	—	File read was successful.
FS_EOF	—	Read past end of file.

Assumptions: None

Side Effects: None

Availability: On AMS 2.00 and higher.

TI-89 / TI-92 Plus Differences: None

See Also: **FOpen, FGetC, FSetPos**

Example:

```
WORD status;
char buf[6];
FILE f1;
if (FS_OK == FOpen("APPDATA", &f1, FM_WRITE, "APP1" )) {
   status = FWrite( "1234", 4, &f1 );
   FClose( &f1 );
   if (FS_OK == status) {
      FOpen( "APPDATA", &f1, FM_READ, "APP1");
      status = FRead( buf, 3, &f1 );  /* status == FS_OK */
      status = FRead( buf, 2, &f1 );  /* status == FS_EOF */
      FClose( &f1 );
   }
}
```

FSetBufSize

Declaration: FSWORD **FSetBufSize** (FILES * *fsPtr*, FSWORD *newBufSize*)

Category(ies): Files

Description: Set the buffer size of a file. The buffer size determines how much memory is reallocated to the file every time a write needs more memory from the heap. The default size (128 bytes) is set when the file is opened and should be sufficient for most uses. Setting a larger value will make writes faster but at the cost of possibly prematurely running out of memory.

Inputs:

fsPtr — Pointer to FILES structure previously opened with **FOpen** for write mode.

newBufSize — Size of new buffer.

Outputs: Returns value stored in *newBufSize*.

Assumptions: None

Side Effects: No immediate effect, the buffer size takes effect when the current buffer becomes full.

Availability: On AMS 2.00 and higher.

TI-89 / TI-92 Plus Differences: None

See Also: **FOpen, FWrite**

Example:

```
FILE f1;
if (FS_OK == FOpen("APPDATA", &f1, FM_WRITE, "APP1" )) {
   FSetBufSize( &f1, 500 );  /* we will be writing a lot of data to this file */
   writeLotsOfData( &f1 );
}
```

FSetPos

Declaration: FSWORD **FSetPos** (FILES * *fsPtr*, FSWORD *pos*)

Category(ies): Files

Description: Set the position of the next read or write for an opened file, return the new position (which may be less if the EOF is exceeded).

Inputs: *fsPtr* — Pointer to FILES structure previously opened with **FOpen**.

pos — File position (0 being the first byte in the file) to set.

Outputs: New file position.

Assumptions: The FILES structure pointed to by *fsPtr* must have been previously opened with **FOpen**.

Side Effects: None

Availability: On AMS 2.00 and higher.

TI-89 / TI-92 Plus Differences: None

See Also: **FOpen, FGetPos, FEof**

Example:

```
FILES f1;
char buf[20];
static char FTYPE2 = "DAT";

if (FS_OK == FOpen( "f3", &f1, FM_WRITE, FTYPE2 )) {
    FWrite( "abc", 3, &f1 );
   FSetPos( &f1, 0 );
   FPutC( 'A', &f1 );
   FSetPos( &f1, 5 );                          /* should go to EOF pos */
   FPutC( 'D', &f1 );
   FClose( &f1 );
   if (FS_OK == FOpen("f3", &f1, FM_APPEND, FTYPE2)) {
      FWrite( "eF", 2, &f1 );
      FSetPos( &f1, 0 );
      FRead( &buf, 6, &f1 );
      if (memcmp(buf, "AbcDeF", 6))
         Disp ("ERROR: file error");
      FClose( &f1 );
   }
}
```

FSetSize

Declaration: FSWORD **FSetSize** (FILES * *fsPtr*, FSWORD *fileSize*)

Category(ies): Files

Description: Truncate the size of a file opened in WRITE mode to *fileSize* which may not exceed the current file size. Return the new file size, but no more than the current size. Note that in READ mode, the current file size is always returned without making any changes.

Inputs:
fsPtr — Pointer to FILES structure previously opened with **FOpen** for write mode.

fileSize — New file size (may not exceed current file size).

Outputs: New file size (will not exceed current file size).

Assumptions: None

Side Effects: None

Availability: On AMS 2.00 and higher.

TI-89 / TI-92 Plus Differences: None

See Also: **FOpen, FGetSize**

Example: See **FGetSize**.

FSetVer

Declaration: BYTE **FSetVer** (FILES * *fsPtr*, BYTE *newVer*)

Category(ies): Files

Description: Change the version number of an opened file and return the old version number. Note that the file can be in read or write mode.

Inputs:
fsPtr — Pointer to FILES structure previously opened with **FOpen** for read or write mode.

newVer — New version number.

Outputs: Old version number of a file.

Assumptions: The version number of a file is stored in the file's symbol table entry. It is a BYTE and is used by the IO system to verify that a newer version of a symbol is not sent to an older TI‑89, TI‑92, or TI‑92 Plus.

Side Effects: None

Availability: On AMS 2.00 and higher.

TI‑89 / TI‑92 Plus Differences: None

See Also: **FOpen**

Example:

```
FILES f1;
BYTE origVer;
char Buf[40];
if (FS_OK == FOpen("f1", &f1, FM_READ, "DAT"))  {
   origVer = FSetVer( &f1, 1 );
  sprintf( buf, "Original version: %d\r\n", origVer );
   Disp( buf );
  FClose( &f1 );
}
```

FStatus

Declaration: WORD **FStatus** (FILES * *fsPtr*)

Category(ies): Files

Description: Return the status of a file. FS_OK if no errors have occurred or FS_ERROR if any errors have occurred. Note that errors accumulate so that multiple writes may be done on a file as long as the status is checked after the last write. The only way to clear the status is to close the file.

Inputs: *fsPtr* — Pointer to FILES structure previously opened with **FOpen**.

Outputs: FS_OK or FS_ERROR.

Assumptions: None

Side Effects: None

Availability: On AMS 2.00 and higher.

TI-89 / TI-92 Plus Differences: None

See Also: **FOpen, FClose, FWrite, FPutC**

Example:

```
FILE f1;
if (FS_OK == FOpen("APPDATA", &f1, FM_WRITE, "EXT" )) {
   for( int i = 1; i <= 200; i++)
      FPutC( i, &f1 );
   FWrite( "123456", 6, &f1 );
   if (FS_OK != FStatus( &f1 )) {
      /* One of the FPutC calls or the FWrite ran out of memory */
      FClose( &f1 );
      Disp("ERROR writing to file");
      FDelete( "APPDATA" );   /* only have partial file */
      return;
   }
   FClose( &f1 );  /* all writes were successful */
}
```

FType

Declaration: WORD **FType** (const char * *fileName*, char * *buf*)

Category(ies): Files

Description: Return the file type (max five bytes) for a file as a zero terminated string. Return FS_OK if successful. If error return FS_ERROR or FS_BAD_NAME (from **FOpen**). FS_ERROR is also returned if the file is opened successfully but the type field is invalid. Note that this will always fail if the given file is already opened for WRITE mode.

Inputs:

fileName	—	String pointer to the name of the file to open.
buf	—	Pointer to a buffer of at least five bytes in length.

Outputs:

FS_OK	—	Successful.
FS_ERROR or FS_BAD_NAME	—	Error.
buf	—	If successful, contains the description field (max five bytes including zero byte terminator).

Assumptions: File must not be opened for WRITE mode.

Side Effects: None

Availability: On AMS 2.00 and higher.

TI-89 / TI-92 Plus Differences: None

See Also: None

Example:

```
char buf[6];
FCreate("APPDATA", "DAT" );
if (FS_OK == FType("APPDATA", buf ))
   Disp( buf );  /* Create succeeded, buf will contain "DAT" */
```

FWrite

Declaration: WORD **FWrite** (void * *buffer*, FSWORD *bytesToWrite*, FILES * *fsPtr*)

Category(ies): Files

Description: Write to a file returning FS_OK if successful. FS_ERROR if the file is not in WRITE mode or FS_MEMORY if the system has run out of memory.

Inputs:

buffer	—	Buffer to write to file.
bytesToWrite	—	Number of bytes to write.
fsPtr	—	Pointer to FILES structure previously opened with **FOpen**.

Outputs:

FS_OK	—	If successful.
FS_ERROR	—	File is not in write mode.
FS_MEMORY	—	Out of memory.

> **NOTE:** Any error will cause the file status to be set to FS_ERROR so that multiple writes may be performed without checking the return status as long as an **FStatus** is done at the end to make sure all of the writes were successful.

Assumptions: File is opened in WRITE mode.

Side Effects: May cause heap compression.

Availability: On AMS 2.00 and higher.

TI-89 / TI-92 Plus Differences: None

See Also: None

Example:

```
FILE f1;
if (FS_OK == FOpen("APPDATA", &f1, FM_WRITE, "EXT" )) {
   FWrite( "123456", 6, &f1 );
   FWrite( "abc", 3, &f1 );
   /* can check each FWrite or check FStatus when done with all writes */
   if (FS_OK != FStatus( &f1 )) {
      FClose( &f1 );
      Disp("ERROR writing to file");
      FDelete( "APPDATA" );   /* only have partial file */
      return;
   }
   FClose( &f1 );  /* all writes were successful */
}
```

TokenizeName

Declaration: WORD **TokenizeName** (const char * *strFileName*, BYTE * *TokFName*)

Category(ies): Files, Symbol Table Utilities

Description: Convert a filename in standard C string format to a tokenized name. Use the TokNameRight MACRO to get the 'start' of the tokenized filename. This routine is not needed for the File routines since they take standard C string filenames and call **TokenizeName**.

> **NOTE:** This routine fully qualifies the name (adds the default folder if one is not specified) and so cannot be used to tokenize folder names.

Inputs:

strFileName	—	Input filename in standard C format (points to first character of the string).
TokFName	—	Address of a buffer of at least MAX_SYM_LEN bytes.

Outputs:

FS_OK	—	If tokenization is successful.
FS_BAD_NAME	—	Input name is reserved or invalid.
TokFName	—	Tokenized output is stored here (starting at the end).

Assumptions: The FILE routines are all passed filenames in standard C format (not tokenized names). They use this routine to tokenize the filenames which does two things. First it validates the name (makes sure it is a valid name and not a reserved name or system command or function) and second it stores it in the tokenized format (except for single alphabetic letters — zero byte terminator followed by the name followed by another zero byte terminator). This tokenized name is passed using the address of the second zero byte terminator (hence the use of the TokNameRight MACRO which just adds MAX_SYM_LEN -1 to the address passed to it) to get to the end of the tokenized buffer.

Side Effects: May cause heap compression.

Availability: On AMS 2.00 and higher.

TI-89 / TI-92 Plus Differences: None

See Also: **TokenizeSymName** (which this routine calls)

(continued)

TokenizeName *(continued)*

Example: There is an example routine, TokenizeFoldName, in the **FolderCur** description that will work for tokenizing folder names.

```
/* This piece of code tokenizes a symbol name stored in VarNameBuf (throwing an
   error if it was invalid) and then calls VarStore to store the value on top
   of the estack to this variable.
*/
if (FS_OK != TokenizeName(&VarNameBuf, tokPath))
   ER_throw( EXPECTED_VAR_OR_FUNC_ERROR );   /* Invalid name */
VarStore(TokNameRight(tokPath), STOF_ESI, 0, (long) top_estack);
```

Appendix A: System Routines — Graphing

CkValidDelta

Declaration: BYTE **CkValidDelta** (BCD16 *maxrng*, BCD16 *minrng*, BCD16 *delta*)

Category(ies): Graphing

Description: Checks to see if the exponent of *delta* is too small relative to the exponents of *maxrng* and *minrng* so that all the significant digits of *delta* would be shifted out of the floating-point mantissa when performing arithmetic.

Inputs:

maxrng — Final value in graph window variable sequence (e.g., xmax, ymax, tmax, etc.).

minrng — First value in graph window variable sequence (e.g., xmin, ymin, tmin, etc.).

delta — Increment value that will be used to compute sequence from *minrng* to *maxrng* (e.g., Δx, Δy, tstep, etc.).

Outputs: 1 if *delta* is valid, 0 if the exponent of *delta* is too small.

Assumptions: Assumes the sign of *delta* has already been verified as correct for computing a sequence from *minrng* to *maxrng*. It is valid for *maxrng* to be less than *minrng* if *delta* is negative.

Side Effects: None

Availability: All versions of the TI-89 / TI-92 Plus.

TI-89 / TI-92 Plus Differences: None

See Also: **StepCk**

Example:

```
/* make sure there is a unique x for every horizontal pixel in the graph */
if( !CkValidDelta( (gr_active->rngp)[GR_XMAX], (gr_active->rngp)[GR_XMIN],
         (gr_active->rngp)[GR_DELTAX] ))
   ER_throw( ER_RANGE );  /* cannot compute a unique x for every pixel */
```

cmd_clrdraw

Declaration: void **cmd_clrdraw** (void)

Category(ies): Graphing

Description: Causes the graph screen to be regraphed the next time the Graph application is displayed. All drawn objects will be erased. This is the TI-BASIC command ClrDraw.

Inputs: None

Outputs: None

Assumptions: None

Side Effects: The next time the graph screen is displayed, the entire graph will be redrawn, which may cause errors to be thrown or heap compression.

Availability: On AMS 1.05 and higher.

TI-89 / TI-92 Plus Differences: None

See Also: **cmd_clrgraph, GT_Regraph_if_neccy**

Example:

```
cmd_clrdraw();  /* set flags to cause regraph */
if (EV_currentApp == EV_getAppID("TIGRAPH"))
   GT_Regraph_if_neccy();  /* if graph is active window, regraph now */
```

cmd_clrgraph

Declaration: void **cmd_clrgraph** (void)

Category(ies): Graphing

Description: Clears any functions or expressions that were graphed with the TI-BASIC Graph or Table command by deleting the temporary folder used for storing those equations. This is the TI-BASIC ClrGraph command.

Inputs: None

Outputs: None

Assumptions: None

Side Effects: None

Availability: On AMS 1.05 and higher.

TI-89 / TI-92 Plus Differences: None

See Also: **cmd_newprob**

Example: The **cmd_newprob** function uses **cmd_clrgraph** and **cmd_clrdraw** to clear the entire graph screen.

```
void cmd_newprob( void ) {
   .
   .
   .
   cmd_clrdraw();
   cmd_clrgraph();
   .
   .
   .
}
```

cmd_rclgdb

Declaration: void **cmd_rclgdb** (EStackIndex *name_idx*)

Category(ies): Graphing, Variables

Description: Recall a graph database (make it the current graph database) from the variable indexed by *name_idx*. This is the TI-BASIC command RclGDB. See chapter **14. Data Types** for a description of graph databases.

Inputs: *name_idx* — Indexes the name of the graph database to recall.

Outputs: None

Assumptions: None

Side Effects: May cause heap compression.

Availability: On AMS 1.05 and higher.

TI-89 / TI-92 Plus Differences: None

See Also: **cmd_stogdb**

Example: This example saves the current graph database, makes some changes (for whatever reason), and then finally restores the previous graph database from the saved one.

```
EStackIndex oldTop, name_idx;

oldTop = top_estack;
TRY
   push_quantum( END_TAG );
   if (TokenizeSymName( (BYTE *) "gdb1", 0 ) == NULL)
      ER_THROW( INVALID_PATHNAME_ERROR );
   name_idx = top_estack;
   cmd_stogdb( name_idx );
   .
   .
   .
   /* may change some of the graphed functions or graph window information here */
   .
   .
   .
   cmd_rclgdb( name_idx );
ONERR
   top_estack = oldTop;
   ERD_dialog( errCode, FALSE );
   return;
ENDTRY
top_estack = oldTop;
```

cmd_stogdb

Declaration: void **cmd_stogdb** (EStackIndex *name_idx*)

Category(ies): Graphing, Variables

Description: Store the current graph database into the variable indexed by *name_idx*. This is the TI-BASIC command StoGDB. See chapter **14. Data Types** for a description of graph databases.

Inputs: *name_idx* — Indexes the name of the graph database to store to.

Outputs: None

Assumptions: None

Side Effects: May cause heap compression.

Availability: On AMS 1.05 and higher.

TI-89 / TI-92 Plus Differences: None

See Also: **cmd_rclgdb**

Example: See **cmd_rclgdb**.

CptDeltax

Declaration: BYTE **CptDeltax** (GR_WIN_VARS * *ptr*)

Category(ies): Graphing

Description: Computes Window variable Δx using the viewing window values and window width in the GR_WIN_VARS struct pointed to by *ptr* and sets the GR_DIRTY flag (`ptr ->gr_win_flags & GR_DIRTY`) if the newly computed Δx is not equal to the old Δx. **CptDeltax** verifies that there will be a unique x coordinate for each pixel column.

Inputs: *ptr* — Pointer to the GR_WIN_VARS struct to use for viewing window values and window width (**gr_active** for the active graph window, **gr_other** for the second graph in two graph mode).

Outputs: Returns 0 if Δx could not be computed due to invalid viewing window values or if the new Δx is so small that it will not insure unique values for each pixel column, otherwise returns 1. If valid, the new Δx is stored in `(ptr ->rngp)[GR_DELTAX]`.

Assumptions: Assumes the window width in `ptr ->xmaxpix` is correct. If the window size has been changed since the graph was last displayed, `ptr ->xmaxpix` will not have been updated, so the new Δx will not be correct. In this case, Δx will automatically be recomputed the next time the graph is displayed, using the correct window width.

Side Effects: If the GR_DIRTY flag is set, the graph will be redrawn next time it is displayed.

Availability: All versions of the TI-89 / TI-92 Plus.

TI-89 / TI-92 Plus Differences: None

See Also: **CkValidDelta, CptDeltay**

Example:

```
/* make sure deltax and deltay are correct */
if(gr_active->gr_win_flags & GR_DIRTY)
         /* viewing window values may have changed, recompute deltax and deltay */
   if ( ! ( CptDeltax(gr_active) && CptDeltay(gr_active) ) )
      ER_throw( ER_RANGE );  /* cannot compute valid deltax and deltay */
```

CptDeltay

Declaration: BYTE **CptDeltay** (GR_WIN_VARS * *ptr*)

Category(ies): Graphing

Description: Computes Window variable Δy using the viewing window values and window height in the GR_WIN_VARS struct pointed to by *ptr* and sets the GR_DIRTY flag (`ptr ->gr_win_flags & GR_DIRTY`) if the newly computed Δy is not equal to the old Δy.

Inputs: *ptr* — Pointer to the GR_WIN_VARS struct to use for viewing window values and window height (**gr_active** for the active graph window, **gr_other** for the second graph in two graph mode).

Outputs: Returns 0 if Δy could not be computed due to invalid viewing window values, otherwise returns 1. If valid, the new Δy is stored in `(ptr ->rngp)[GR_DELTAY]`.

Assumptions: Assumes the window height in `ptr ->ymaxpix` is correct. If the window size has been changed since the graph was last displayed, `ptr ->ymaxpix` will not have been updated, so the new Δy will not be correct. In this case, Δy will automatically be recomputed the next time the graph is displayed, using the correct window height.

Side Effects: If the GR_DIRTY flag is set, the graph will be redrawn next time it is displayed.

Availability: All versions of the TI-89 / TI-92 Plus.

TI-89 / TI-92 Plus Differences: None

See Also: **CptDeltax**

Example:

```
/* make sure deltax and deltay are correct */
if(gr_active->gr_win_flags & GR_DIRTY)
         /* viewing window values may have changed, recompute deltax and deltay */
   if ( ! ( CptDeltax(gr_active) && CptDeltay(gr_active) ) )
      ER_throw( ER_RANGE );  /* cannot compute valid deltax and deltay */
```

CptFuncX

Declaration: BCD16 **CptFuncX** (BCD16 *incs*, GR_WIN_VARS * *ptr*)

Category(ies): Graphing

Description: Add the given number of Δx increments (*incs*) to the original xmin of the specified graph viewing window. The current xmin will be different from the original xmin only if panning has occurred in function mode. **CptFuncX** is the same as **XCvtPtoF**, except that increment 0 always corresponds to the original xmin, which may not be column 0 after panning.

Inputs:

incs — Number of Δx increments to the left (negative) or right (positive) of original xmin. Increment 0 always corresponds to the original xmin, before any panning that may have occurred.

ptr — Pointer to the GR_WIN_VARS struct to use for viewing window values (**gr_active** for the active graph window, **gr_other** for the second graph in two graph mode).

Outputs: Returns the x coordinate obtained by computing original xmin+Δx**incs* in BCD16 floating point format, rounded to 12 digits.

Assumptions: Assumes an x, y coordinate system, even in 3D mode.

Side Effects: None

Availability: All versions of the TI-89 / TI-92 Plus.

TI-89 / TI-92 Plus Differences: None

See Also: **XCvtPtoF**

Example:

```
for( inc = -10; inc <= 10; ++inc )
{  /* generate table of x values before and after xmin  */
              /* compute BCD x value corresponding to next increment */
   x = CptFuncX( inc, gr_active );
   .
   .
   .
}
```

CptIndep

Declaration: BYTE **CptIndep** (BCD16 * *min*, BCD16 * *newindep*, USHORT *inc*)

Category(ies): Graphing

Description: Compute the independent graph variable value as a floating-point value based on a set of range values and an integer increment (0 being the first increment, 1 the next, and so on).

Inputs:

min — Pointer to an array of BCD16 floating-point values defining the min, max, and step size of the independent variable: min[0] = min, min[1] = max, min[2] = step.

newindep — Pointer to the output independent variable.

inc — Current iteration.

Outputs: Returns 1 if the value pointed to by *newindep* was set to the next independent variable or 0 if the next value was out of range.

Assumptions: None

Side Effects: None

Availability: On AMS 1.05 and higher.

TI-89 / TI-92 Plus Differences: None

See Also: **CptFuncX**

(continued)

CptIndep *(continued)*

Example: This example creates a list of the independent values for the current graph mode from min to max and step size as determined by the current Window variables. The result is stored in a list called lst1.

```
Access_AMS_Global_Variables;
USHORT curinc = 0;
BCD16 xValue, *rangePtr;
BYTE lst1[] =  {0,'l','s','t','1',0};
EStackIndex saveTop = top_estack;

switch( gr_active->graph_mode)  {
   case GR_PAR: rangePtr = &((gr_active->rngp)[GR_TMIN]); break;
   case GR_POL: rangePtr = &((gr_active->rngp)[GR_THETMIN]); break;
   default: ER_throw( ER_GRAPH_MODE );
}
push_quantum( END_TAG );
while(CptIndep( rangePtr, &xValue, curinc )) {
   curinc++;
   push_Float( xValue );
}
push_reversed_tail( top_estack );
push_quantum( LIST_TAG );
VarStore( lst1+5, STOF_ESI, 0, top_estack );
top_estack = saveTop;
```

EQU_select

Declaration: BOOL **EQU_select** (SSHORT *cFunc*, EQU_SELECT *sel*)

Category(ies): Graphing

Description: Turn on/off/toggle function graph selection flag. This controls which functions have a check mark by them in the Y= screen and subsequently are graphed in the Graph screen.

This routine uses the current Graph mode setting to determine which type of functions are affected.

Inputs:

cFunc — Function number to select/deselect (1 . . . 99). Parametric graph mode allows functions yt1 . . . yt99 to be selected by specifying function numbers -1 . . . -99.

sel — SELECT_ON to graph function.
SELECT_OFF to deselect function from graph.
SELECT_TOGGLE to toggle function selection.

Outputs: Return TRUE if the graph function exists, otherwise return FALSE.

Assumptions: 3D graph mode expects no more than one function to be selected. If you select a 3D function to graph, make sure you deselect all other 3D functions.

Side Effects: None

Availability: All versions of the TI‑89 / TI‑92 Plus.

TI‑89 / TI‑92 Plus Differences: None

See Also: **EQU_setStyle**

Example:

```
BOOL selectfunc(SSHORT cFunc)
{
   Access_AMS_Global_Variables;
   BOOL bSel;

   bSel = EQU_select(cFunc, SELECT_ON);        /* select function to graph */
   if (gr_active->graph_mode == GR_3D && bSel) /* 3D mode and was func selected? */
   {
      SSHORT n;
      for (n = 1; n <= 99; n += 1)             /* deselect all other functions */
      {
         if (n != cFunc)                       /* except one just selected */
            EQU_select(n, SELECT_OFF);
      }
   }
   return bSel;
}
```

EQU_setStyle

Declaration: void **EQU_setStyle** (GR_WIN_VARS * *g*, SSHORT *cFunc*, USHORT *style*, BOOL *bSelect*, BOOL *bError*)

Category(ies): Graphing

Description: Set style of graph function.

Inputs:

g — Pointer to graph data segment, either **gr_active** or **gr_other**.

cFunc — Function number (1 . . . 99).

style — A value indicating which style to set (FA_LINE, FA_DOT, FA_THICK, FA_ANIMATE, FA_PATH, FA_ABOVE, FA_BELOW, FA_SQUARE).

bSelect — TRUE means select function for graphing, otherwise leave function graphing selection as is.

bError — TRUE means throw an error if function does not exist.

Outputs: None

Assumptions: **EQU_setStyle** does not check for unsuitable graph mode/style combinations. It is your responsibility to make sure you do not set an invalid graph style for the current graph mode. Here are graph mode/style combinations you should avoid.

- In Parametric, Polar, and Diff Equation modes, you cannot use styles FA_ABOVE and FA_BELOW.
- In Sequence mode, you cannot use styles FA_ANIMATE, FA_PATH, FA_ABOVE, and FA_BELOW.
- 3D graph mode accepts no style settings.

Side Effects: None

Availability: All versions of the TI‑89 / TI‑92 Plus.

TI‑89 / TI‑92 Plus Differences: None

See Also: **EQU_select**

Example:

```
/* Set style of function 1 (this is y1 in function graph mode) to shade above
   and select it to graph. Do not throw an error if y1 does not exist.
*/
EQU_setStyle(gr_active, 1, FA_ABOVE, TRUE, FALSE);
```

FindFunc

Declaration: HSYM **FindFunc** (BYTE *funcNum*, BYTE *funcName*[ ], HSYM * *ypar*)

Category(ies): Graphing

Description: Creates a function name in *funcName* for function number *funcNum* in the current graph mode (**gr_active**) and returns the HSYM of that variable if it is selected. Also sets the graph reference flag for that function if `gr_flags.gr_in_progress` is set. Returns H_NULL if the function is not found or not selected. In parametric mode, will return both HSYMs if either function is selected. Sets graph reference flags in both.

Inputs:

funcNum — Function number (i.e., 1 for y1(x) in function mode).

funcName — Address of BYTE array of at least 9 bytes to use for function name.

ypar — Address of an HSYM for yt function in parametric mode, not used in other modes.

Outputs: Returns HSYM of function.

funcName — Name of function found stored here (in tokenized form – so the first byte will be zero).

ypar — In parametric mode, HSYM of yt function.

Assumptions: None

Side Effects: None

Availability: All versions of the TI-89 / TI-92 Plus.

TI-89 / TI-92 Plus Differences: None

See Also: **FindGrFunc**

Example: CountActiveFuncs is a routine that returns the number of active graph functions for the current graph mode.

```
SWORD CountActiveFuncs( void )
{
   BYTE NameBuf[9];
   HSYM ypar;
   SWORD i, Count;

   for (i = 1, Count = 0; i <= 99; i++)
      if (FindFunc( i, NameBuf, &ypar))
         Count++;
   return Count;
}
```

FindGrFunc

Declaration: SYM_ENTRY * **FindGrFunc** (BYTE *funcNum*, HSYM * *hsym*, EStackIndex * *funcTag*, BYTE *funcName*[ ], UCHAR *xory*)

Category(ies): Graphing

Description: Create a function name in *funcName* for function number *funcNum* in the current graph mode (**gr_active**) and return a pointer to the symbol table entry for that function, its HSYM, and a pointer to its function tag.

Inputs:

funcNum — Function number (i.e. 1 for y1(x) in function mode).

hsym — Address of an HSYM.

funcTag — BYTE pointer.

funcName — Address of BYTE array of at least 9 bytes to use for function name.

xory — 'x' or 'y' for parametric function name.

Outputs: SYM_ENTRY of symbol if found, NULL otherwise.

hsym — Returned HSYM of symbol if found, NULL otherwise.

funcTag — Returned pointer to function tag of symbol if found.

funcName — Name of function found stored here (in tokenized form – so the first byte will be zero).

Assumptions: None

Side Effects: None

Availability: All versions of the TI-89 / TI-92 Plus.

TI-89 / TI-92 Plus Differences: None

See Also: **FindFunc**

Example: This example pushes a list of the names of all of the graph functions with defined functions for the current graph mode on the estack.

```
short fNum;
HSYM hsym;
SYM_ENTRY *symPtr;
BYTE fName[9];
EStackIndex funcTag;

push_quantum( END_TAG );
for (fNum = 99; fNum >= 1; fNum--) {
   if (symPtr = FindGrFunc( fNum, &hsym, &funcTag, fName, 'x' ))
      push_zstr( (char *) fName+1 ); /* 1st byte is zero */
}
push_quantum( LIST_TAG );
```

gr_CptIndepInc

Declaration: BYTE **gr_CptIndepInc** (BCD16 *indep*, BCD16 * *min*, USHORT * *inc*)

Category(ies): Graphing

Description: Compute the increment equal to or less than the given independent variable value for DE, Sequence, Parametric, and Polar mode. Use **XCvtFtoP** for function mode. See the example for 3D mode.

Inputs:

indep — Independent variable value.

min — Pointer to an array of BCD16 floating-point values defining the min, max, and step size of the independent variable: indep[0] = min, indep[1] = max, indep[2] = step.

inc — Pointer to address to store the computed increment.

Outputs: Returns 1 if the increment that corresponds to the given independent variable value (or next lowest if between increments) is valid and returns the increment in the location pointed to by *inc*. Returns 0 if the independent value was out of range.

Assumptions: None

Side Effects: None

Availability: All versions of the TI-89 / TI-92 Plus.

TI-89 / TI-92 Plus Differences: None

See Also: **XCvtFtoP**

(continued)

gr_CptIndepInc *(continued)*

Example: Given a floating-point independent value and a pointer to the min, max, and step values for the independent variable, ICvtFtoP returns the corresponding increment or 0xFFFF if the independent value is out of range.

```
WORD ICvtFtoP( BCD16 f, BCD16 *indep_rng )
{  Access_AMS_Global_Variables;
   WORD NewInc;

   switch (gr_active->graph_mode) {
      case GR_FUNC:
         return( (WORD) XCvtFtoP(f, gr_active) );
      case GR_DE:
      case GR_SEQ:
      case GR_PAR:
      case GR_POL:
         if (gr_CptIndepInc(f, indep_rng, &NewInc))
           return( NewInc );
      case GR_3D:
         return ((f - gr_active->rngp[GR_XMIN]) * (gr_active->rngp[GR_XGRID] /
                 (gr_active->rngp[GR_XMAX] - gr_active->rngp[GR_XMIN])));
   }
   return 0xFFFF;
}
```

gr_delete_fldpic

Declaration: void **gr_delete_fldpic** (GR_WIN_VARS * *ptr*)

Category(ies): Graphing, Variables

Description: Delete graph system variable fldpic if it exists. fldpic cannot be unlocked and deleted using other symbol table or variable routines since there may be two copies of fldpic in two graph mode. The handle to the data in fldpic is saved in **gr_active** and/or **gr_other**.

Inputs: *ptr* — **gr_active** or **gr_other**

Outputs: None

Assumptions: None

Side Effects: None

Availability: All versions of the TI-89 / TI-92 Plus.

TI-89 / TI-92 Plus Differences: None

See Also: None

Example:

```
gr_delete_fldpic( gr_active );
gr_delete_fldpic( gr_other );
```

gr_DispLabels

Declaration: void **gr_DispLabels** (GR_WIN_VARS * *ptr*)

Category: Graphing

Description: Draws the axis labels on the graph window (not in the backup screen), if the graph mode is function, parametric, polar, sequence, or differential equations, and the label format is on.

Inputs: *ptr* — Pointer to the GR_WIN_VARS struct containing the graph data and flags to use (**gr_active** for the active graph window, **gr_other** for the second graph in two graph mode).

Outputs: None

Assumptions: The graph screen is on the display.

Side Effects: None

Availability: All versions of the TI-89 / TI-92 Plus.

TI-89 / TI-92 Plus Differences: None

See Also: None

Example:

```
/* After drawing on the graph screen, replace labels that may have been
   corrupted if desired
*/
   gr_DispLabels( gr_active );  /* display labels if labels on */
```

gr_xres_pixel

Declaration: USHORT **gr_xres_pixel** (USHORT *pixel*, BYTE *f*)

Category: Graphing

Description: Finds the first pixel column number greater than or equal to *pixel* that is a multiple of the graph system variable xres. This is a valid point to be plotted on a function graph which only computes values at xres multiples. If *f* is not 0, `gr_active->panshift` is added to *pixel* before determining the output pixel column number. This is only necessary if panning may have occurred, since the xres multiples in function graphing are relative to the original viewing window, which may be different from the current viewing window. For example, column 0 is always computed when a function graph is first drawn, but after panning left or right, the current column 0 may not have a computed point on it anymore depending on the xres value. So to find a column on a valid multiple of xres after panning, `gr_active->panshift` must be taken into account.

Inputs:

pixel — Pixel column number. The leftmost column is column 0 in all graph windows.

f — If *f* is 0, `gr_active->panshift` is assumed to be 0 and nothing is added to *pixel* before the computation occurs. If *f* is not 0, `gr_active->panshift` is added to *pixel* before computation.

Outputs: The first pixel column number greater than or equal to *pixel* that is a multiple of xres. This pixel is valid to be a computed point on a function graph even if it is currently off-screen, as is the case when the rightmost column in a function graph is not a multiple of xres and a point beyond the graph must be computed in order to complete the graph to the edge of the window.

Assumptions: The current graph mode is function mode.

Side Effects: None

Availability: All versions of the TI-89 / TI-92 Plus.

TI-89 / TI-92 Plus Differences: None

See Also: None

Example:

```
/* Find the first increment of xres >= rightmost column as last point to compute. */
lastpix = gr_xres_pixel(gr_active->xmaxpix, 0);  /* last pt to cpt for func graph */
```

GraphActivate

Declaration: BOOL **GraphActivate** (BOOL *RealCoords*)

Category(ies): Graphing

Description: Activate the Graph app if not already active. If the activation fails (user presses ON) then return FALSE. Otherwise, return TRUE (and repaint screen before returning).

Inputs:

RealCoords	—	Set to TRUE if floating-point coordinates used, FALSE if pixel coordinates are used. An error is thrown if *RealCoords* is true and the grapher is in 3D mode.

Outputs:

TRUE	—	Success in activating the Graph app.
FALSE	—	Activation failed (user pressed ON or invalid functions being graphed).

Assumptions: None

Side Effects: May cause heap compression.

> **NOTE:** Activating another app will cause the current app to be deactivated (receive a CM_QUIT message). The app will have to reactivate itself to get the focus back. See the second example for a way to handle this.

Availability: All versions of the TI-89 / TI-92 Plus.

TI-89 / TI-92 Plus Differences: None

See Also: **GT_Regraph_if_neccy**

(continued)

GraphActivate *(continued)*

Example: The cmd_circle function, if it were part of an app, would be as shown below along with two of its helper routines (GetAttr and isShortFloat). It (like most of the TI-BASIC graphing commands and functions) uses **GraphActivate** to activate the Graph app and then it draws an ellipse to the graph window using the parameters passed to it.

The second example, appfocus.c, is a complete app that:

- Sets a global flag before activating the Graph app so that this app will ignore the quit message when the grapher is active. The app will still lose the focus and cannot write to any of its windows but it will not completely shut down.
- Activates the Graph app, drawing a circle just to do something.
- Reactivates itself so now it is back to being the currently active app.

```
/* Get a valid attribute value (0, 1, -1) and return its corresponding
   screen attribute (A_REVERSE, A_NORMAL, A_XOR).
*/
short GetAttr( EStackIndex i ) {
   SWORD RetInt;

   RetInt = GetValue( i, -1, 1 );
   if (RetInt == -1)
      return A_XOR;
   else if (RetInt == 0)
      return ((RetInt == 0) ? A_REVERSE : A_NORMAL);
}

/* If the given Float value is in the range of a short integer return that value;
   otherwise, throw a domain error.
*/
short isShortFloat( BCD16 Float ) {
   short RetInt;

   if (Float < FPN32768 || Float > FP32767)
      ER_THROW( ER_DOMAIN )
   RetInt = Float;
   return( RetInt );
}

void cmd_circle (EStackIndex i, EStackIndex j, EStackIndex k, EStackIndex m) {
   Access_AMS_Global_Variables;
   BCD16 rFlt;
   short a0, b0;
   BYTE OrigAttr;
```

(continued)

GraphActivate *(continued)*

```
   if (GraphActivate(TRUE)) {
      rFlt = ForceFloat( k );
      a0  = isShortFloat( rFlt / gr_active->rngp[GR_DELTAX] );
      b0  = isShortFloat( rFlt / gr_active->rngp[GR_DELTAY] );
      if (a0 < 0 || b0 < 0)
         ER_THROW( ER_DOMAIN );
      OrigAttr = WinAttr(gr_active->grwinp, GetAttr(m) );
      WinEllipse( gr_active->grwinp, XCvtFtoP(ForceFloat(i),gr_active),
                  YCvtFtoP(ForceFloat(j),gr_active), a0, b0 );
      WinAttr( gr_active->grwinp, OrigAttr );
   }
}


/* Example app to activate grapher and reactivate itself.
   AppFocus.c
*/

#define _92
#include "..\tiams.h"

static void AP_app(pFrame self, PEvent e);

FRAME(appObj, OO_SYSTEM_FRAME, 0, OO_APP_FLAGS, 3)
   ATTR(OO_APP_FLAGS, APP_INTERACTIVE)
   ATTR(OO_APP_NAME, "appfocus")
   ATTR(OO_APP_PROCESS_EVENT, &AP_app)
ENDFRAME

pFrame pAppObj = (pFrame)&appObj;
WINDOW appW;
BOOL graphActive;
AppID ourID;
short x, y;

void activateGraph( void )
{ Access_AMS_Global_Variables;

   graphActive = TRUE;  /* signal we are activating the grapher */
   if (GraphActivate( FALSE )) {
      WinEllipse(gr_active->grwinp, x++, y++, 25, 25 );
      EV_startApp( ourID, AP_START_CURRENT );
   }  else
      graphActive = FALSE;                  /* failed to activate grapher */
}

static void AP_app(pFrame self, PEvent e)
{  Access_AMS_Global_Variables;
   WIN_RECT appWR;
   WINDOW *winPtr = &appW;
```

(continued)

GraphActivate *(continued)*

```
   switch (e->command) {
      case CM_START:
         /* Do not open our window if already open. */
         if (!(winPtr->Next)) {
            ourID = EV_currentApp;
            appWR = *(e->info.startInfo.startRect);
            x = y = 50;
            if (WinOpen( winPtr, &appWR, WF_TTY | WF_DUP_SCR))
               WinClr( winPtr );
            else
               EV_quit();
         }
         break;
      case CM_ACTIVATE:
         EV_defaultHandler(e);
         WinActivate( winPtr );
         WinStr( winPtr, "Press 'G' to activate graph\n" );
         graphActive = FALSE;
         break;
      case CM_KEY_PRESS:
         if (tolower(e->info.keyInfo.keyCode) == 'g') {
            WinStr( winPtr, "Graph activated\n" );
            activateGraph();
         }  else
            EV_defaultHandler(e);
         break;
      case CM_QUIT:
         /* Ignore quit if just activating grapher. */
         if (!graphActive) {
            if (winPtr->Next) {
               WinClose( winPtr );
               winPtr->Next = NULL;
            }
         }
         break;
      case CM_WPAINT:
         DrawWinBorder( winPtr, &(winPtr->Window) );
         WinBackupToScr( winPtr );
         break;
      default:
         EV_defaultHandler(e);
      break;
   }
}
```

GrAxes

Declaration: void **GrAxes** (SWORD *panpixels*, GR_WIN_VARS * *grPtr*)

Category(ies): Graphing

Description: Draw axes for the graph on the screen, based on the current viewing window variables and format settings.

Inputs:

panpixels — Number of pixels screen was shifted for panning:

- < 0 — Number of pixels panned using the left arrow.
- 0 — Draw entire screen.
- > 0 — Number of pixels panned using the right arrow.

grPtr — Pointer to graph window structure (**gr_active** normally).

Outputs: None

Assumptions: Graph is active and not in 3D mode

Side Effects: None

Availability: All versions of the TI-89 / TI-92 Plus.

TI-89 / TI-92 Plus Differences: None

See Also: **GraphActivate**

Example:

```
if (gr_active->graph_mode != GR_3D)   /* no axes or stat in 3d */
   GrAxes( 0, gr_active );  /* draw axes on screen  */
```

GrClipLine

Declaration: BYTE **GrClipLine** (BCD16 *x1*, BCD16 *y1*, BCD16 *x2*, BCD16 *y2*, BCD16 * *nx1*, BCD16 * *ny1*, BCD16 * *nx2*, BCD16 * *ny2*, GR_WIN_VARS * *grPtr*)

Category(ies): Graphing

Description: Clip a line to the given graph window.

Inputs:

x1, y1, x2, y2 — End-points of line to clip.

grPtr — Graph window structure (**gr_active**, **gr_other**) defining window to clip to.

Outputs: Return 1 if output line (*nx1*, *ny1*, *nx2*, *ny2*) is valid and *x2*, *y2* was on screen.

Return a value > 1 if output line was valid and *x2*, *y2* was off screen.

Return 0 if the line is outside the specified window region.

nx1, ny1, nx2, ny2 — End-points of the line after it has been clipped to the given graph window.

Assumptions: Assumes an x, y coordinate system, even in 3D mode so should not be used for 3D graphing.

Side Effects: None

Availability: All versions of the TI-89 / TI-92 Plus.

TI-89 / TI-92 Plus Differences: None

See Also: **GrLineFlt**

(continued)

GrClipLine *(continued)*

Example: This example is the TI-BASIC Line command. It draws a line (coordinates: L1, L2, L3, L4) on the current graph using attribute newAttr. The source for the helper routine, GetAttr, is listed in the example for **GraphActivate**. **GrClipLine** is used to clip the given floating point coordinates to the current graph window.

```
void cmd_line (EStackIndex L1, EStackIndex L2, EStackIndex L3,
               EStackIndex L4, EStackIndex newAttr)
{
   Access_AMS_Global_Variables;
   BYTE OrigAttr;
   BCD16 x1, y1, x2, y2;

   if (GraphActivate(TRUE)) {
      OrigAttr = WinAttr(gr_active->grwinp, GetAttr(newAttr) );
      if (GrClipLine( ForceFloat(L1), ForceFloat(L2),
                      ForceFloat(L3), ForceFloat(L4),
         &x1, &y1, &x2, &y2, gr_active))
            WinLine( gr_active->grwinp, MakeWinRect(
               XCvtFtoP(x1,gr_active), YCvtFtoP(y1,gr_active),
               XCvtFtoP(x2,gr_active), YCvtFtoP(y2,gr_active)));
      WinAttr( gr_active->grwinp, OrigAttr );
   }
}
```

GrLineFlt

Declaration: void **GrLineFlt** (BCD16 *x1*, BCD16 *y1*, BCD16 *x2*, BCD16 *y2*, GR_WIN_VARS * *grPtr*, SWORD *pixcur*[ ], BYTE *funcAttr*, BYTE *pattern*)

Category(ies): Graphing

Description: Draw a line on the current graph based on the associated viewing window variables as defined by *grPtr*.

Inputs:

- *x1, y1* — First end-point of line to draw.
- *x2, y2* — Second end-point.
- *grPtr* — Graph window structure (**gr_active**, **gr_other**) defining the window to draw to and the viewing window variables.
- *pixcur* — An array of 2 SWORDs.
- *funcAttr* — Attribute to use to draw the line: FA_LINE, FA_DOT, FA_THICK, FA_ANIMATE, FA_PATH, FA_ABOVE, FA_BELOW, FA_SQUARE.
- *pattern* — If *funcAttr* is FA_ABOVE or FA_BELOW then *pattern* specifies the shade pattern: A_SHADE_V, A_SHADE_H, A_SHADE_NS, A_SHADE_PS. Shading will not occur unless `gr_flags.gr_in_progress` is set.

Outputs:

- *pixcur* — The pixel values of the 2nd end-point ([0] will be -1 if no plot).

Assumptions: The graph is active and not in 3D mode.

Side Effects: None

Availability: All versions of the TI-89 / TI-92 Plus.

TI-89 / TI-92 Plus Differences: None

See Also: **GrClipLine** (which this routine calls), **GraphActivate**

Example: If the graph is active (see **GraphActivate**) the DrawFBox routine shown below will draw a box on the graph screen with opposite corners (x0, y0) and (x1, y1).

```
void DrawFBox( BCD16 x0, BCD16 y0, BCD16 x1, BCD16 y1 )
{  Access_AMS_Global_Variables;
   SWORD pixcur[2];

   GrLineFlt( x0, y1, x0, y0, gr_active, pixcur, FA_LINE, 0 );
   GrLineFlt( x0, y0, x1, y0, gr_active, pixcur, FA_LINE, 0 );
   GrLineFlt( x1, y0, x1, y1, gr_active, pixcur, FA_LINE, 0 );
   GrLineFlt( x1, y1, x0, y1, gr_active, pixcur, FA_LINE, 0 );
}
```

GT_Regraph

Declaration: void **GT_Regraph** (void)

Category(ies): Graphing

Description: Force a regraph of the current graph.

Inputs: None

Outputs: None

Assumptions: Graph is active.

Side Effects: May cause heap compression.

Availability: All versions of the TI-89 / TI-92 Plus.

TI-89 / TI-92 Plus Differences: None

See Also: **GraphActivate, GT_Regraph_if_neccy**

Example: See **round12_err**.

GT_Regraph_if_neccy

Declaration: void **GT_Regraph_if_neccy** (void)

Category(ies): Graphing

Description: If the current graph needs to be regraphed then do it.

Inputs: None

Outputs: None

Assumptions: None

Side Effects: May cause heap compression.

Availability: All versions of the TI‑89 / TI‑92 Plus.

TI‑89 / TI‑92 Plus Differences: None

See Also: **GraphActivate**

Example: One of the first things the trace function calls is **GT_Regraph_if_neccy** to make sure that it is tracing a valid graph.

```
BOOL GT_Trace( WORD PromptId, BCD16 *RetVal, WORD Flags )
{
.
.
.
GT_Regraph_if_neccy();  /* need to regraph if graph not clean */
.
.
.
```

StepCk

Declaration: void **StepCk** (BCD16 * *indep*)

Category: Graphing

Description: Verifies that the input min, max, and step values in the array pointed to by *indep* are valid values for the independent variable in parametric mode (tmin, tmax, and tstep) or polar mode (θmin, θmax, and θstep). The function will return to the calling routine if the values are valid, otherwise an error is thrown.

Inputs: *indep* — Pointer to an array of BCD16 floating-point values where *indep*[0] = min, *indep*[1] = max, and *indep*[2] = step.

Outputs: None

Assumptions: None

Side Effects: An error is thrown if the input values are not valid for the independent variable in parametric mode or polar mode, such as the step value being negative when it should have been positive for the given min and max.

Availability: All versions of the TI-89 / TI-92 Plus.

TI-89 / TI-92 Plus Differences: None

See Also: **CkValidDelta**

Example:

```
if( gr_active->graph_mode == GR_PAR )
     /* parametric mode - make sure viewing window variables tmin,tmax,tstep OK */
   StepCk(&((gr_active->rngp)[GR_TMIN]));
```

XCvtFtoP

Declaration: SSHORT **XCvtFtoP** (BCD16 *x*, GR_WIN_VARS * *ptr*)

Category: Graphing

Description: Converts the input floating point x coordinate to a pixel column number based on the specified graph viewing window.

Inputs:

x — Floating point x coordinate. Valid *x* is not limited to the actual viewing window, allowing computation of off-screen pixels to the left (negative) or right of the visible window.

ptr — Pointer to the GR_WIN_VARS struct to use for viewing window values (**gr_active** for the active graph window, **gr_other** for the second graph in two graph mode).

Outputs: Returns a pixel column number or GXY_INVALID if the value computed is outside the range of SSHORT. The leftmost column in a window is column 0.

Assumptions: Assumes an x, y coordinate system, even in 3D mode.

Side Effects: None

Availability: All versions of the TI-89 / TI-92 Plus.

TI-89 / TI-92 Plus Differences: None

See Also: **XCvtPtoF, CptFuncX, YCvtPtoF, YCvtFtoP**

Example:

```
/* convert float coordinates to pixel coordinates and plot point on the pixel */
col = XCvtFtoP( x, gr_active );  /* find pixel column */
row = YCvtFtoP( y, gr_active );  /* find pixel row */
WinPixSet( gr_active->grwinp, col, row );  /* plot point */
```

XCvtPtoF

Declaration: BCD16 **XCvtPtoF** (SSHORT *x*, GR_WIN_VARS * *ptr*)

Category: Graphing

Description: Converts pixel column number *x* to the corresponding floating point x coordinate at the center of that column, based on the specified graph viewing window.

Inputs:

x — Column number. The leftmost column is column 0 in all graph windows. Valid column numbers are not limited to the actual window width, allowing computation for off-screen values to the left (negative column number) or right of the visible window.

ptr — Pointer to the GR_WIN_VARS struct to use for viewing window values (**gr_active** for the active graph window, **gr_other** for the second graph in two graph mode).

Outputs: Returns the x coordinate at the center of column *x* in BCD16 floating point format, rounded to 12 digits.

Assumptions: Assumes an x, y coordinate system, even in 3D mode.

Side Effects: Possible overflow error.

Availability: All versions of the TI-89 / TI-92 Plus.

TI-89 / TI-92 Plus Differences: None

See Also: **XCvtFtoP, CptFuncX, YCvtPtoF, YCvtFtoP**

Example:

```
for( col=0; col <= maxpix; ++col )
{
/* compute BCD x value corresponding to next pixel to plot */
   xplot = XCvtPtoF( col, gr_active );
   .
   .
   .
}
```

YCvtFtoP

Declaration: SSHORT **YCvtFtoP** (BCD16 *y*, GR_WIN_VARS * *ptr*)

Category: Graphing

Description: Converts the input floating point y coordinate to a pixel row number based on the specified graph viewing window.

Inputs:

y — Floating point y coordinate. Valid *y* is not limited to the actual viewing window, allowing computation of off-screen pixels above (negative) or below the visible window.

ptr — Pointer to the GR_WIN_VARS struct to use for viewing window values (**gr_active** for the active graph window, **gr_other** for the second graph in two graph mode).

Outputs: Returns a pixel row number or GXY_INVALID if the value computed is outside the range of SSHORT. The topmost row in a window is row 0.

Assumptions: Assumes an x, y coordinate system, even in 3D mode.

Side Effects: None

Availability: All versions of the TI-89 / TI-92 Plus.

TI-89 / TI-92 Plus Differences: None

See Also: **XCvtPtoF, CptFuncX, YCvtPtoF, XCvtFtoP**

Example:

```
/* convert float coordinates to pixel coordinates and plot point on the pixel */
col = XCvtFtoP( x, gr_active );  /* find pixel column */
row = YCvtFtoP( y, gr_active );  /* find pixel row */
WinPixSet( gr_active->grwinp, col, row );  /* plot point */
```

YCvtPtoF

Declaration: BCD16 **YCvtPtoF** (SSHORT *y*, GR_WIN_VARS * *ptr*)

Category: Graphing

Description: Converts pixel row number *y* to the corresponding floating point y coordinate at the center of that row, based on the specified graph viewing window.

Inputs:

y — Row number. The topmost row is row 0 in all graph windows. Valid row numbers are not limited to the actual window height, allowing computation for off-screen values above (negative row number) or below the visible window.

ptr — Pointer to the GR_WIN_VARS struct to use for viewing window values (**gr_active** for the active graph window, **gr_other** for the second graph in two graph mode).

Outputs: Returns the y coordinate at the center of row *y* in BCD16 floating-point format, rounded to 12 digits.

Assumptions: Assumes an x, y coordinate system, even in 3D mode.

Side Effects: Possible overflow error.

Availability: All versions of the TI-89 / TI-92 Plus.

TI-89 / TI-92 Plus Differences: None

See Also: **XCvtFtoP, CptFuncX, XCvtPtoF, YCvtFtoP**

Example:

```
GetWinCursor( gr_active->grwinp, &col, &row );
/* find BCD x, y coordinates corresponding to current free-moving cursor */
x = XCvtPtoF( col, gr_active );
y = YCvtPtoF( row, gr_active );
```

Appendix A: System Routines — Home Screen

See Also:

cmd_clrhome

Declaration: void **cmd_clrhome** (void)

Category(ies): Home Screen

Description: Delete expressions in Home screen history and clear home window.

This routine implements the TI-BASIC ClrHome command.

Inputs: None

Outputs: None

Assumptions: None

Side Effects: CAS arbitrary real and integer variable number counters are reset to zero.

Availability: On AMS 1.05 and higher.

TI-89 / TI-92 Plus Differences: None

See Also: **cmd_clrio**

Example:

```
cmd_clrhome();
```

cmd_disphome

Declaration: void **cmd_disphome** (void)

Category(ies): Home Screen, Operating System

Description: Start the Home application if it is not already running. Make sure the Home window is being displayed switching from the Program I/O window if necessary. Activate the author line for user input.

This routine implements the TI-BASIC DispHome command.

Inputs: None

Outputs: None

Assumptions: None

Side Effects: The OS will deactivate or terminate the current application to activate the Home screen.

Availability: On AMS 1.05 and higher.

TI-89 / TI-92 Plus Differences: None

See Also: **EV_quit**

Example:

```
cmd_disphome();
```

HomeAlone

Declaration: WORD **HomeAlone** (void)

Category(ies): Home Screen

Description: Useful if one needs to do background processing while nothing else is going on.

Inputs: None

Outputs: Returns true if the Home screen is active on a full screen and events are not captured.

Assumptions: None

Side Effects: None

Availability: All versions of the TI-89 / TI-92 Plus.

TI-89 / TI-92 Plus Differences: None

See Also: None

Example:

HomeExecute

Declaration: void **HomeExecute** (UCHAR * *sCmd*, USHORT *nCmd*)

Category(ies): Home Screen, Strings, Expression Evaluation / Algebraic Simplification

Description: Executes string on Home screen.

Inputs:
sCmd — Pointer to string to execute. This is not a zero-terminated string.
nCmd — Length of string.

Outputs: None

Assumptions: The Home screen app is started if necessary, *nCmd* is pasted to the author line, and a [ENTRY] keypress event is sent.

Side Effects: May cause heap compression. May throw errors.

Availability: All versions of the TI-89 / TI-92 Plus.

TI-89 / TI-92 Plus Differences: None

See Also: Not applicable.

Example:

```
UCHAR *pCommand;
   .
   . /* Point pCommand to command string */
   .
HomeExecute(pCommand, strlen((char *)pCommand));
```

HS_getAns

Declaration: HANDLE **HS_getAns** (USHORT *j*)

Category(ies): Home Screen

Description: Gets an answer from the Home screen.

Inputs: *j* — Number from 1 to 99.

Outputs: Returns the handle to the *jth* answer from the Home screen. Returns H_NULL if the *jth* answer does not exist. By default, the home app keeps only the 30 most recent answers. The user can change this limit in the Home Format dialog.

Assumptions: You should not free or modify the contents of this handle as the home app needs it to format its screen properly.

Side Effects: None

Availability: All versions of the TI‑89 / TI‑92 Plus.

TI‑89 / TI‑92 Plus Differences: None

See Also: **HS_getEntry**

Example:

```
HANDLE hAns = HS_getAns(1); /* Get handle to most recent answer */
```

HS_getEntry

Declaration: HANDLE **HS_getEntry** (USHORT *j*)

Category(ies): Home Screen

Description: Gets an entry from the Home screen.

Inputs: *j* — Number from 1 to 99.

Outputs: Returns the handle to the *jth* entry from the Home screen. Returns H_NULL if the *jth* entry does not exist. By default, the home app keeps only the 30 most recent entries. The user can change this limit in the Home Format dialog.

Assumptions: You should not free or modify the contents of this handle as the home app needs it to format its screen properly.

Side Effects: None

Availability: All versions of the TI‑89 / TI‑92 Plus.

TI‑89 / TI‑92 Plus Differences: None

See Also: **HS_getAns**

Example:

```
HANDLE hAns = HS_getEntry(1); /* Get handle to most recent entry */
```

HS_popEStack

Declaration: HANDLE **HS_popEStack** (void)

Category(ies): Home Screen

Description: Pops the top element from the estack into heap memory.

Inputs: None

Outputs: Returns a handle to the popped contents.

Assumptions: If there is insufficient memory to copy the estack into the heap, the estack is popped anyway, and ER_MEMORY is thrown.

Side Effects: May cause heap compression.

Availability: All versions of the TI-89 / TI-92 Plus.

TI-89 / TI-92 Plus Differences: None

See also: None

Example:

```
HANDLE h;
   .
   .
   .
TRY
   h = HS_popEStack();
   /* h now contains handle to popped estack */
ONERR
   /* Could not allocate memory to pop estack */
   .
   .
   .
ENDTRY
```

Appendix A: System Routines — Interrupts

See Also:

idle

Declaration: void **idle** (void)

Category(ies): Interrupts

Description: Sets the calculator in low power mode until an interrupt (usually a timer, keypress, or link activity) occurs. Leaves the LCD powered and the screen visible.

Inputs: None

Outputs: None

Assumptions: None

Side Effects: None

Availability: All versions of the TI-89 / TI-92 Plus.

TI-89 / TI-92 Plus Differences: None

See Also: **off**

Example:

```
/* Try to get the USER timer, if already in use, return FALSE.
   Else, wait up to a minute for a key.
   If a key is pressed within a minute, return TRUE.
   If no key is pressed within a minute return FALSE

   This code is intended to illustrate the usage of
   timer routines: off, idle, kbhit, ngetchx, and pushkey.
*/

#include "tiams.h"
enum {EX_CONTINUE = 0, /* ERROR */
      EX_BREAK,     /* break key */
      EX_TIMEOUT,   /* timer expired before key hit */
      EX_KEY,       /* key hit before timer expired */
      EX_2ND_OFF,   /* turn calc off and return when turned back on */
      EX_OPT_OFF    /* turn calc off and continue when turned back on */
} i;
WORD waitOneMinForKey( void )
{
   Access_AMS_Global_Variables;
   i = EX_CONTINUE;
   if( !OSRegisterTimer( USER, ONE_MINUTE ))
      return FALSE;             /* USER TIMER already in use */

   while ( TRUE )  {
```

(continued)

idle *(continued)*

```
    if ( !OSTimerExpired( APD ) && !OSTimerExpired( USER ) && !kbhit() )  {
       idle();   /* Stop CPU, but keep LCD alive . . . save power */
    }

    if ( OSTimerExpired( APD ) ) {  /* if the APD timer went off */
       off();                  /* turn the calculator off until they turn it on */
       OSResetTimer( APD );    /* reset APD */
       continue;               /* go back to looking for keys */
    }

    if ( OSTimerExpired( USER ) )    /* Timer elapsed */
       i = EX_TIMEOUT;         /* If key hit and timer, key hit takes precedence. */

    if ( kbhit() )  {
       switch ( i = ngetchx() )  {
          case KB_ON+KB_OPTION:
             i = EX_OPT_OFF;   /* for next loop around */
             off();            /* turn the calculator off until they turn it on */
             break;
          case KB_OFF:
             i = EX_2ND_OFF;   /* signal they want to quit */
             off();            /* turn the calculator off until they turn it on */
             break;
          default:
             i = EX_KEY;       /* regular key hit */
             pushkey( i );     /* push the key back to key queue */
       }
       OSResetTimer( APD );    /* reset APD */
    }

    if ( OSOnBreak )           /* Break key */
       i = EX_BREAK;

    if ( !i )
       continue ;              /* go back to sleep */
    else  {
       OSFreeTimer( USER );
       return i;
    }
 }  //while ( TRUE )}          /* waitOneMinForKey */
```

off

Declaration: void **off** (void)

Category(ies): Interrupts

Description: Sets the calculator in low power mode with the LCD off until link activity or the ON key.

Inputs: None

Outputs: None

Assumptions: None

Side Effects: None

Availability: All versions of the TI-89 / TI-92 Plus.

TI-89 / TI-92 Plus Differences: None

See Also: None

Example: See **idle**.

OSSetSR

Declaration: WORD **OSSetSR** (WORD *InterruptLevel*)

Category(ies): Interrupts

Description: Sets the 68000 status register to block all interrupts at and below *InterruptLevel*.

Inputs: *InterruptLevel* — One of:

0x0700	—	Block all interrupts
0x0600	—	Block [ON] key and below
0x0500	—	Block Timers and below
0x0400	—	Block Link and below
0x0300	—	Block Slow Clock (not implemented) and below
0x0200	—	Block Keyboard and below
0x0100	—	Block key scan
0x0000	—	Enable all interrupts

Outputs: Previous mask value.

Assumptions: The 68000 status register is set to *InterruptLevel* and 0x0F00.

Side Effects: The calculator is normally set to run with interrupts enabled. The previous value of the interrupt mask should always be restored prior to swapping out or exiting. Setting an interrupt mask above 0x0500 will disable the calculator's ability to save state on power outage, and is discouraged.

Availability: All versions of the TI-89 / TI-92 Plus.

TI-89 / TI-92 Plus Differences: None

See Also: None

Example:

```
saveSR=OSSetSR(0x700);  /* disable interrupts */
.                       /* do some stuff */
.
.
OSSetSR(saveSR);        /* restore interrupts */
```

Appendix A: System Routines — Keyboard

See Also:

alphaLockOff

Declaration: void **alphaLockOff** (FLAGS8 * *saveAlpha*)

Category(ies): Keyboard

Description: Returns the current alpha-lock keyboard status in *saveAlpha* and turns alpha-lock status OFF.

Inputs: *saveAlpha* — Pointer to FLAGS8 structure.

Outputs: *saveAlpha* — Previous alpha-lock status.

Assumptions: None

Side Effects: The alpha-lock status remains until changed by the user or another app.

Availability: On AMS 2.00 and higher.

TI-89 / TI-92 Plus Differences: For compatibility reasons, this routine is available on the TI-92 Plus but does nothing on that platform.

See Also: **alphaLockOn, restoreAlphaLock**

Example:

```
FLAGS8 saveStat;
alphaLockOff( &saveStat );
editField();  /* user may be entering NUMERIC data here */
restoreAlphaLock( &saveStat );
```

alphaLockOn

Declaration: void **alphaLockOn** (FLAGS8 * *saveAlpha*)

Category(ies): Keyboard

Description: Returns the current alpha-lock keyboard status in *saveAlpha* and turns alpha-lock status ON.

Inputs: *saveAlpha* — Pointer to FLAGS8 structure.

Outputs: *saveAlpha* — Previous alpha-lock status.

Assumptions: None

Side Effects: The alpha-lock status remains until changed by the user or another app.

Availability: On AMS 2.00 and higher.

TI-89 / TI-92 Plus Differences: For compatibility reasons, this routine is available on the TI-92 Plus but does nothing.

See Also: **alphaLockOff, restoreAlphaLock**

Example:

```
FLAGS8 saveStat;
alphaLockOn( &saveStat );
editField();  /* user may be entering TEXT data here */
restoreAlphaLock( &saveStat );
```

GetAlphaStatus

Declaration: BYTE **GetAlphaStatus** (void)

Category(ies): Keyboard

Description: On the TI-89, returns the status of AlphaLock. Since there is no AlphaLock on the TI-92 Plus, 0 is always returned.

Inputs: None

Outputs: On the TI-89, returns 1 if AlphaLock is on and 0 if AlphaLock is off. On the TI-92 Plus, always returns 0.

Assumptions: None

Side Effects: None

Availability: All versions of the TI-89 / TI-92 Plus.

TI-89 / TI-92 Plus Differences: This function is only for the TI-89. If called on the TI-92 Plus, 0 will always be returned.

See Also: **SetAlphaStatus**

Example:

GKeyFlush

Declaration: void **GKeyFlush** (void)

Category(ies): Keyboard

Description: Remove any keys from the keyboard buffer.

Inputs: None

Outputs: None

Assumptions: None

Side Effects: The keyboard buffer is global to all apps.

Availability: All versions of the TI‑89 / TI‑92 Plus.

TI‑89 / TI‑92 Plus Differences: None

See Also: **kbhit**

Example:

```
GKeyFlush();  /* in case key already queued up */
key = DlgMessage( "WARNING", "Delete all user data?" );
```

GKeyIn

Declaration: WORD **GKeyIn** (SCR_RECT * *rBlink*, WORD *Flags*)

Category(ies): Keyboard

Description: Wait for and return the next keypress (bypassing the event manager).

Inputs:

rBlink — If not null, this is the rectangular region that defines the cursor that will be "blinked" (by XORing a reverse/normal video rectangle) while waiting on a key. If it is NULL then there is no cursor.

Flags —

GKF_NORMAL — No special key processing.

GKF_MODAL
If the key pressed is a "modal" key ([OFF], [APPS], [◆]-[APPS], [2nd]-[APPS], [MODE], [VAR-LINK], [MEM], [QUIT], or any of the built-in app keys: [HOME], [GRAPH], . . .) then [ESC] is returned and the key is repushed onto the keyboard buffer. Note that DIALOG boxes usually set GKF_MODAL and GKF_SYS so that if a user presses [VAR-LINK] in the dialog box, the dialog box is closed and then the [VAR-LINK] key is acted on (though there is a flag to allow [VAR-LINK] to be activated on inside a dialog box).

GKF_SYS — Same as GKF_MODAL only for "system" keys: [MATH], [CATALOG], [CHAR], [CUSTOM].

GKF_REPUSH_KEY — Whatever key is pressed, return it and repush that key onto the keyboard buffer.

GKF_NO_EVS — Ignore [CATALOG] key.

Outputs: Keypressed.

Assumptions: Apps should use the event manager to process keys. **GKeyIn** is only used in special cases where the event manager is not accessible.

Side Effects: By-passes the event manager's handling of keys so this routine should be used with caution.

Availability: All versions of the TI-89 / TI-92 Plus.

(continued)

GKeyIn *(continued)*

TI-89 / TI-92 Plus Differences: None

See Also: **GKeyFlush, kbhit**

Example: See **WinHome** for an example using SCR_RECT.

```
/* pause to show results before redrawing backup graph image */
GKeyIn( NULL, GKF_REPUSH_KEY );
WinBackupToScr( &graphWindow );
```

kbhit

Declaration: WORD **kbhit** (void)

Category(ies): Keyboard

Description: Returns TRUE if there is a key pending in the key queue.

Inputs: None

Outputs: TRUE if a key is pending in the key queue.

Assumptions: None

Side Effects: None

Availability: All versions of the TI-89 / TI-92 Plus.

TI-89 / TI-92 Plus Differences: None

See Also: None

Example:

```
while ( !kbhit () );     /* wait for a keypress */
return ( ngetchx ( ) );  /* return the key */
```

KeyYesOrNo

Declaration: short **KeyYesOrNo** (WORD *Key*)

Category(ies): Keyboard

Description: Return TRUE if *Key* represents YES (ENTER or the first letter of “yes” for the current language), FALSE if it represents NO (ESC or the first letter of “no” for the current language), -1 if it is neither.

Inputs: Key value.

Outputs: TRUE, FALSE, -1.

Assumptions: None

Side Effects: None

Availability: On AMS 2.00 and higher.

TI-89 / TI-92 Plus Differences: None

See Also: **GKeyIn**

Example:

```
WORD key; short deleteIt;
.
.
.
/* wait for user to confirm deletion of config file */
while (TRUE) {
   key = GKeyIn( NULL, GKF_NORMAL );
   if (-1 == (deleteIt = KeyYesOrNo(key)))
      continue;
}
```

ngetchx

Declaration: WORD **ngetchx** (void)

Category(ies): Keyboard

Description: Return the next key in the key queue.

Inputs: None

Outputs: If there is no key in the key queue block. Else returns the key value.

Assumptions: None

Side Effects: None

Availability: All versions of the TI-89 / TI-92 Plus.

TI-89 / TI-92 Plus Differences: None

See Also: List of key definition equates in tiams.h, **OSFastArrows**.

Example:

```
while ( !kbhit () ) ;   /* wait for a key */
return ngetchx();      /* return the key value */
```

OSCheckBreak

Declaration: WORD **OSCheckBreak** (void)

Category(ies): Keyboard

Description: Test the break flag.

Inputs: None

Outputs: True if the ON key was pressed with break enabled, false otherwise.

Assumptions: None

Side Effects: The ON key is tied to a high level interrupt (level 6), and is used on the calculator as a “break” key to allow the user to interrupt the current flow of activity. This call returns true and clears all pending keys from the key queue if the ON key has been pressed at least once since the last call to **OSClearBreak**.

Availability: All versions of the TI-89 / TI-92 Plus.

TI-89 / TI-92 Plus Differences: None

See Also: **OSOnBreak, OSClearBreak, OSEnableBreak, OSDisableBreak**

Example:

```
while (! OSCheckBreak())   /* increment i until ON key is pressed */
   i++;
```

OSClearBreak

Declaration: void **OSClearBreak** (void)

Category(ies): Keyboard

Description: Clear the break flag.

Inputs: None

Outputs: None

Assumptions: None

Side Effects: Sets the **OSOnBreak** flag to FALSE.

Availability: All versions of the TI-89 / TI-92 Plus.

TI-89 / TI-92 Plus Differences: None

See Also: **OSOnBreak, OSCheckBreak, OSEnableBreak, OSDisableBreak**

Example:

```
if(errCode == ER_BREAK)
   OSClearBreak();
```

OSDisableBreak

Declaration: void **OSDisableBreak** (void)

Category(ies): Keyboard.

Description: Disables the ON key interrupt handler and the key scan routine from setting the **OSOnBreak** flag.

Inputs: None

Outputs: None

Assumptions: None

Side Effects: None

Availability: All versions of the TI-89 / TI-92 Plus.

TI-89 / TI-92 Plus Differences: None

See Also: **OSOnBreak, OSCheckBreak, OSClearBreak, OSEnableBreak**

Example:

```
TRY
   OSDisableBreak();
   .
   .                    /* do some stuff without the ON key */
   .                    /* interrupt handler active */
ONERR
   .
   .                    /* handle an error if it occurs */
   .
ENDTRY
   OSEnableBreak();     /* restore ON key interrupt handler
```

OSEnableBreak

Declaration: void **OSEnableBreak** (void)

Category(ies): Keyboard

Description: Enables the ON key interrupt handler and the key scan routine to set the **OSOnBreak** flag.

Inputs: None

Outputs: None

Assumptions: None

Side Effects: None

Availability: All versions of the TI-89 / TI-92 Plus.

TI-89 / TI-92 Plus Differences: None

See Also: **OSOnBreak, OSCheckBreak, OSClearBreak, OSDisableBreak**

Example: See **OSDisableBreak**.

OSInitBetweenKeyDelay

Declaration: WORD **OSInitBetweenKeyDelay** (WORD *val*)

Category(ies): Keyboard

Description: For keys that auto-repeat, set the delay between pushing the first and second key. Use of this routine may destabilize key scans for other applications. Be sure to restore the original delay values.

Inputs: *val* — Number of system ticks to delay after pushing the first key, before pushing the next key value.

Outputs: Value that the key initial delay was previously set to.

Assumptions: None

Side Effects: None

Availability: All versions of the TI-89 / TI-92 Plus.

TI-89 / TI-92 Plus Differences: The system tick is based on the system clock, which may vary across platforms. If you must change these values, it is recommended that you first query the system to see what the delay is set at, then set the delay to some fraction or multiple of the system default value.

See Also: **OSInitKeyInitDelay, OSFastArrows**

Example:

```
WORD sysDelay;
/* Dummy call to get the system default value */
sysDelay = OSInitBetweenKeyDelay( 0 );

/* Set the delay to 4/3 the system's value */
OSInitBetweenKeyDelay ( sysDelay * 4 / 3 ) ;

// Your code goes here

/* Set the delay back to system default before swapping out or quitting the app */
OSInitBetweenKeyDelay( sysDelay );
```

OSInitKeyInitDelay

Declaration: WORD **OSInitKeyInitDelay** (WORD *val*)

Category(ies): Keyboard

Description: Sets the delay between the time that a key is pressed and the time that keypress is scanned. Use of this routine may destabilize key scans for other applications. Be sure to restore the original delay values.

Inputs: *val* — Number of system ticks to delay after pushing the first key, before pushing the next key value.

Outputs: Value that the key initial delay was previously set to.

Assumptions: None

Side Effects: None

Availability: All versions of the TI-89 / TI-92 Plus.

TI-89 / TI-92 Plus Differences: The system tick is based on the system clock, which may vary across platforms. If you must change these values, it is recommended that you first query the system to see what the delay is set at, then set the delay to some fraction or multiple of the system default value.

See Also: **OSBetweenKeyDelay, OSFastArrows**

Example:

```
WORD sysDelay;
/* Dummy call to get the system default value */
sysDelay = OSInitKeyInitDelay( 0 );

/* Set the delay to 4/3 the system's value */
OSInitKeyInitDelay ( sysDelay * 4 / 3 ) ;

// Your code goes here

/* Set the delay back to system default before swapping out or quitting the app */
OSInitKeyInitDelay( sysDelay );
```

push_getkey

Declaration: void **push_getkey** (void)

Category(ies): Keyboard

Description: Pushes the current key code onto the estack, zero if no key is currently held down.

Inputs: None

Outputs: For a listing of key codes, see **Appendix B** in the TI-92 Guidebook. Note that in the standard include file, tiams.h, some keys are conditionally defined differently on the TI-89 versus the TI-92 Plus. So, for example, KB_LEFT is defined to have the value 0x151 when _92 is defined whereas it has the value 0x152 if _89 is defined. The **push_getkey** routine will always return the value 0x151 which is the remapped TI-92 Plus value.

Assumptions: None

Side Effects: May expand expression stack, cause heap compression, or throw an error.

Availability: On AMS 1.05 and higher.

TI-89 / TI-92 Plus Differences: This routine remaps the TI-89 keys to match the TI-92 Plus. This is the TI-BASIC GetKey routine and not the low level keyboard routine. Some of the return values are different than those returned by **GKeyIn** or the event handler.

See Also: **pushkey, GKeyIn**

Example: This example routine waits for [ENTER] or [ESC] to be pressed. It returns TRUE if [ENTER] was pressed, FALSE if [ESC] was pressed.

```
Boolean tKey( void )
{  Access_AMS_Global_Variables;
   unsigned short key, result;

   do {
      push_getkey();
      result = estack_to_ushort(top_estack, &key)
      delete_expression( top_estack );
      if (1 == result) {
         if (KB_ENTER == key)
            return TRUE;
         if (KB_ESC == key)
            return FALSE;
      }
   } while (TRUE);
}
```

pushkey

Declaration: void **pushkey** (WORD *key*)

Category(ies): Keyboard

Description: Pushes a key code back onto the keyboard queue so that key will be the next key retrieved. Note that the **GKeyPush** macro is defined to be **pushkey**.

Inputs: *key* — Key code to push onto key queue.

Outputs: None

Assumptions: None

Side Effects: None

Availability: All versions of the TI-89 / TI-92 Plus.

TI-89 / TI-92 Plus Differences: None

See Also: **push_getkey, GKeyIn**

Example: See **QSysKey** (using **GKeyPush**).

QModeKey

Declaration: BOOL **QModeKey** (WORD *Key*)

Category(ies): Keyboard

Description: Return TRUE if *Key* is a mode key: [OFF], [APPS], [♦]-[APPS], [2nd]-[APPS], [MODE], [VAR-LINK], [MEM], [QUIT], or any of the built-in app keys: [HOME], [GRAPH], ...

Inputs: *Key* — Key value.

Outputs: TRUE or FALSE.

Assumptions: None

Side Effects: None

Availability: All versions of the TI-89 / TI-92 Plus.

TI-89 / TI-92 Plus Differences: On the TI-92 Plus, the built-in app keys are [♦]-Q, W, E, R, T, Y. On the TI-89 they are [♦]-[F1] . . . [F5] and [HOME].

See Also: **QSysKey**

Example:

```
/* Mode keys exit dialog boxes by using code like the following */
if (QModeKey(Key)) {
   GKeyPush(Key);  /* repush key on keyboard buffer */
   ER_throwVar( KB_ESC | 0x8000 ); /* exit edit buffer code and eventually dialog box */
}
```

QSysKey

Declaration: BOOL **QSysKey** (WORD *Key*)

Category(ies): Keyboard

Description: Return TRUE if *Key* is a system key: MATH, CATALOG, CHAR, CUSTOM.

Inputs: *Key* — Key value.

Outputs: TRUE or FALSE.

Assumptions: None

Side Effects: None

Availability: All versions of the TI-89 / TI-92 Plus.

TI-89 / TI-92 Plus Differences: None

See Also: **QModeKey**

Example: The menu system uses **GKeyIn** to get keys. It sets the GKF_MODAL and GKF_SYS keys which cause **GKeyIn** to check for system or mode keys and repush those keys onto the keyboard buffer and return KB_ESC. This causes system and mode keys to exit menus and be acted on after the menu is closed.

```
Key = GKeyIn( NULL, GKF_MODAL | GKF_SYS ); /* in menu code */
.
.
.
/* In the GKeyIn code, QSysKey and QModeKey are used to check for system and mode keys */
.
.
.
if ( ((Flags & GKF_SYS) && QSysKey(Key)) || ((Flags & GKF_MODAL) && QModeKey(Key)) )
{
   GKeyPush( Key );
   Key = KB_ESC;
}
```

restoreAlphaLock

Declaration: void **restoreAlphaLock** (FLAGS8 * *saveAlpha*)

Category(ies): Keyboard

Description: Restores the alpha-lock status to the value saved in *saveAlpha*.

Inputs: *saveAlpha* — Pointer to FLAGS8 structure.

Outputs: None

Assumptions: *saveAlpha* was previously set by **alphaLockOn** or **alphaLockOff**.

Side Effects: None

Availability: On AMS 2.00 and higher.

TI-89 / TI-92 Plus Differences: For compatibility reasons, this routine is available on the TI-92 Plus but does nothing on that platform.

See Also: **alphaLockOn, alphaLockOff**

Example:

```
FLAGS8 saveStat;
alphaLockOff( &saveStat );
editField();  /* user may be entering NUMERIC data here */
restoreAlphaLock( &saveStat );
```

Appendix A: System Routines — Link

See Also:

BatTooLowFlash

Declaration: BOOL **BatTooLowFlash** (WORD *delayCount*)

Category(ies): Link, Flash Memory, Status Line

Description: Returns TRUE if battery level is (or ever was) too low to write to Flash. If *delayCount* is non-zero, and the USER timer is not already in use, a delay of *delayCount* will occur before a reading is done.

Inputs: *delayCount* — Length of delay before battery reading is taken (recommended value is 50 * MS100).

Outputs: Returns TRUE if battery level is (or ever was) too low to write to Flash. Otherwise returns FALSE.

Assumptions: USER timer is not already in use.

Side Effects: None

Availability: On AMS 2.00 and higher.

TI-89 / TI-92 Plus Differences: None

See Also: None

Example:

```
if (BatTooLowFlash(50*MS100))   /* batteries too low to send product code */
      ERD_dialog( ER_BATT_LOW, FALSE );
```

LIO_RecvData

Declaration: WORD **LIO_RecvData** (BYTE * *data*, DWORD *count*, DWORD *timeout*)

Category(ies): Link

Description: Receive data from the link port.

Inputs:

data — Pointer to buffer where data will be stored.

count — Number of bytes to receive.

timeout — If 0, receive until all data is sent or an error occurs. If nonzero, quit waiting after a delay of *timeout*.

Outputs: Returns non-0 if an error occurs. Otherwise returns 0.

Assumptions: None

Side Effects: None

Availability: All versions of the TI-89 / TI-92 Plus.

TI-89 / TI-92 Plus Differences: None

See Also: **LIO_SendData**

Example:

```
if (error = LIO_RecvData(msg, len+2, ONE_SECOND * 2))
      return error;
```

LIO_SendData

Declaration: WORD **LIO_SendData** (BYTE * *data*, DWORD *count*)

Category(ies): Link

Description: Send data over the link port.

Inputs: *data* — Pointer to data buffer.

count — Number of bytes to send.

Outputs: Returns non-0 if an error occurs. Otherwise returns 0.

Assumptions: None

Side Effects: None

Availability: All versions of the TI-89 / TI-92 Plus.

TI-89 / TI-92 Plus Differences: None

See Also: **LIO_RecvData**

Example:

```
rc = LIO_SendData(frame, frameLen + 2);
if (rc)
   ER_throwVar(rc);
```

OSCheckLinkOpen

Declaration: WORD **OSCheckLinkOpen** (void)

Category(ies): Link

Description: Determines whether the link port is open.

Inputs: None

Outputs: Returns TRUE if the link port is open. Otherwise returns FALSE.

Assumptions: None

Side Effects: None

Availability: On AMS 2.05 and higher.

TI-89 / TI-92 Plus Differences: None

See Also: **OSLinkOpen**

Example:

OSLinkClose

Declaration: void **OSLinkClose** (void)

Category(ies): Link

Description: Closes the link port if it is open.

Inputs: None

Outputs: None

Assumptions: None

Side Effects: None

Availability: All versions of the TI-89 / TI-92 Plus.

TI-89 / TI-92 Plus Differences: None

See Also: **OSLinkOpen**

Example:

OSLinkOpen

Declaration: void **OSLinkOpen** (void)

Category(ies): Link

Description: Opens the link port, flushes the link queue, and sets the OSLinkOpenFlag to TRUE.

Inputs: None

Outputs: None

Assumptions: This function does not check whether the link port is closed. You should always call **OSCheckLinkOpen** prior to calling this function.

Side Effects: None

Availability: All versions of the TI-89 / TI-92 Plus.

TI-89 / TI-92 Plus Differences: None

See Also: **OSCheckLinkOpen, OSLinkClose**

Example:

OSLinkReset

Declaration: void **OSLinkReset** (void)

Category(ies): Link

Description: Closes the link if it is open, flushes the link queue, attempts to reset the link hardware, and asserts a DBus error. This function should only be called in the event of an error.

Inputs: None

Outputs: None

Assumptions: None

Side Effects: None

Availability: All versions of the TI-89 / TI-92 Plus.

TI-89 / TI-92 Plus Differences: None

See Also: **OSCheckLinkOpen**

Example:

Appendix A: System Routines — Lists and Matrices

See Also:

all_tail

Declaration: Boolean **all_tail** (Boolean (* *bool_fun*) (EStackIndex), EStackIndex *i*)

Category(ies): Lists and Matrices, Math

Description: Determines whether the *bool_fun* returns TRUE for each expression in the tail of expressions indexed by *i*.

Inputs:
bool_fun — A Boolean function that takes the index of an expression as its one argument.

i — Index of a tail.

Outputs: Returns TRUE if *bool_fun* returns TRUE for each expression in the tail of expressions indexed by *i*. Otherwise returns FALSE.

Assumptions: None

Side Effects: None

Availability: On AMS 1.05 and higher.

TI-89 / TI-92 Plus Differences: None

See Also: **any_tail, map_tail, did_map_aggregate_arg**

Example:

```
Boolean is_constant (EStackIndex i)
/* Returns TRUE if the expression indexed by i is free of all variables.
   See also the slightly different function likely_approx_to_number.
   Otherwise returns FALSE.
*/
{  for (;;)
         if (IS_NUM_VAR_OR_ZERO_ARG_TAG (ESTACK (i)))
            return (! IS_VARIABLE_TAG (ESTACK (i)));
         else if (IS_ONE_ARG_TAG (ESTACK (i)))
               --i;                          /* Loop to check the one argument */
         else if (IS_TWO_ARG_TAG (ESTACK (i)))
               if (is_constant (--i))
                  i=next_expression_index(i);  /* Loop to check 2nd arg */
               else return FALSE;
         else
            {  if (USER_FUN_TAG == ESTACK (i--))
                  i = next_expression_index (i);     /* Skip function name */
               return all_tail (is_constant, i);
            }
}
```

any_tail

Declaration: Boolean **any_tail** (Boolean (* *bool_fun*) (EStackIndex), EStackIndex *i*)

Category(ies): Lists and Matrices, Math

Description: Determines whether the *bool_fun* returns TRUE for at least one expression in the tail of expressions indexed by *i*.

Inputs:

bool_fun — A Boolean function that takes the index of an expression as its one argument.

i — Index of a tail.

Outputs: Returns TRUE if *bool_fun* returns TRUE for at least one expression in the tail of expressions indexed by *i*. Otherwise returns FALSE.

Assumptions: None

Side Effects: None

Availability: On AMS 1.05 and higher.

TI-89 / TI-92 Plus Differences: None

See Also: **all_tail, map_tail, did_map_aggregate_arg**

Example:

```
Boolean has_inf_mag (EStackIndex i)
/* Returns TRUE if i indexes a real or complex infinity. Otherwise returns FALSE. */
{  EStackIndex j;
   return IS_INF_MAG_TAG (ESTACK (i)) ||
         IM_RE_TAG == ESTACK (i) &&
            (IS_INF_MAG_TAG (ESTACK (i-1u)) &&
            IS_INF_MAG_TAG (ESTACK (next_expression_index (i - 1u))) ) ||
         LIST_TAG == ESTACK (i) && any_tail (has_inf_mag, i - 1u);
}
```

cmd_sorta

Declaration: void **cmd_sorta** (EStackIndex *lst_name*)

Category(ies): Lists and Matrices, Math, Variables

Description: Sorts in ascending order the list named by *lst_name* and stores the result back in the same list.

Inputs: *lst_name* — Indexes the name of a list to sort.

Outputs: None

Assumptions: None

Side Effects: May expand expression stack, cause heap compression, or throw an error.

Availability: On AMS 1.05 and higher.

TI-89 / TI-92 Plus Differences: None

See Also: **cmd_sortd**

Example: This example creates a list in L4. Note that this by-passes the normal routines to tokenize symbol names (**TokenizeName** or **TokenizeSymName**) which should normally be used. The list is then sorted both in ascending and then descending order.

```
EStackIndex vName;

push_quantum( END_TAG );  /* executing commands requires this */
push_zstr( "l4" );   /* pushes STR_DATA_TAG also */
vName = top_estack - 1;   /* names do not need STR_DATA_TAG so skip it */
push_quantum(END_TAG); push_Float(2.0);
push_Float(-5.1); push_Float(8.0);
push_quantum(LIST_TAG);
VarStore( vName, STOF_ESI, 0, top_estack );

cmd_sorta( vName );
cmd_sortd( vName );
```

cmd_sortd

Declaration: void **cmd_sortd** (EStackIndex *lst_name*)

Category(ies): Lists and Matrices, Math, Variables

Description: Sorts in descending order the list named by *lst_name* and stores the result back in the same list.

Inputs: *lst_name* — Indexes the name of a list to sort.

Outputs: None

Assumptions: None

Side Effects: May expand expression stack, cause heap compression, or throw an error.

Availability: On AMS 1.05 and higher.

TI-89 / TI-92 Plus Differences: None

See Also: **cmd_sorta**

Example: See **cmd_sorta**.

did_map_aggregate_arg

Declaration: Boolean **did_map_aggregate_arg**
(void (* *proc*) (EStackIndex, EStackIndex), EStackIndex *i*, EStackIndex *j*)

Category(ies): Lists and Matrices

Description: Determines if *i* and/or *j* index aggregates.

A NON_CONFORMING_LISTS_ERROR is thrown if *i* and *j* index nonconforming aggregates.

Inputs:
proc — Address of a procedure that takes two EStackIndex arguments.

i, *j* — Indices of the top tag of internally-simplified algebraic expressions.

Outputs: If *i* and/or *j* index aggregates, maps *proc* over their elements then returns TRUE. Otherwise returns FALSE.

Assumptions: None

Side Effects: None

Availability: On AMS 2.04 and higher.

TI-89 / TI-92 Plus Differences: None

See Also: **map_tail, all_tail, any_tail**

Example:

```
/* Title: push_dot_mult
** Description: push element by element product of two arguments
**              onto the estack
** Input: estack index i of the first simplified argument
**        estack index j of the second simplified argument
** Output: pushes the element by element product onto the estack
*/
void push_dot_mult (EStackIndex i, EStackIndex j)
{  if (! did_map_aggregate_arg (push_dot_mult, i, j))
         push_product (i, j);
}
```

is_matrix

Declaration: Boolean **is_matrix** (EStackIndex *i*)

Category(ies): Lists and Matrices

Description: Determines whether the expression indexed by *i* is a matrix.

Inputs: *i* — Index of the top tag of an internally-simplified expression.

Outputs: Returns TRUE if the expression indexed by *i* is a matrix. Otherwise returns FALSE.

Assumptions: None

Side Effects: None

Availability: On AMS 1.05 and higher.

TI-89 / TI-92 Plus Differences: None

See Also: **is_square_matrix**

Example:

```
void push_colnorm (EStackIndex matrix_idx)
   /* Pushes the largest of the sums of the absolute values of the
      elements in each column of the matrix indexed by matrix_idx.
*/
{  Access_AMS_Global_Variables;
   EStackIndex i, j, old_top = top_estack;
   if (is_matrix (matrix_idx))
   {  if (! can_be_approxed (matrix_idx, TRUE))
            ER_THROW (ER_DOMAIN);
      i = matrix_idx - 1u;
      push0 ();
      while (END_TAG != ESTACK (i))
      {  j = top_estack;
         push_abs (i);
         replace_top2_with_sum (j);
         i = next_expression_index (i);
      }
      j = top_estack;
      push_max1 (j);
      delete_between (old_top, j);
   }
   else  ER_throw( ER_DATATYPE );
}
```

is_square_matrix

Declaration: Boolean **is_square_matrix** (EStackIndex *i*)

Category(ies): Lists and Matrices

Description: Determines whether the expression indexed by *i* is a square matrix.

Inputs: *i* — Index of the top tag of an internally-simplified expression.

Outputs: Returns TRUE if the expression indexed by *i* is a square matrix. Otherwise returns FALSE.

Assumptions: None

Side Effects: None

Availability: On AMS 1.05 and higher.

TI-89 / TI-92 Plus Differences: None

See Also: **is_matrix**

Example:

```
push_Float (3.7);
is_square_matrix (top_estack);  /* Returns FALSE */
```

last_element_index

Declaration: EStackIndex **last_element_index** (EStackIndex *i*)

Category(ies): Lists and Matrices

Description: Returns an index to the last element of the expression indexed by *i*.

Inputs: *i* — Index of the top tag of a tail of expressions.

Outputs: Returns *i* if *i* indexes an END_TAG. Otherwise returns the index of the last element of a nonempty tail indexed by *i*.

Assumptions: None

Side Effects: None

Availability: On AMS 1.05 and higher.

TI-89 / TI-92 Plus Differences: None

See Also: **remaining_element_count**

Example:

```
push_quantum (END_TAG);
last_element_index (top_estack);            /* Returns top_estack */
push_Float (3.7);
foo = top_estack;
push_Float (5.4);
last_element_index (top_estack);            /* Returns foo */
```

map_tail

Declaration: void **map_tail** (void (* *proc*) (EStackIndex), EStackIndex *i*)

Category(ies): Lists and Matrices, Math

Description: Pushes an END_TAG onto the estack. Applies *proc* to each of the successive expressions in the tail indexed by *i*, beginning with the deepest.

If *proc* does not push something onto the estack, the END_TAG pushed by **map_tail** will probably have to be deleted.

Inputs: *proc* — Address of a procedure that takes one EStackIndex argument.

i — Index of the top tag of an internally-simplified tail of expressions.

Outputs: None

Assumptions: None

Side Effects: None

Availability: On AMS 1.05 and higher.

TI-89 / TI-92 Plus Differences: None

See Also: **all_tail, any_tail, did_map_aggregate_arg**

Example:

```
void push_ord (EStackIndex i)
{  if (STR_DATA_TAG == ESTACK(i))           /* if arg is a string */
      push_quantum_as_nonnegative_int
                                 (ESTACK (next_expression_index(i) + 2u));
   else if (LIST_TAG == ESTACK(i))
   {  map_tail (push_ord, i-1);
      push_quantum (LIST_TAG);
   }
   else
      ER_THROW (ER_ARG_MUST_BE_STRING);
}
```

push_augment

Declaration: void **push_augment** (EStackIndex *i*, EStackIndex *j*)

Category(ies): Lists and Matrices

Description: Performs column augmentation of two lists or two matrices.

Inputs: *i, j* — EStackIndexes of two lists or two matrices.

Outputs: Pushes onto the expression stack the result of appending the second argument onto the first argument as new columns.

Assumptions: Either the arguments are both lists or they are both matrices with the same number of rows.

Side Effects: May expand expression stack, cause heap compression, or throw an error.

Availability: On AMS 2.00 and higher.

TI-89 / TI-92 Plus Differences: None

See Also: None

Example:

```
/* If i is the EStackIndex of the 2x2 matrix [1, 2; 3, 4], and
   j is the EStackIndex of the 2x1 matrix [5; 6], then */

push_augment (i, j);

/* pushes onto the expression stack the matrix [1, 2, 5; 3, 4, 6]. */
```

push_coldim

Declaration: void **push_coldim** (EStackIndex *mat_idx*)

Category(ies): Lists and Matrices, Math

Description: Pushes onto the estack the column dimension of the matrix indexed by *mat_idx*.

Inputs: *mat_idx* — Indexes the input matrix.

Outputs: None

Assumptions: None

Side Effects: May expand expression stack, cause heap compression, or throw an error.

Availability: On AMS 2.00 and higher.

TI-89 / TI-92 Plus Differences: None

See Also: **push_rowdim**

Example:

```
EStackIndex mat_idx;
.
.
.
push_coldim( mat_idx );
```

push_colnorm

Declaration: void **push_colnorm** (EStackIndex *mat_idx*)

Category(ies): Lists and Matrices, Math

Description: Pushes onto the estack the column norm of the matrix indexed by *mat_idx*. The column norm is the largest value of the sums of the absolute values of the elements in each column of the matrix.

Inputs: *mat_idx* — Indexes the input matrix.

Outputs: None

Assumptions: None

Side Effects: May expand expression stack, cause heap compression, or throw an error.

Availability: On AMS 2.00 and higher.

TI-89 / TI-92 Plus Differences: None

See Also: **push_rownorm, push_matnorm**

Example:

```
EStackIndex mIndx;

/* A matrix is just a list of lists */
push_quantum(END_TAG);
push_quantum(END_TAG); push_Float(-12); push_Float(-24); push_quantum(LIST_TAG);
push_quantum(END_TAG); push_Float(7); push_Float(5); push_quantum(LIST_TAG);
push_quantum(LIST_TAG);
mIndx = top_estack;  /* mIndx points to [[5, 7]] [-24, -12]] */

/* The column norm (29 == abs(-24)+abs(5)) will be pushed onto the estack */
push_colnorm( mIndx );

/* The row norm (36 == abs(-24)+abs(-12)) will be pushed onto the estack */
push_rownorm( mIndx );

/* Push the matrix norm (28.178) */
push_matnorm( mIndx );

/* Push the determinant (108) */
push_determinant( mIndx, NULL );
```

push_cross_product

Declaration: void **push_cross_product** (EStackIndex *i*, EStackIndex *j*)

Category(ies): Lists and Matrices, Math

Description: Computes the cross product of two vectors.

Inputs: *i, j* — EStackIndexes of two vectors represented in one of three ways: both as lists, both as single row matrices, or both as single column matrices. Each must have either two elements or three elements.

Outputs: Pushes onto the expression stack the cross product in the same form as the input. List input generates list output. Row matrix input generates row matrix output. Column matrix input generates column matrix output.

Assumptions: Arguments must be in internal-tokenized form. External-tokenized form may cause an error throw.

Side Effects: May expand expression stack, cause heap compression, or throw an error.

Availability: On AMS 2.00 and higher.

TI-89 / TI-92 Plus Differences: None

See Also: None

Example:

```
/* If i is the EStackIndex of the internal-tokenized form of {1, a, b},
   and j is the EStackIndex of the internal-tokenized form of {c, 2, 3}, then */

push_cross_product (i, j);

/* pushes the internal-tokenized form of {3*a-2*b, b*c-3, 2-a*c} onto the
   expression stack. */
```

push_cumsum

Declaration: void **push_cumsum** (EStackIndex *mat_list_indx*)

Category(ies): Lists and Matrices, Math

Description: Pushes onto the estack the cumulative sum of the matrix or list indexed by *mat_list_indx*. For a list the result is a list with each element being the running sum of all of the elements. For a matrix the result is a matrix with each column being the running sum of all of the elements in that column.

Inputs: *mat_list_indx* — Indexes the input list or matrix.

Outputs: None

Assumptions: None

Side Effects: May expand expression stack, cause heap compression, or throw an error.

Availability: On AMS 2.00 and higher.

TI-89 / TI-92 Plus Differences: None

See Also: **push_sumlist**

Example:

```
EStackIndex lIndx;
BYTE lName[] = {0,'c','s','u','m','1',0};

/* push test list */
push_quantum(END_TAG);
push_Float(1.5);
push_Float(2.5);
push1();
push_quantum(LIST_TAG);
lIndx = top_estack;

/* calculate the cumulative sum: {1, 3.5, 5.0} */
push_cumsum( lIndx );

/* Store to CSUM1 */
VarStore( lName+6, STOF_ESI, 0, top_estack );

/* Now push the sum of all of the elements in the list: 5.0 */
push_sumlist( lIndx );
```

push_determinant

Declaration: void **push_determinant** (EStackIndex *mat_idx*, EStackIndex *tol_idx*)

Category(ies): Lists and Matrices, Math

Description: Pushes onto the estack the determinant of the matrix indexed by *mat_idx*.

Inputs:

mat_idx — Indexes the input matrix.

tol_idx — If not NULL then indexes a tolerance factor. Any matrix element is treated as zero if its absolute value is less than the tolerance.

Outputs: None

Assumptions: None

Side Effects: May expand expression stack, cause heap compression, or throw an error.

Availability: On AMS 2.00 and higher.

TI-89 / TI-92 Plus Differences: None

See Also: None

Example: See **push_colnorm**.

push_diag

Declaration: void **push_diag** (EStackIndex *mat_lst_idx*)

Category(ies): Lists and Matrices, Math

Description: For a list, row vector, or column vector pushes onto the estack a square matrix with the elements of the list or vector on the diagonal and zeroes elsewhere. For a matrix pushes onto the estack a row vector containing the diagonal of the matrix.

Inputs: *mat_lst_idx* — Indexes the input list or matrix.

Outputs: None

Assumptions: None

Side Effects: May expand expression stack, cause heap compression, or throw an error.

Availability: All versions of the TI-89 / TI-92 Plus.

TI-89 / TI-92 Plus Differences: None

See Also:

Example: Assuming the top of the estack contains a conforming matrix, this example computes **diag** (**top_estack**) ∗ eigvc (mat_idx).

```
EStackIndex vl, mat_idx;
.
.
.
push_diag ( top_estack );
vl = top_estack;
push_eigvc ( mat_idx );
push_product ( vl, top_estack );
```

push_dimension

Declaration: void **push_dimension** (EStackIndex *mat_lst_str_idx*)

Category(ies): Lists and Matrices, Math

Description: For a list or string pushes on the estack the dimension as a number. For a matrix pushes the dimensions as a two element list.

Inputs: *mat_lst_str_idx* — Indexes the input list, matrix, or string.

Outputs: None

Assumptions: None

Side Effects: May expand expression stack, cause heap compression, or throw an error.

Availability: On AMS 2.00 and higher.

TI-89 / TI-92 Plus Differences: None

See Also:

Example: This example pushes the dimension (three) of a string on the estack.

```
BYTE str[] =  {0,'1','2','3',0,STR_DATA_TAG};

push_dimension( str+5 );
```

push_dot_add

Declaration: void **push_dot_add** (EStackIndex *i*, EStackIndex *j*)

Category(ies): Lists and Matrices, Math

Description: Pushes the element by element sum of two internal tokenized arguments onto the estack

Inputs: *i* — EStackIndex of the first internal tokenized argument.

j — EStackIndex of the second internal tokenized argument.

Outputs: Pushes the internal tokenized element by element sum of the two arguments onto the estack.

Assumptions: The arguments must have the same dimensions. They must be the same length lists, or the same size matrices, or both be scalars.

Side Effects: May cause estack expansion, heap compression, or throw an error if the arguments are invalid.

Availability: On AMS 2.00 and higher.

TI-89 / TI-92 Plus Differences: None

See Also: **push_dot_div, push_dot_mult, push_dot_sub**

Example:

If m indexes the bolded tag in the list {a, 2, 0} as follows
END_TAG 0 NONNEGATIVE_INTEGER_TAG 2 1 NONNEGATIVE_INTEGER_TAG A_VAR_TAG **LIST_TAG**

and n indexes the bolded tag in the list {1, 1, 1} as follows
END_TAG 1 1 NONNEGATIVE_INTEGER_TAG 1 1 NONNEGATIVE_INTEGER_TAG 1 1 NONNEGATIVE_INTEGER_TAG **LIST_TAG**

then

```
push_dot_add (m, n);
```

pushes the list {a + 1, 3, 1} onto the estack such that **top_estack** points to the bolded tag as follows.
END_TAG 1 1 NONNEGATIVE_INTEGER_TAG 3 1 NONNEGATIVE_INTEGER_TAG 1 1 NONNEGATIVE_INTEGER_TAG A_VAR_TAG ADD_TAG **LIST_TAG**

push_dot_div

Declaration: void **push_dot_div** (EStackIndex *i*, EStackIndex *j*)

Category(ies): Lists and Matrices, Math

Description: Pushes the element by element ratio of two internal tokenized arguments onto the estack.

Inputs: *i* — EStackIndex of the internal tokenized numerator.

j — EStackIndex of the internal tokenized denominator.

Outputs: Pushes the internal tokenized element by element ratio of the two arguments onto the estack.

Assumptions: The arguments must have the same dimensions. They must be the same length lists, or the same size matrices, or both be scalars.

Side Effects: May cause estack expansion, heap compression, or throw an error if the arguments are invalid.

Availability: On AMS 2.00 and higher.

TI-89 / TI-92 Plus Differences: None

See Also: **push_dot_add, push_dot_mult, push_dot_sub**

Example:

If m indexes the bolded tag in the list {a, 2, 0} as follows
END_TAG 0 NONNEGATIVE_INTEGER_TAG 2 1 NONNEGATIVE_INTEGER_TAG A_VAR_TAG **LIST_TAG**

and n indexes the bolded tag in the list {3, 3, 3} as follows
END_TAG 3 1 NONNEGATIVE_INTEGER_TAG 3 1 NONNEGATIVE_INTEGER_TAG 3 1 NONNEGATIVE_INTEGER_TAG **LIST_TAG**

then

```
push_dot_div (m, n);
```

pushes the list {a * 1/3, 2/3, 0} onto the estack such that **top_estack** points to the bolded tag as follows.
END_TAG 0 NONNEGATIVE_INTEGER_TAG 3 1 2 1 POSITIVE_FRACTION_TAG 3 1 1 1 POSITIVE_FRACTION_TAG A_VAR_TAG MULTIPLY_TAG **LIST_TAG**

push_dot_mult

Declaration: void **push_dot_mult** (EStackIndex *i*, EStackIndex *j*)

Category(ies): Lists and Matrices, Math

Description: Pushes the element by element product of two internal tokenized arguments onto the estack.

Inputs: *i* — EStackIndex of the first internal tokenized argument.

j — EStackIndex of the second internal tokenized argument.

Outputs: Pushes the internal tokenized element by element product of the two arguments onto the estack.

Assumptions: The arguments must have the same dimensions. They must be the same length lists, or the same size matrices, or both be scalars.

Side Effects: May cause estack expansion, heap compression, or throw an error if the arguments are invalid.

Availability: On AMS 2.00 and higher.

TI-89 / TI-92 Plus Differences: None

See Also: **push_dot_add, push_dot_div, push_dot_sub**

Example:

If m indexes the bolded tag in the list {a, 2, 0} as follows
END_TAG 0 NONNEGATIVE_INTEGER_TAG 2 1 NONNEGATIVE_INTEGER_TAG A_VAR_TAG **LIST_TAG**

and n indexes the bolded tag in the list {3, 3, 3} as follows
END_TAG 3 1 NONNEGATIVE_INTEGER_TAG 3 1 NONNEGATIVE_INTEGER_TAG 3 1 NONNEGATIVE_INTEGER_TAG **LIST_TAG**

then

```
push_dot_mult (m, n);
```

pushes the list {a * 3, 6, 0} onto the estack such that **top_estack** points to the bolded tag as follows.
END_TAG 0 NONNEGATIVE_INTEGER_TAG 6 1 NONNEGATIVE_INTEGER_TAG 3 1 NONNEGATIVE_INTEGER_TAG A_VAR_TAG MULTIPLY_TAG **LIST_TAG**

push_dot_sub

Declaration: void **push_dot_sub** (EStackIndex *i*, EStackIndex *j*)

Category(ies): Lists and Matrices, Math

Description: Pushes the element by element difference of two internal tokenized arguments onto the estack.

Inputs: *i* — EStackIndex of the first internal tokenized argument.

j — EStackIndex of the second internal tokenized argument.

Outputs: Pushes the internal tokenized element by element difference of the two arguments onto the estack.

Assumptions: The arguments must have the same dimensions. They must be the same length lists, or the same size matrices, or both be scalars.

Side Effects: May cause estack expansion, heap compression, or throw an error if the arguments are invalid.

Availability: On AMS 2.00 and higher.

TI-89 / TI-92 Plus Differences: None

See Also: **push_dot_add, push_dot_div, push_dot_mult**

Example:

If m indexes the bolded tag in the list {a, 2, 0} as follows
END_TAG 0 NONNEGATIVE_INTEGER_TAG 2 1 NONNEGATIVE_INTEGER_TAG A_VAR_TAG **LIST_TAG**

and n indexes the bolded tag in the list {1, 1, 1} as follows
END_TAG 1 1 NONNEGATIVE_INTEGER_TAG 1 1 NONNEGATIVE_INTEGER_TAG 1 1 NONNEGATIVE_INTEGER_TAG **LIST_TAG**

then

```
push_dot_sub (m, n);
```

pushes the list {a + (-1), 1, -1} onto the estack such that **top_estack** points to the bolded tag as follows.
END_TAG 1 1 NEGATIVE_INTEGER_TAG 1 1 NONNEGATIVE_INTEGER_TAG 1 1 NEGATIVE_INTEGER_TAG A_VAR_TAG ADD_TAG **LIST_TAG**

push_dotproduct

Declaration: void **push_dotproduct** (EStackIndex *i*, EStackIndex *j*)

Category(ies): Lists and Matrices, Math

Description: Computes the dot product of two vectors.

Inputs: *i, j* — EStackIndexes of two vectors are represented in one of three ways: both as lists, both as single row matrices, or both as single column matrices.

Outputs: Pushes the dot product of the two vectors onto the expression stack.

Assumptions: Arguments must be in internal-tokenized form. External-tokenized form may cause an error throw.

Side Effects: May expand expression stack, cause heap compression, or throw an error.

Availability: On AMS 2.00 and higher.

TI-89 / TI-92 Plus Differences: None

See Also: None

Example:

```
/* If i is the EStackIndex of the internal-tokenized form of {1, a, b}, and j is the
   EStackIndex of the internal-tokenized form of {c, 2, 3}, then */

push_dotproduct (i, j);

/* Pushes the internal-tokenized form of 2 * a + 3 * b + c onto the expression stack.
*/
```

push_eigvc

Declaration: void **push_eigvc** (EStackIndex *mat_idx*)

Category(ies): Lists and Matrices, Math

Description: Returns a matrix containing the eigen vectors for a real or complex square matrix on the estack. This is the TI-BASIC eigVc command.

Inputs: *mat_idx* — Indexes the input square matrix.

Outputs: None

Assumptions: None

Side Effects: May expand expression stack, cause heap compression, or throw an error.

Availability: On AMS 2.00 and higher.

TI-89 / TI-92 Plus Differences: None

See Also: **push_eigvl**

Example: See **push_diag**.

push_eigvl

Declaration: void **push_eigvl** (EStackIndex *mat_idx*)

Category(ies): Lists and Matrices, Math

Description: Returns the eigen values for a real or complex square matrix on the estack. This is the TI-BASIC eigVl command.

Inputs: *mat_idx* — Indexes the input square matrix.

Outputs: None

Assumptions: None

Side Effects: May expand expression stack, cause heap compression, or throw an error.

Availability: On AMS 2.00 and higher.

TI-89 / TI-92 Plus Differences: None

See Also: **push_eigvc**

Example: This example is essentially the eigVl example from the TI-89 guidebook only written as an app. Note that it forces the current complex mode to rectangular. The output rather than being displayed in the Home screen is stored in the variable LST3. If we remove the **replace_top_with_post_simplified** then the result will not be in displayable or storable format.

```
Access_AMS_Global_Variables;
BYTE lst3[] =  {0,'l','s','t','3',0};
// [-1,2,5; 3,-6,9; 2,-5,7]
const Quantum tArray[] =    {END_TAG,
   END_TAG,7,1,NONNEGATIVE_INTEGER_TAG, 5,1,NEGATIVE_INTEGER_TAG,
      2,1,NONNEGATIVE_INTEGER_TAG, LIST_TAG,
   END_TAG,9,1,NONNEGATIVE_INTEGER_TAG, 6,1,NEGATIVE_INTEGER_TAG,
      3,1,NONNEGATIVE_INTEGER_TAG, LIST_TAG,
   END_TAG,5,1,NONNEGATIVE_INTEGER_TAG, 2,1,NONNEGATIVE_INTEGER_TAG,
      1,1,NEGATIVE_INTEGER_TAG, LIST_TAG,
   LIST_TAG
};
#define tArrayIndex ((EStackIndex) (tArray+sizeof(tArray)-1))

MO_currentOptions();
MO_option[MO_OPT_COMPLEX_FORMAT] = MO_VECT_RECT;
MO_digestOptions(H_NULL);
TRY
   push_eigvl( tArrayIndex );
   // output may have been in internal format
   replace_top_with_post_simplified(top_estack);
   VarStore( lst3+5, STOF_ESI, 0, top_estack );
ONERR
   ERD_dialog( errCode, FALSE );
ENDTRY
```

push_identity_mat

Declaration: void **push_identity_mat** (EStackIndex *size*)

Category(ies): Lists and Matrices, Math

Description: Pushes onto the estack the identity matrix using the expression indexed by *size* to specify the size.

Inputs: *size* — Indexes the expression specifying the size of the identity matrix to push on the estack.

Outputs: None

Assumptions: The input expression must be a positive integer.

Side Effects: None

Availability: On AMS 2.00 and higher.

TI-89 / TI-92 Plus Differences: None

See Also: None

Example: This example augments a matrix with the identity matrix.

```
EStackIndex mat;
.
.
.
/* Assume matrix already on estack */
mat = top_estack;
push_rowdim( mat );
push_identity_mat ( top_estack );
push_augment( mat, top_estack );
```

push_list_to_mat

Declaration: void **push_list_to_mat** (EStackIndex *list_idx,* EStackIndex *el_per_row_idx*)

Category(ies): Lists and Matrices

Description: Converts the list indexed by *list_idx* to a matrix with the elements per row as specified by the value indexed by *el_per_row_idx* and pushes the result on the estack.

Inputs:

list_idx — Indexes the input list.

el_per_row_idx — Indexes the number of elements per row for the resulting matrix. If NULL then the number of elements in the list is used (a row matrix is created).

Outputs: None

Assumptions: None

Side Effects: May expand expression stack, cause heap compression, or throw an error.

Availability: On AMS 2.00 and higher.

TI-89 / TI-92 Plus Differences: None

See Also: **push_mat_to_list**

Example:

```
/* Push a row matrix on the estack from a list indexed by list_index.
   Assume list_index already setup, make sure it is a list and not a matrix.
*/
if ((ESTACK(list_index) == LIST_TAG) && (ESTACK(list_index-1) != LIST_TAG)) {
   push_list_to_mat (list_index, NULL);
   /* Now push a column matrix on the estack */
   push_list_to_mat (list_index, Integer1Index);
}
```

push_mat_to_list

Declaration: void **push_mat_to_list** (EStackIndex *mat_idx*)

Category(ies): Lists and Matrices

Description: Convert the matrix indexed by *mat_idx* to a list and push the result on the estack.

Inputs: *mat_idx* — Indexes the input matrix.

Outputs: None

Assumptions: None

Side Effects: May expand expression stack, cause heap compression, or throw an error.

Availability: On AMS 2.00 and higher.

TI-89 / TI-92 Plus Differences: None

See Also: **push_list_to_mat**

Example: This example code segment takes a list indexed by list_index. If list_index indexes a row or column matrix, then **push_mat_to_list** is used to create a list. In this way, the code that follows can operate on a list or a vector (row or column matrix).

```
EStackIndex list_index;
unsigned short nrows, ncols;
.
.
.
if (LIST_TAG != ESTACK(list_index))
   ER_throw( ER_DATATYPE );
nrows = ncols = 0;
if (ESTACK(list_index - 1u) == LIST_TAG)  {
   /* input is a matrix/vector */
   nrows = remaining_element_count (list_index - 1u);
   ncols = remaining_element_count (list_index - 2u);
   /* see if it is a vector */
   if ( (1 == nrows) || (1 == ncols) ) {
      push_mat_to_list (list_index);
      list_index = top_estack;
   }  else
      ER_throw( ER_DATATYPE );
}
.
.
.
```

push_matnorm

Declaration: void **push_matnorm** (EStackIndex *mat_idx*)

Category(ies): Lists and Matrices, Math

Description: Pushes onto the estack the norm of a matrix indexed by *mat_idx*. This is the square root of the sum of the square of each matrix element.

Inputs: *mat_idx* — Indexes the input matrix.

Outputs: None

Assumptions: None

Side Effects: May expand expression stack, cause heap compression, or throw an error.

Availability: On AMS 2.00 and higher.

TI-89 / TI-92 Plus Differences: None

See Also: **push_rownorm, push_colnorm**

Example: See **push_colnorm**.

push_mean

Declaration: void **push_mean** (EStackIndex *mat_lst_idx*)

Category(ies): Lists and Matrices, Statistics, Math

Description: Pushes onto the estack the mean of the input list or matrix. For lists pushes the average of all elements on the estack. For matrices pushes a row vector containing the averages of the matrix columns.

Inputs: *mat_lst_idx* — Indexes the input matrix or list.

Outputs: None

Assumptions: None

Side Effects: May expand expression stack, cause heap compression, or throw an error.

Availability: On AMS 2.00 and higher.

TI-89 / TI-92 Plus Differences: None

See Also: **push_median**

Example: This example takes a list pointed to by lst_idx (a test list is pushed on the estack). It then creates a new list on the estack where each element is the square of the difference between the element and the mean. This list is then saved in the variable L3.

```
EStackIndex lst_idx, mean_idx, sum_idx;
BYTE L3[] =  {0,'l','3',0};

push_quantum(END_TAG); push_Float(2); push_Float(5); push_Float(8);
push_quantum(LIST_TAG);
lst_idx = top_estack;                        /* Point to input list */
push_mean( lst_idx );
negate_top ();
mean_idx = top_estack;                       /* Point to -(mean(list)) */
/* Create a list where each element is the square of its difference from the mean */
push_quantum (END_TAG);
lst_idx = lst_idx - 1;   /* skip LIST_TAG */
while (END_TAG != *lst_idx) {
  push_sum( lst_idx, mean_idx );
  lst_idx = next_expression_index( lst_idx );
  sum_idx = top_estack;
  push_product( top_estack, top_estack );
  delete_expression( sum_idx );
}
push_quantum( LIST_TAG );
VarStore( L3+3, STOF_ESI, 0, top_estack );
```

push_median

Declaration: void **push_median** (EStackIndex *mat_lst_idx*)

Category(ies): Lists and Matrices, Statistics, Math

Description: Pushes onto the estack the median of the input list or matrix. For lists pushes the median on the estack. For matrices pushes a row vector containing the medians of the matrix columns.

Inputs: *mat_lst_idx* — Indexes the input matrix or list.

Outputs: None

Assumptions: None

Side Effects: May expand expression stack, cause heap compression, or throw an error.

Availability: On AMS 2.00 and higher.

TI-89 / TI-92 Plus Differences: None

See Also: **push_mean**

(continued)

push_median *(continued)*

Example: This example pushes a list on the estack that corresponds to the median for the input list. For each element in the input list the element in the output list is a negative one if the element is less than the median, a zero if it is equal to the median, and a positive one if it greater than the median. Note that since the output list is created in reverse order on the estack, **push_reversed_tail** is used to reverse the order of the list back to normal.

```
const BYTE INT_NEG1[] =  {END_TAG,1,1,NEGATIVE_INTEGER_TAG};
const BYTE INT_0[] =  {END_TAG,0,NONNEGATIVE_INTEGER_TAG};
const BYTE INT_1[] =  {END_TAG,1,1,NONNEGATIVE_INTEGER_TAG};
EStackIndex old_top, idx, median_idx, cond_idx;

old_top = top_estack;
push_quantum(END_TAG);
push_Float(-5.0);
push_Float(1.0);
push_Float(2.0);
push_Float(4.0);
push_Float(99.0);
idx = top_estack;
push_quantum(LIST_TAG);

push_median( idx+1 );
median_idx = top_estack;
push_quantum( END_TAG );
while (END_TAG != ESTACK(idx)) {
   push_difference( median_idx, idx );
   if (is0(top_estack))
      cond_idx = (EStackIndex) INT_0+2;
   else if (is_positive(top_estack))
      cond_idx = (EStackIndex) INT_1+3;
   else
      cond_idx = (EStackIndex) INT_NEG1+3;
   delete_expression( top_estack );
   push_expression( cond_idx );
   idx = next_expression_index( idx );
}
idx = top_estack;
push_reversed_tail( idx );
delete_between ( old_top, idx );
push_quantum( LIST_TAG );
```

push_mrow

Declaration: void **push_mrow** (EStackIndex *mult_idx,* EStackIndex *mat_idx,* EStackIndex *row_idx*)

Category(ies): Lists and Matrices, Math

Description: Pushes onto the estack a matrix which is the result of multiplying an expression (indexed by *mult_idx*) by all of the elements in a row (indexed by *row_idx*) of the input matrix (indexed by *mat_idx*).

row(*row_idx*) = row(*row_idx*) * *mult_idx*

Inputs:

mult_idx — Indexes the multiplier expression.

mat_idx — Indexes the input matrix.

row_idx — Indexes the row to operate on.

Outputs: None

Assumptions: None

Side Effects: May expand expression stack, cause heap compression, or throw an error.

Availability: On AMS 2.00 and higher.

TI-89 / TI-92 Plus Differences: None

See Also: **push_mrowadd, push_rowadd**

(continued)

push_mrow *(continued)*

Example: This example uses **push_mrow**, **push_mrowadd**, and **push_rowadd**. It uses an AMS global variable **Integer2Index** which is only available on AMS 2.05 and above so it undefines that global and redefines its own. In this way this code will run on all versions of AMS 2.00 and above. Notice how it stores its own test matrix (a matrix is just a list of lists).

```
#undef Integer2Index
const Quantum Integer2 [] = {2u, 1u, NONNEGATIVE_INTEGER_TAG};
#define Integer2Index ((EStackIndex) (Integer2+2))
const Quantum Integer5 [] = {5u, 1u, NONNEGATIVE_INTEGER_TAG};
#define Integer5Index ((EStackIndex) (Integer5+2))
const Quantum tArray[] =    {END_TAG,
   END_TAG,4,1,NONNEGATIVE_INTEGER_TAG, 3,1,NONNEGATIVE_INTEGER_TAG,LIST_TAG,
   END_TAG,2,1,NONNEGATIVE_INTEGER_TAG, 1,1,NONNEGATIVE_INTEGER_TAG,LIST_TAG,
   LIST_TAG
};
#define tArrayIndex ((EStackIndex) (tArray+sizeof(tArray)-1))
BYTE m5[] =  {0,'m','5',0};

/* Start out with [1,2;3,4] (note that lists are stored backwards */

push_mrow( Integer5Index, tArrayIndex, Integer1Index );
/* Now have [5,10;3,4] */

push_mrowadd( Integer2Index, top_estack, Integer1Index, Integer2Index );
/* Now have [5,10;13,24] */

push_rowadd( top_estack, Integer2Index, Integer1Index );
/* Now have [18,34;13,24] */

VarStore( m5+3, STOF_ESI, 0, top_estack );
```

push_mrowadd

Declaration: void **push_mrowadd**(EStackIndex *mult_idx,* EStackIndex *mat_idx,* EStackIndex *mult_row_idx*, EStackIndex *add_row_idx*)

Category(ies): Lists and Matrices, Math

Description: Pushes onto the estack a matrix (the input of which is indexed by *mat_idx*) which is the result of multiplying an expression (indexed by *mult_idx*) by all of the elements in a row (indexed by *mult_row_idx*) and adding those elements to another row (indexed by *add_row_idx*).

row(*add_row_idx*) = row(*mult_row_idx*) * *mult_idx* + row(*add_row_idx*)

Inputs:

mult_idx	—	Indexes the multiplier expression.
mat_idx	—	Indexes the input matrix.
mult_row_idx	—	Indexes the row to multiply by *multi_idx*.
add_row_idx	—	Indexes the row to add the values from the row indexed by *mult_row_idx*.

Outputs: None

Assumptions: None

Side Effects: May expand expression stack, cause heap compression, or throw an error.

Availability: On AMS 2.00 and higher.

TI-89 / TI-92 Plus Differences: None

See Also: **push_mrow, push_rowadd**

Example: See **push_mrow**.

push_newlist

Declaration: void **push_newlist** (EStackIndex *len_index*)

Category(ies): Lists and Matrices

Description: Pushes onto the estack a list of zeros.

Inputs: *len_index* — Indexes the number of elements in the new list.

Outputs: None

Assumptions: None

Side Effects: May expand expression stack, cause heap compression, or throw an error.

Availability: On AMS 2.00 and higher.

TI-89 / TI-92 Plus Differences: None

See Also: **push_newmat, push_randmat**

Example: This example creates a 10-element list of zeros named L2. It then stores to L2[1] and L2[11]. Note that **VarStore** can be used to increase a list by at most one element.

```
BYTE lName[] = {0,'l','2',0};

/* create a list of 10 zeros on the estack */
push_ushort_to_integer( 10 );
push_newlist( top_estack );
/* Store list created with push_newlist to 'L2' */
VarStore( lName+3, STOF_ESI, 0, top_estack );
/* Store 123 to L2[1] */
push_ushort_to_integer( 123 );
VarStore( lName+3, STOF_ELEMENT, 0, top_estack, 1u, 0 );
/* Store 1234567890 to L2[11] */
push_ulong_to_integer( 1234567890l );
VarStore( lName+3, STOF_ELEMENT, 0, top_estack, 11u, 0 );
```

push_newmat

Declaration: void **push_newmat** (EStackIndex *rdim_index*, EStackIndex *cdim_index*)

Category(ies): Lists and Matrices

Description: Pushes onto the estack a matrix of zeros.

Inputs: *rdim_index* — Indexes the number of rows in the new matrix.

cdim_index — Indexes the number of columns in the new matrix.

Outputs: None

Assumptions: None

Side Effects: May expand expression stack, cause heap compression, or throw an error.

Availability: On AMS 2.00 and higher.

TI-89 / TI-92 Plus Differences: None

See Also: **push_newlist, push_randmat**

Example: This example creates a 2x2 matrix named M0 and stores to one of its elements.

```
BYTE mName[] = {0,'m','0',0};

/* create a 2x2 matrix of zeros on the estack */
push_ushort_to_integer( 2 );
push_newmat( top_estack, top_estack );
/* Store matrix created with push_newmat to 'M0' */
VarStore( mName+3, STOF_ESI, 0, top_estack );
/* Store 1.234 to M0[2,1] */
push_Float( 1.234 );
VarStore( mName+3, STOF_ELEMENT, 0, top_estack, 1u, 2u );
```

push_prodlist

Declaration: void **push_prodlist** (EStackIndex *i*)

Category(ies): Lists and Matrices, Math

Description: Pushes onto the estack the product of the matrix or list elements indexed by *i.* For a list the result is an expression which is the product of all elements of the list. For a matrix the result is a row vector containing the products of the matrix columns.

Inputs: *i* — Indexes the input list or matrix.

Outputs: None

Assumptions: None

Side Effects: May expand expression stack, cause heap compression, or throw an error.

Availability: On AMS 2.00 and higher.

TI-89 / TI-92 Plus Differences: None

See Also: **push_cumsum, push_sumlist**

Example:

If j indexes at the bolded tag the matrix [1, 2; 3, 4] as follows
END_TAG END_TAG 4 1 NONNEGATIVE_INTEGER_TAG 3 1 NONNEGATIVE_INTEGER_TAG LIST_TAG END_TAG 2 1 NONNEGATIVE_INTEGER_TAG 1 1 NONNEGATIVE_INTEGER_TAG LIST_TAG **LIST_TAG**

then

```
push_prodlist (j);
```

pushes a row vector containing the column products of the matrix, [3, 8], onto the estack such that **top_estack** points to the bolded tag as follows.
END_TAG END_TAG 8 1 NONNEGATIVE_INTEGER_TAG 3 1 NONNEGATIVE_INTEGER_TAG LIST_TAG **LIST_TAG**

push_randmat

Declaration: void **push_randmat** (EStackIndex *rdim_index*, EStackIndex *cdim_index*)

Category(ies): Lists and Matrices

Description: Pushes onto the estack a matrix of random integers between –9 and +9.

Inputs: *rdim_index* — Indexes the number of rows in the new matrix.

cdim_index — Indexes the number of columns in the new matrix.

Outputs: None

Assumptions: None

Side Effects: May expand expression stack, cause heap compression, or throw an error.

Availability: On AMS 2.00 and higher.

TI-89 / TI-92 Plus Differences: None

See Also: **push_newlist**, **push_newmat**

Example: This example pushes a 4x10 matrix of random integers between -9 and 9 on the estack.

```
EStackIndex num4;

push_ushort_to_integer( 4 );
num4 = top_estack;
push_ushort_to_integer( 10 );
push_randmat( num4, top_estack );
```

push_red_row_ech

Declaration: void **push_red_row_ech** (EStackIndex *mat_idx*, EStackIndex *tol_idx*)

Category(ies): Lists and Matrices, Math

Description: Pushes onto the estack the reduced row echelon form of a matrix.

Inputs:
mat_idx — Indexes the input matrix.

tol_idx — If not NULL then indexes a tolerance factor. Any matrix element is treated as zero if its absolute value is less than the tolerance.

Outputs: None

Assumptions: None

Side Effects: May expand expression stack, cause heap compression, or throw an error.

Availability: On AMS 2.00 and higher.

TI-89 / TI-92 Plus Differences: None

See Also: **push_row_echelon**

Example: Assuming rMat indexes a matrix and colsum is the sum of the absolute values of the columns of that matrix, this example computes a tolerance factor from colsum and then pushes the reduced row echelon form of rMat onto the estack using the computed tolerance.

```
short n;
float tol, colsum[4];
EStackIndex rMat;

.
.
.
/* Assume rMat already setup */
tol = colsum[0];
for ( n=1 ; n<=3 ; n++ )
   if ( colsum[n] > tol )
      tol = colsum[n];
tol *= 1.2e-12;
push_Float( tol );
push_red_row_ech( rMat, top_estack );
```

push_reversed_tail

Declaration: void **push_reversed_tail** (EStackIndex *k*)

Category(ies): Lists and Matrices

Description: Pushes an END_TAG onto the estack, then pushes the elements of the tail in reverse order.

Inputs: *k* — Indexes the top tag of a tail of expressions.

Outputs: None

Assumptions: None

Side Effects: None

Availability: On AMS 1.05 and higher.

TI-89 / TI-92 Plus Differences: None

See Also: None

Example:

```
push_quantum (END_TAG);
push_Float (3.7);
push_Float (5.2);
/* Pushes tagged 3.7 on top of tagged 5.2 on top of an END_TAG. */
push_reversed_tail (top_estack);
```

push_row_echelon

Declaration: void **push_row_echelon** (EStackIndex *mat_idx*, EStackIndex *tol_idx*)

Category(ies): Lists and Matrices, Math

Description: Pushes onto the estack the row echelon form of a matrix onto the estack.

Inputs:

mat_idx — Indexes the input matrix.

tol_idx — If not NULL then indexes a tolerance factor. Any matrix element is treated as zero if its absolute value is less than the tolerance.

Outputs: None

Assumptions: None

Side Effects: May expand expression stack, cause heap compression, or throw an error.

Availability: On AMS 2.00 and higher.

TI-89 / TI-92 Plus Differences: None

See Also: **push_red_row_ech**

Example:

```
push_row_echelon( top_estack, NULL );
```

push_rowadd

Declaration: void **push_rowadd** (EStackIndex *mat_idx,* EStackIndex *row_src_idx,* EStackIndex *row_dest_idx*)

Category(ies): Lists and Matrices, Math

Description: Pushes onto the estack a matrix which is the result of adding a row (indexed by *row_src_idx*) to another row (indexed by *row_dest_idx*) of the input matrix (indexed by *mat_idx*).

row(*row_dest_idx*) = row(*row_dest_idx*) + row(*row_src_idx*)

Inputs:

mat_idx — Indexes the input matrix.

row_src_idx — Indexes the row to add from.

row_dest_idx — Indexes the row to add to.

Outputs: None

Assumptions: None

Side Effects: May expand expression stack, cause heap compression, or throw an error.

Availability: On AMS 2.00 and higher.

TI-89 / TI-92 Plus Differences: None

See Also: **push_mrow, push_mrowadd**

Example: See **push_mrow**.

push_rowdim

Declaration: void **push_rowdim** (EStackIndex *mat_idx*)

Category(ies): Lists and Matrices, Math

Description: Pushes onto the estack the row dimension of the matrix indexed by *mat_idx*.

Inputs: *mat_idx* — Indexes the input matrix.

Outputs: None

Assumptions: None

Side Effects: May expand expression stack, cause heap compression, or throw an error.

Availability: On AMS 2.00 and higher.

TI-89 / TI-92 Plus Differences: None

See Also: **push_coldim**

Example: See **push_identity_mat**.

push_rownorm

Declaration: void **push_rownorm** (EStackIndex *mat_idx*)

Category(ies): Lists and Matrices, Math

Description: Pushes onto the estack the row norm of the matrix indexed by *mat_idx*. The row norm is the largest value of the sums of the absolute values of the elements in each row of the matrix.

Inputs: *mat_idx* — Indexes the input matrix.

Outputs: None

Assumptions: None

Side Effects: May expand expression stack, cause heap compression, or throw an error.

Availability: On AMS 2.00 and higher.

TI-89 / TI-92 Plus Differences: None

See Also: **push_colnorm, push_matnorm**

Example: See **push_colnorm**.

push_rowswap

Declaration: void **push_rowswap** (EStackIndex *mat_idx*, EStackIndex *row_index1*, EStackIndex *row_index2*)

Category(ies): Lists and Matrices

Description: Pushes onto the estack a matrix with two rows swapped.

Inputs:
mat_idx — Index of the input matrix.
row_index1 — Index of one of the rows to swap.
row_index2 — Index of the other row to swap.

Outputs: None

Assumptions: None

Side Effects: May expand expression stack, cause heap compression, or throw an error.

Availability: On AMS 2.00 and higher.

TI-89 / TI-92 Plus Differences: None

See Also: None

Example: This code fragment swaps two rows in a matrix on the estack (indexed by mat_idx) if piv_row is not equal to cur_row. The old matrix is removed from the estack and mat_idx is set to point to the newly created matrix.

```
EStackIndex piv_row_idx, cur_row_idx, mat_idx, old_mat_idx;
unsigned short piv_row, cur_row;

.
.
.
/* assume mat_idx, piv_row, and cur_row already set */
if (piv_row != cur_row) {
   old_mat_idx = mat_idx;
   push_ulong_to_integer ((unsigned long)piv_row);
   piv_row_idx = top_estack;
   push_ulong_to_integer ((unsigned long)cur_row);
   cur_row_idx = top_estack;

   /* Swap rows so current row is pivot row. */
   push_rowswap (mat_idx, piv_row_idx, cur_row_idx);
   delete_between (old_mat_idx, cur_row_idx);
   delete_expression (mat_idx);
   mat_idx = top_estack;
}
.
.
.
```

push_sign

Declaration: void **push_sign** (EStackIndex *k*)

Category(ies): Lists and Matrices

Description: Pushes the sign of the expression indexed by *k* onto the estack.
sign(positive) -> 1, sign(+0) -> 1, and sign(+0.0) -> 1;
sign(negative) -> -1, sign(-0) -> -1, and sign(-0.0) -> -1;
sign(+-infinity) -> sign(0) and sign(+-FloatInfinity) -> sign(0.0);
sign(undef) -> sign(0) and sign(NAN) -> sign(0).

For unreal z, sign(z) -> z/abs(z).

If IS_DOMAIN_REAL sign(0) and sign(0.0) represent + -1 and display that way.

If IS_DOMAIN_COMPLEX, sign(0) displays as sign(0) and sign(0.0) displays as sign(0.0), both representing the unit circle in the complex plane.

Inputs: *k* — Indexes the top tag of an internally-simplified algebraic expression, algebraic comparison, or aggregate thereof.

Outputs: None

Assumptions: None

Side Effects: None

Availability: On AMS 2.02 and higher.

TI-89 / TI-92 Plus Differences: None

See Also: None

Example:

```
push_Float (-3.7);
push_sign (top_estack);  /* Pushes –1.0 onto the estack. */
```

push_stddev

Declaration: void **push_stddev** (EStackIndex *mat_lst_idx*)

Category(ies): Lists and Matrices, Statistics, Math

Description: Pushes onto the estack the standard deviation of a list or matrix. For a list pushes the standard deviation of the elements onto the estack. For a matrix pushes a row vector containing the standard deviations of the matrix columns.

Inputs: *mat_lst_idx* — Indexes the input matrix or list.

Outputs: None

Assumptions: None

Side Effects: May expand expression stack, cause heap compression, or throw an error.

Availability: On AMS 2.00 and higher.

TI-89 / TI-92 Plus Differences: None

See Also: **push_variance**

Example:

```
EStackIndex lst_idx;

push_quantum(END_TAG);
push_Float(1.5);
push_Float(9.1);
push_Float(5.1);
push_quantum(LIST_TAG);
lst_idx = top_estack;
push_stddev( lst_idx );
push_variance( lst_idx );
```

push_submat

Declaration: void **push_submat** (EStackIndex *mat_idx,* EStackIndex *start_row_idx*, EStackIndex *start_col_idx*, EStackIndex *end_row_idx*, EStackIndex *end_col_idx*)

Category(ies): Lists and Matrices

Description: Pushes onto the estack a submatrix of the matrix indexed by *mat_idx*. An error is thrown if any of the row or column values are outside of the size of the input matrix.

Inputs:

mat_idx	—	Index of input matrix.
start_row_idx	—	Index of starting row (1 is used if NULL).
start_col_idx	—	Index of starting column (1 is used if NULL).
end_row_idx	—	Index of ending row (the number of rows in the input matrix is used if NULL).
end_col_idx	—	Index of ending column (the number of columns in the input matrix is used if NULL).

Outputs: None

Assumptions: None

Side Effects: May expand expression stack, cause heap compression, or throw an error.

Availability: On AMS 2.00 and higher.

TI-89 / TI-92 Plus Differences: None

See Also: None

(continued)

push_submat *(continued)*

Example: This example pushes the submatrix [(1,1) . . . (nRows, nCols)] of the matrix indexed by inMat onto the estack.

```
unsigned long nRows, nCols;
EStackIndex endRowIdx, endColIdx, oldTop, inMat;
.
.
.
/* assume inMat, nRows, nCols already setup */
oldTop = top_index;
push_ulong_to_integer ( nRows );
endRowIdx = top_estack;
push_ulong_to_integer ( nCols );
endColIdx = top_estack;
push_submat ( inMat, NULL, NULL, endRowIdx, endColIdx );
delete_between( oldTop, endColIdx );
.
.
.
```

push_sumlist

Declaration: void **push_sumlist** (EStackIndex *mat_lst_idx*)

Category(ies): Lists and Matrices, Math

Description: Pushes onto the estack the sum of the matrix or list elements indexed by *mat_lst_idx*. For a list the result is an expression which is the sum of all elements of the list. For a matrix the result is a row vector containing the sums of the matrix columns.

Inputs: *mat_lst_idx* — Indexes the input list or matrix.

Outputs: None

Assumptions: None

Side Effects: May expand expression stack, cause heap compression, or throw an error.

Availability: On AMS 2.00 and higher.

TI-89 / TI-92 Plus Differences: None

See Also: **push_cumsum**

Example: See **push_cumsum**.

push_transpose_aux

Declaration: void **push_transpose_aux** (EStackIndex *mat_idx*, unsigned short *flag*)

Category(ies): Lists and Matrices, Math

Description: Transposes a matrix or its complex conjugate.

Inputs:

mat_idx — EStackIndex of a matrix.

flag — 0 — transpose the matrix. The matrix may be in internal or external tokenized form and the result is in the same form.

1 — transpose the complex conjugate of the matrix. The matrix must be in internal tokenized form and the result will be in internal tokenized form.

Outputs: Pushes the transposed matrix onto the expression stack.

Assumptions: None

Side Effects: May cause estack expansion, heap compression, or throw an error.

Availability: On AMS 1.05 and higher.

TI-89 / TI-92 Plus Differences: None

See Also: None

(continued)

push_transpose_aux *(continued)*

Example:

If m indexes the bolded tag of the matrix [a, b; c, d] in the following estack expression
END_TAG END_TAG D_VAR_TAG C_VAR_TAG LIST_TAG END_TAG B_VAR_TAG A_VAR_TAG LIST_TAG **LIST_TAG**

then

```
push_transpose_aux (m, 0);
```

pushes the transposed matrix [a, c; b, d] onto the estack such that **top_estack** points to the bolded tag as follows.
END_TAG END_TAG D_VAR_TAG B_VAR_TAG LIST_TAG END_TAG C_VAR_TAG A_VAR_TAG LIST_TAG **LIST_TAG**

If m indexes the bolded tag in the following internally tokenized matrix [b, 3+4i; a-i, -2i] as follows
END_TAG END_TAG 0 NONNEGATIVE_INTEGER_TAG 2 1 NEGATIVE_INTEGER_TAG IM_RE_TAG A_VAR_TAG 1 1 NEGATIVE_INTEGER_TAG IM_RE_TAG LIST_TAG END_TAG 3 1 NONNEGATIVE_INTEGER_TAG 4 1 NONNEGATIVE_INTEGER_TAG IM_RE_TAG B_VAR_TAG LIST_TAG **LIST_TAG**

then

```
push_transpose_aux (m, 1);
```

pushes the complex conjugate transpose matrix [b, a + i; 3 - 4i, 2i] onto the estack such that **top_estack** points to the bolded tag as follows.
END_TAG END_TAG 0 NONNEGATIVE_INTEGER_TAG 2 1 NONNEGATIVE_INTEGER_TAG IM_RE_TAG 3 1 NONNEGATIVE_INTEGER_TAG 4 1 NEGATIVE_INTEGER_TAG IM_RE_TAG LIST_TAG END_TAG A_VAR_TAG 1 1 NONNEGATIVE_INTEGER_TAG IM_RE_TAG B_VAR_TAG LIST_TAG **LIST_TAG**

push_unitv

Declaration: void **push_unitv** (EStackIndex *vec_idx*)

Category(ies): Lists and Matrices, Math

Description: Pushes onto the estack the unit vector based on the input vector. For a row vector pushes the corresponding unit row vector. For a column vector pushes the corresponding unit column vector.

Inputs: *vec_idx* — Indexes the input row or column vector.

Outputs: None

Assumptions: None

Side Effects: May expand expression stack, cause heap compression, or throw an error.

Availability: On AMS 2.00 and higher.

TI-89 / TI-92 Plus Differences: None

See Also: None

Example: See **push_round**.

push_variance

Declaration: void **push_variance** (EStackIndex *mat_lst_idx*)

Category(ies): Lists and Matrices, Statistics, Math

Description: Pushes onto the estack the variance of a list or matrix. For a list pushes the variance of the elements onto the estack. For a matrix pushes a row vector containing the variances of the matrix columns.

Inputs: *mat_lst_idx* — Indexes the input matrix or list.

Outputs: None

Assumptions: None

Side Effects: May expand expression stack, cause heap compression, or throw an error.

Availability: On AMS 2.00 and higher.

TI-89 / TI-92 Plus Differences: None

See Also: **push_stddev**

Example: See **push_stddev**.

remaining_element_count

Declaration: ELEMENT_COUNT **remaining_element_count** (EStackIndex *i*)

Category(ies): Lists and Matrices

Description: Determines the number of remaining elements in the expression indexed by *i*.

Inputs: *i* — Indexes a tail of expressions terminated by an END_TAG.

Outputs: Returns the number of elements in the tail indexed by *i*.

Assumptions: None

Side Effects: None

Availability: On AMS 1.05 and higher.

TI-89 / TI-92 Plus Differences: None

See Also: **last_element_index**

Example:

```
push_quantum (END_TAG);
push0 ();
push1 ();
remaining_element_count (top_estack); /* Returns 2 */
```

Appendix A: System Routines — Logic

and_onto_top

Declaration: void **and_onto_top** (EStackIndex *i*)

Category(ies): Logic

Description: Replaces the top expression with the internally-simplified logical “and” of it with the expression indexed by *i*.

Inputs: *i* — Index of the top tag of a Boolean expression.

Outputs: None

Assumptions: **top_estack** indexes the top tag of a Boolean expression.

Side Effects: None

Availability: On AMS 2.02 and higher.

TI-89 / TI-92 Plus Differences: None

See Also: **push_and, replace_top2_with_and**

Example:

```
void push_but_conjunct_factor (EStackIndex i, EStackIndex j)
/* i indexes a (perhaps degenerate) conjunct and j indexes one of its
   remaining Boolean factors.
   Pushes onto the estack the conjunct without the lead factor of j.
*/
{  if (i == j)
      push_expression (remaining_conjuncts_index (i));
   else
      {  push_but_conjunct_factor (remaining_conjuncts_index (i), j);
         and_onto_top (lead_conjunct_factor_index (i));
      }
}
```

is_equivalent_to

Declaration: Boolean **is_equivalent_to** (EStackIndex *i*, EStackIndex *j*)

Category(ies): Logic

Description: Determines whether the expressions indexed by *i* and *j* are identical or can be determined by the CAS to be equivalent.

Inputs: *i, j* — Indices of the top tags of internally-simplified expressions or strings.

Outputs: Returns TRUE if the expressions indexed by *i* and *j* are identical or can be determined by the CAS to be equivalent. Otherwise returns FALSE, even though they might actually be equivalent.

Assumptions: None

Side Effects: None

Availability: On AMS 2.02 and higher.

TI-89 / TI-92 Plus Differences: None

See Also: **compare_expressions, are_expressions_identical**

Example:

```
push_quantum_as_nonnegative_int (2u);
exponent = top_estack;
push_quantum (8u);                          /* push x */
add1_to_top ();                             /* top_estack -> x + 1 */
replace_top2_with_pow (exponent);
unexpanded = top_estack;                    /* unexpanded -> (x + 1)^2 */
push_quantum_as_nonnegative_int (2u);
coefficient = top_estack;
push_quantum (8u);                          /* push x */
replace_top2_with prod (coefficient);       /* top_estack -> 2 * x  */
add1_to_top();
partial_sum = top_estack;                   /* partial_sum -> 2 * x + 1  */
push_quantum_as_nonnegative_int (2u);
exponent = top_estack;
push_quantum (8u);                          /* push x */
replace_top2_with_pow (exponent);           /* top_estack -> x^2 */
replace_top2_with_sum (partial_sum);        /* top_estack -> x^2 + 2 * x + 1  */
is_equivalent_to (top_estack, unexpanded);  /* Returns TRUE */
```

is_negative

Declaration: Boolean **is_negative** (EStackIndex *i*)

Category(ies): Logic

Description: Determines whether the internally-simplified algebraic expression indexed by *i* is negative for all finite values of all variables therein. Takes into account bounds indexed by **NG_such_that_index**. -0 and -0.0 are NOT considered negative.

Inputs: *i* — Index of the top tag of an internally-simplified expression.

Outputs: Returns TRUE if it can determine that the internally-simplified algebraic expression indexed by *i* is negative for all finite values of all variables therein. Otherwise returns FALSE, even though the expression indexed by *i* might always be negative.

Assumptions: None

Side Effects: None

Availability: On AMS 2.02 and higher.

TI-89 / TI-92 Plus Differences: None

See Also: **is_nonnegative, is_positive, is_nonpositive, is0, is_never0, is_real**

Example:

```
int ge0_lt0_unknown (EStackIndex i)
/* i indexes a zero expression.
   Returns 1 if it determines that the expression >= 0, else -1 if < 0, else 0.
*/
{  return is_nonnegative (i) ? 1 : -is_negative (i);
}
```

is_never0

Declaration: Boolean **is_never0** (EStackIndex *i*)

Category(ies): Logic

Description: Determines whether the internally-simplified algebraic expression indexed by *i* is nonzero (possibly unreal) for all finite values of all variables therein.

Inputs: *i* — Index of the top tag of an internally-simplified expression.

Outputs: Returns TRUE if it can determine that the internally-simplified algebraic expression indexed by *i* is nonzero (possibly unreal) for all finite values of all variables therein. Otherwise returns FALSE, even though expression *i* might always be nonzero.

Assumptions: None

Side Effects: None

Availability: On AMS 2.02 and higher.

TI-89 / TI-92 Plus Differences: None

See Also: **is_nonnegative, is_positive, is_nonpositive, is0, is_negative, is_real**

Example:

```
void push_maybe0_factors (EStackIndex i)
/* i initially indexes a term (that might recur to a sum).
   Pushes the product of the factors of i that might be zero.
*/
{  if (IS_NUMBER_TAG (ESTACK (i)))
         push_expression (Integer1Index);
   else
   {  push_maybe0_factors (remaining_factors_index (i));
      i = lead_factor_index (i);
      if (is_never0 (i) &&
         (EXPONENTIATION_TAG != ESTACK (i) ||
         is_nonnegative (POWER_EXPONENT_INDEX (i)) ||
         is_never0 (POWER_BASE_INDEX (i)) ) )
         ;
      else
         times_top (i);
   }
}  /* end push_maybe0_factors */
```

is_nonnegative

Declaration: Boolean **is_nonnegative** (EStackIndex *i*)

Category(ies): Logic

Description: Determines whether the internally-simplified algebraic expression indexed by *i* is non-negative real for all finite values of all variables therein. Takes into account bounds indexed by **NG_such_that_index**. ALL zeros are considered non-negative, even ⁻0 and ⁻0.0

Inputs: *i* — Index of the top tag of an internally-simplified expression.

Outputs: Returns TRUE if it can determine that the internally-simplified algebraic expression indexed by *i* is non-negative real for all finite values of all variables therein. Otherwise returns FALSE, even though the expression indexed by *i* might always be non-negative.

Assumptions: None

Side Effects: None

Availability: On AMS 2.02 and higher.

TI-89 / TI-92 Plus Differences: None

See Also: **is_never0, is_positive, is_nonpositive, is0, is_negative, is_real**

Example:

```
EStackIndex index_nonnegative_factor (EStackIndex i)
/* i indexes a term.
   Returns NULL_INDEX if it cannot determine that one of its explicit
   factors is non-negative.
   Otherwise returns i or the index of its first remaining factors for which
   the lead factor has been determined to be nonnegative.
*/
{  while (! is1 (i))
      if (is_nonnegative (lead_factor_index (i)))
         return i;
      else
         i = remaining_factors_index (i);
   return NULL_INDEX;
}
```

is_nonpositive

Declaration: Boolean **is_nonpositive** (EStackIndex *i*)

Category(ies): Logic

Description: Determines whether the internally-simplified algebraic expression indexed by *i* is real and <= 0 for all finite values of all variables therein. Takes into account bounds indexed by **NG_such_that_index**. -0 and -0.0 are NOT considered nonpositive.

Inputs: *i* — Index of the top tag of an internally-simplified expression.

Outputs: Returns TRUE if it can determine that the internally-simplified algebraic expression indexed by *i* is real and <= 0 for all finite values of all variables therein. Otherwise returns FALSE, even though the expression indexed by *i* might always be nonpositive.

Assumptions: None

Side Effects: None

Availability: On AMS 2.02 and higher.

TI-89 / TI-92 Plus Differences: None

See Also: **is_never0, is_positive, is_nonnegative, is0, is_negative, is_real**

Example:

```
Boolean have_same_sign (EStackIndex i, EStackIndex j)
/* Returns TRUE if the expressions indexed by i and j are both non-negative
   or both nonpositive.
*/
{  return is_nonnegative (i) && is_nonnegative (j) ||
          is_nonpositive (i) && is_nonpositive (j);
}
```

is_positive

Declaration: Boolean **is_positive** (EStackIndex *i*)

Category(ies): Logic

Description: Determine whether the internally-simplified algebraic expression indexed by *i* is positive real for all finite values of all variables therein. Takes into account bounds indexed by **NG_such_that_index**. +0 and +0.0 are NOT considered positive.

Inputs: *i* — Index of the top tag of an internally-simplified expression.

Outputs: Returns TRUE if it can determine that the internally-simplified algebraic expression indexed by *i* is positive real for all finite values of all variables therein. Otherwise returns FALSE, even though the expression indexed by *i* might always be positive.

Assumptions: None

Side Effects: None

Availability: On AMS 2.02 and higher.

TI-89 / TI-92 Plus Differences: None

See Also: **is_never0, is_nonpositive, is_nonnegative, is0, is_negative, is_real**

Example:

```
int gt0_le0_unknown (EStackIndex i)
/* i indexes a nonzero expression.
   Returns 1 if it determines that the expression is > 0, else -1 if <= 0, else 0.
*/
{  return is_positive (i) ? 1 : -is_nonpositive (i);
}
```

is_real

Declaration: Boolean **is_real** (EStackIndex *i*)

Category(ies): Logic

Description: Determine whether the internally-simplified algebraic expression indexed by *i* is real for all finite real values of all variables therein.

Inputs: *i* — Index of the top tag of an internally-simplified expression or an aggregate thereof.

Outputs: Returns TRUE if it can determine that the internally-simplified algebraic expression indexed by *i* is real for all finite real values of all variables therein. Otherwise returns FALSE, even though the expression might always be real.

Assumptions: None

Side Effects: None

Availability: On AMS 2.02 and higher.

TI‑89 / TI‑92 Plus Differences: None

See Also: None

Example:

```
void push_real_factors (EStackIndex i)
/* Pushes the product of the decidably real factors of i. */
{  if (MULTIPLY_TAG == ESTACK (i))
   {  push_real_factors (next_expression_index (--i));
      if (is_real (i))
         times_top (i);
   }
   else if (is_real (i))
      push_expression (i);
   else
      push_expression (Integer1Index);
}
```

is_undefined

Declaration: Boolean **is_undefined** (EStackIndex *k*)

Category(ies): Logic

Description: Determines whether the real and/or imaginary part of the expression indexed by *k* is an UNDEFINED_TAG.

Inputs: *i* — Indexes the top tag of an internally-simplified Algebraic expression.

Outputs: Returns TRUE if the real and/or imaginary part of the expression indexed by *k* is UNDEFINED_TAG. Otherwise returns FALSE.

Assumptions: None

Side Effects: None

Availability: On AMS 2.04 and higher.

TI-89 / TI-92 Plus Differences: None

See Also: **is_float_signed_infinity, is_float_infinity, is_float_unsigned_inf_or_nan, is_float_transfinite, is_nan**

Example:

```
push1();
real_part = top_estack;
push_quantum (UNDEFINED_TAG);
replace_top2_with_imre (real_part);
is_undefined (top_estack);  /* Returns TRUE */
```

lead_conjunct_factor_index

Declaration: EStackIndex **lead_conjunct_factor_index** (EStackIndex *i*)

Category(ies): Logic

Description: Determines the lead conjunction factor of the expression indexed by *i.*

Internally-simplified conjuncts have the most main Boolean factor shallowest, with less main factors deeper. Also, the lead Boolean factor of an internally-simplified conjunct is never a conjunct.

Inputs: *i* — Index of the top tag of a Boolean expression.

Outputs: If *i* indexes an AND_TAG, returns the index of the operand just below it. Otherwise returns *i.*

Assumptions: None

Side Effects: None

Availability: On AMS 2.02 and higher.

TI-89 / TI-92 Plus Differences: None

See Also: **remaining_conjuncts_index, lead_disjunct_term_index, remaining_disjuncts_index**

Example:

```
lead_conjunct_factor_index ("internalSimplified" ((b and a) and c))
```

Returns index (a).

```
lead_conjunct_factor_index ("internalSimplified"(x > 3))
```

Returns index (x > 3),

```
void push_but_conjunct_factor (EStackIndex i, EStackIndex j)
/* i indexes a (perhaps degenerate) conjunct and j == i or j indexes one of its
   remaining factors.
   Pushes onto the estack the conjunct without the lead factor of j.
*/
{  if (i == j) push_expression (remaining_conjuncts_index (i));
   else
   {
      push_but_conjunct_factor (remaining_conjuncts_index (i), j);
      and_onto_top (lead_conjunct_factor_index (i));
   }
}
```

lead_disjunct_term_index

Declaration: EStackIndex **lead_disjunct_term_index** (EStackIndex *i*)

Category(ies): Logic

Description: Determines the lead disjunct term of the expression indexed by *i*.

Internally-simplified disjuncts have the most main Boolean term shallowest, with less main terms deeper. Also, the lead Boolean term of an internally-simplified disjunct is never a disjunct.

Inputs: *i* — Index of the top tag of an internally-simplified Boolean expression.

Outputs: If *i* indexes an OR_TAG, returns the index of the operand just below it. Otherwise returns *i*.

Assumptions: None

Side Effects: None

Availability: On AMS 2.02 and higher.

TI-89 / TI-92 Plus Differences: None

See Also: **remaining_conjuncts_index, lead_conjunct_factor_index, remaining_disjuncts_index**

Example: If i indexes the internally-simplified expression
`(b or a) or c`
then **lead_disjunct_term_index**(i) returns index(a).

If i indexes the internally-simplified expression
`x > 3`
then **lead_disjunct_term_index**(i) returns index(x > 3).

```
void push_but_disjunct_term (EStackIndex i, EStackIndex j)
/* i indexes a (perhaps degenerate) disjunct and j == i or j indexes one of its
   remaining terms.
   Pushes onto the estack the disjunct without the lead term of j.
*/
{  if (i == j)
      push_expression (remaining_disjuncts_index (i));
   else
   {
      push_but_disjunct_term (remaining_disjuncts_index (i), j);
      and_onto_top (lead_disjunct_term_index (i));
   }
}
```

or_onto_top

Declaration: void **or_onto_top** (EStackIndex *i*)

Category(ies): Logic

Description: Replaces the top expression on the estack with the internally-simplified logical “or” of it with the expression indexed by *i*.

Inputs: *i* — Indexes the top tag of an internally-simplified Boolean expression.

Outputs: None

Assumptions: *i* and **top_estack** index internally-simplified Boolean expressions.

Side Effects: None

Availability: On AMS 2.02 and higher.

TI-89 / TI-92 Plus Differences: None

See Also: **push_or, replace_top2_with_or**

Example:

```
push_quantum (8u);
foo = top_estack;      /* foo -> variable x */
push_quantum (8u);
push_quantum (NOT_TAG);  /* top_estack -> NOT x */
or_onto_top (foo);    /* Replaces top NOT x with TRUE_TAG. */
```

push_and

Declaration: void **push_and** (EStackIndex *i*, EStackIndex *j*)

Category(ies): Logic

Description: Pushes onto the estack the internally-simplified form of the logical “and” of the Boolean expressions indexed by *i* and *j*.

Boolean expressions generally simplify to a disjunctive form a1 and a2 and . . . or b1 and b2 and . . . or . . . .

Example: a and (b or c<d) simplifies to a and b or a and c<d.

Inputs: *i*, *j* — Indices of the top tags of an internally-simplified comparisons, Boolean expressions or aggregates thereof.

Outputs: None

Assumptions: None

Side Effects: None

Availability: On AMS 2.02 and higher.

TI-89 / TI-92 Plus Differences: None

See Also: **and_onto_top, replace_top2_with_and**

Example:

```
void and_onto_top (EStackIndex i)
/* i and top_estack index Boolean expressions.
   Replaces the latter expression with the logical And of the 2 expressions.
*/
{  Access_AMS_Global_Variables;
   EStackIndex old_top = top_estack;
   if (FALSE_TAG == ESTACK (old_top) ||
         TRUE_TAG == ESTACK (i) ||
         are_expressions_identical (i, old_top) )
      return;
   if (TRUE_TAG == ESTACK (old_top) || FALSE_TAG == ESTACK (i))
      {  top_estack = next_expression_index (top_estack);
         push_expression (i);
      }
   else
      {  push_and (i, old_top);
         delete_expression (old_top);
      }
}
```

push_but_conjunct_factor

Declaration: void **push_but_conjunct_factor** (EStackIndex *i*, EStackIndex *j*)

Category(ies): Logic

Description: An internally-simplified conjunct has either the degenerate form
deepest: b1,
or the flattened form
deepest: bn . . . b2 AND_TAG b1 AND_TAG,
where Boolean factors b1 through bn are not AND_TAGs.

i indexes such a conjunct (may be degenerate) and *j* == *i* or *j* indexes of its partial conjuncts: either the deepest Boolean factor, all of *i*, or one of the other above AND_TAGs. Pushes onto the estack the conjunct *i* without the lead Boolean factor of *j*.

Examples:
If *i* indexes the above flattened form and *j* indexes the partial conjunct bn . . . b2 AND_TAG, then `push_but_conjuct_factor (i, j)` pushes bn . . . b3 AND_TAG b1.

If *i* indexes an equation and *j* == *i*, then `push_but_conjuct_factor (i, j)` pushes a TRUE_TAG.

Inputs: *i* — Index of the top tag of an internally-simplified Boolean expression.

j — *i* or the index of one of its partial conjuncts.

Outputs: None

Assumptions: None

Side Effects: None

Availability: On AMS 2.02 and higher.

TI-89 / TI-92 Plus Differences: None

See Also: **lead_conjunct_factor_index, remaining_conjuncts_index**

Example:

```
void push_but_conjunct_factor (EStackIndex i, EStackIndex j)
{  if (i == j)
      push_expression (remaining_conjuncts_index (i));
   else
      {  push_but_conjunct_factor (remaining_conjuncts_index (i), j);
         and_onto_top (lead_conjunct_factor_index (i));
      }
}
```

push_not

Declaration: void **push_not** (EStackIndex *i*)

Category(ies): Logic

Description: Pushes onto the estack the internally-simplified form of the logical negation of the expression indexed by *i*. ~ distributes over equations and inequalities. Also, Boolean expressions generally simplify to a disjunctive form a1 and a2 and . . . or b1 and b2 and . . . or . . . , where a1, a2, etc. are equations, inequalities, variables, or a NOT_TAG on top of a variable.

Example: ~(~a or (b and c = d)) simplifies to a and ~b or a and c ≠ d.

Inputs: *i* — Indexes the top tag of an internally-simplified Boolean expression.

Outputs: None

Assumptions: None

Side Effects: None

Availability: On AMS 2.02 and higher.

TI-89 / TI-92 Plus Differences: None

See Also: **push_and, push_or, and_onto_top, or_onto_top, replace_top2_with_and, replace_top2_with_or**

Example:

```
push_quantum (TRUE_TAG);
push_not (top_estack);    /* Pushes FALSE_TAG */
```

push_or

Declaration: void **push_or** (EStackIndex *i*, EStackIndex *j*)

Category(ies): Logic

Description: Pushes onto the estack the internally-simplified form of the logical “or” of the Boolean expressions indexed by *i* and *j*.

Inputs: *i*, *j* — Indices of the top tags of internally-simplified Boolean expressions or aggregates thereof.

Outputs: None

Assumptions: None

Side Effects: None

Availability: On AMS 2.02 and higher.

TI-89 / TI-92 Plus Differences: None

See Also: **or_onto_top, replace_top2_with_or, push_and, and_onto_top, replace_top2_with_and, push_not**

Example:

```
push_quantum (8u);
arg2 = top_estack;    /* Push variable x */
push_not (arg2);
push_or (top_estack, arg2);   /* Pushes a TRUE_TAG */
```

push_when

Declaration: void **push_when** (EStackIndex *i*)

Category(ies): Logic

Description: If the Boolean expression or all of its elements internally simplify to TRUE, then the internally-simplified equivalent of the “then” expression is pushed onto the estack.

Otherwise if the Boolean expression or any of its elements internally simplify to FALSE and there is an Else expression, then its internally-simplified equivalent is pushed.

Otherwise if there is an unknown expression, then its internally-simplified equivalent is pushed.

Otherwise the tail indexed by *i* is pushed, then a WHEN_TAG is pushed.

Inputs: *i* — Indexes a tail of 2, 3 or 4 successive expressions on top of an END_TAG: a Boolean expression or an aggregate thereof on top of a Then expression, optionally on top of an Else expression, optionally on top of an Unknown expression.

Outputs: None

Assumptions: None

Side Effects: None

Availability: On AMS 2.02 and higher.

TI-89 / TI-92 Plus Differences: None

See Also: **push_sign**

Example:

```
push_quantum (END_TAG);
push0 ();
push1 ();
push_quantum (TRUE_TAG);
push_when (top_estack);  /* Pushes a copy of the above-pushed one. */
```

remaining_conjuncts_index

Declaration: EStackIndex **remaining_conjuncts_index** (EStackIndex *i*)

Category(ies): Logic

Description: If *i* indexes an AND_TAG, returns the index of the deeper of its two operands. Otherwise returns index_true. Internally-simplified conjuncts have the most main Boolean factor shallowest.

Note that **index_true** is not ordinarily physically within the expression indexed by *i*.

Inputs: *i* — Indexes the top tag of an expression.

Outputs: None

Assumptions: None

Side Effects: None

Availability: On AMS 2.02 and higher.

TI-89 / TI-92 Plus Differences: None

See Also: **lead_conjunct_index, lead_disjunct_index, remaining_disjuncts_index**

Example:

```
remaining_conjuncts_index (“internalSimplified” ((b and a) and c))
```

Returns index (b and c).

```
remaining_conjuncts_index (x > 3)
```

Returns index_true.

```
push_quantum (8u);  /* Push variable x */
remaining_conjuncts_index (top_estack);  /* Returns index_true. */
```

remaining_disjuncts_index

Declaration: EStackIndex **remaining_disjuncts_index** (EStackIndex *i*)

Category(ies): Logic

Description: If *i* indexes an OR_TAG, returns the index of the deeper of its two operands. Otherwise returns index_false. Internally-simplified disjuncts have the most main Boolean term shallowest.

Note that **index_false** is not ordinarily physically within the expression indexed by *i*.

Inputs: *i* — Indexes the top tag of an internally-simplified expression.

Outputs: None

Assumptions: None

Side Effects: None

Availability: On AMS 2.02 and higher.

TI-89 / TI-92 Plus Differences: None

See Also: **lead_disjunct_index, lead_conjunct_index, remaining_conjuncts_index**

Example: If i indexes the internally-simplified expression
`(b or a) or c`
then **remaining_disjuncts_index**(i) returns index(b or c).

If i indexes the internally-simplified expression
`x > 3`
then **remaining_disjuncts_index**(i) returns index_false.

```
push_quantum (8u);  /* Push variable x */
remaining_disjuncts_index (top_estack);  /* Returns index_false. */
```

replace_top2_with_and

Declaration: void **replace_top2_with_and** (EStackIndex *i*)

Category(ies): Logic

Description: Replaces the top two expressions with their internally-simplified logical "and."

Inputs: *i* — Indexes the top tag of the internally-simplified Boolean expression **next_expression_index** (**top_estack**).

Outputs: None

Assumptions: The top two expressions are internally-simplified Boolean expressions.

Side Effects: None

Availability: On AMS 2.02 and higher.

TI‑89 / TI‑92 Plus Differences: None

See Also: **and_onto_top, push_and**

Example:

```
/* Replaces the top two expressions with a variable x. */
push_quantum (TRUE_TAG);
i = top_estack;
push_quantum (8u);
replace_top2_with_and (i);
```

replace_top2_with_or

Declaration: void **replace_top2_with_or** (EStackIndex *i*)

Category(ies): Logic

Description: Replaces the top two expressions with their internally-simplified logical “or.”

Inputs: *i* — Indexes the top tag of the internally-simplified Boolean expression **next_expression_index** (**top_estack**).

Outputs: None

Assumptions: The top two expressions are internally-simplified Boolean expressions.

Side Effects: None

Availability: On AMS 2.02 and higher.

TI-89 / TI-92 Plus Differences: None

See Also: **or_onto_top, push_or**

Example:

```
/* Replaces the top two expressions with a variable x. */
push_quantum (FALSE_TAG);
i = top_estack;
push_quantum (8u);
replace_top2_with_or (i);
```

Appendix A: System Routines — Math

See Also:

are_units_consistent

Declaration: Boolean **are_units_consistent** (EStackIndex *i*, EStackIndex *j*)

Category(ies): Math

Description: Tests whether two expressions have consistent units.

Inputs: *i* — EStackIndex of an expression.

j — EStackIndex of an expression.

Outputs: Returns TRUE if *i* and *j* have consistent units, otherwise returns FALSE.

Assumptions: None

Side Effects: May cause estack expansion or heap compression.

Availability: On AMS 2.00 and higher.

TI-89 / TI-92 Plus Differences: None

See Also: None

Example:

If r indexes the bolded tag in the following expression 1.2_m as follows
0x40 0x00 0x12 0x00 0x00 0x00 0x00 0x00 0x00 FLOAT_TAG 0 _ m 0 **MULTIPLY_TAG**

and s indexes the bolded tag in the following expression 3.5791_ft as follows
0x40 0x00 0x35 0x79 0x10 0x00 0x00 0x00 0x00 FLOAT_TAG 0 _ f t 0 **MULTIPLY_TAG**

then

```
are_units_consistent (r, s);
```

will return TRUE, since both expression are length measurements.

did_push_anti_deriv

Declaration: Boolean **did_push_anti_deriv** (EStackIndex *i*, EStackIndex *vi*, Boolean *accept_only_closed_form*)

Category(ies): Math

Description: Returns TRUE if it pushed an antiderivative of the expression indexed by *i* with respect to the variable indexed by *vi*.

Inputs:

i	—	Index of the top tag of an internally-simplified expression or an aggregate thereof.
vi	—	Index of the top tag of a variable.
accept_only_closed_form	—	TRUE if only completely closed-form antiderivatives are allowed.

Outputs: Returns TRUE if it pushed an antiderivative of the expression indexed by *i* with respect to the variable indexed by *vi*. Otherwise returns FALSE.

Assumptions: None

Side Effects: None

Availability: On AMS 2.02 and higher.

TI-89 / TI-92 Plus Differences: None

See Also: **push_def_int, push_nint**

Example:

```
void push_anti_derivative (EStackIndex i, EStackIndex vi)
{  (void)did_push_anti_deriv (i, vi, FALSE);
}
```

did_push_approx_inflection_point

Declaration: Boolean **did_push_approx_inflection_point** (EStackIndex *i*, EStackIndex *vi*, Float *a*, Float *b*)

Category(ies): Math

Description: Returns TRUE if it pushed an approximate inflection point of the expression indexed by *i* with respect to the variable indexed by *vi* between lower bound *a* and upper bound *b*.

Inputs:

i — Index of the top tag of an internally-simplified expression or an aggregate thereof.

vi — Index of the top tag of a variable.

a, *b* — Float bounds with $a < b$.

Outputs: Returns TRUE if it pushed an approximate inflection point of the expression indexed by *i* with respect to the variable indexed by *vi* between lower bound *a* and upper bound *b*. Otherwise returns FALSE.

Assumptions: None

Side Effects: None

Availability: On AMS 2.02 and higher.

TI-89 / TI-92 Plus Differences: None

See Also: **push_min, push_max**

Example:

```
/* Returns TRUE after pushing tagged float 0.0 onto top of the estack. */
push_quantum_as_nonnegative_int (3u);
push_quantum (8u);  /* Push variable x */
vi = top_estack;
push_quantum (EXPONENTIATION_TAG);  /* x^3 */
did_push_approx_inflection_point (top_estack, vi, -1.0, 1.0);
```

did_push_series

Declaration: Boolean **did_push_series** (EStackIndex *i*, EStackIndex *ki*, EStackIndex *k*, EStackIndex *j*, Boolean *stop_after_1st_non0_term*)

Category(ies): Math

Description: Returns TRUE if it successfully pushes onto the estack the *kth*-order Taylor series expansion of expression *i* with respect to variable *ki* expanded about *ki* = expression *j*.

j == NULL_INDEX makes the expansion point 0.

Stops early after the first nonzero term if *stop_after_1st_non0_term* is TRUE. If invoked via **push_internal_simplify**, *ki* and *i* are simplified to deepest variable. However, if the deepest variable value of *ki* has a such-that or STO▸ value, that value is substituted for the deepest variable value after computing the symbolic series.

For example, taylor(e^x, x, 3, 0) | x = 1 -> 8/3.

Inputs:

i	—	Index of the top tag of an internally-simplified expression, algebraic comparison, or an aggregate thereof.
ki	—	Index of the top tag of a variable.
k	—	Index of a non-negative number.
j	—	NULL_INDEX or the index of the top tag of an algebraic expression.
stop_after_1st_non0_term	—	TRUE if function should stop after the first nonzero term.

Outputs: Returns TRUE if it successfully pushes onto the estack the *kth*-order Taylor series expansion of expression *i* with respect to variable *ki* expanded about *ki* = expression *j*. Otherwise returns FALSE.

Assumptions: None

Side Effects: None

Availability: On AMS 2.02 and higher.

TI-89 / TI-92 Plus Differences: None

(continued)

did_push_series *(continued)*

See Also: **push_lim, push_substitute_simplify, push_subst_no_simp**

Example:

```
/* Returns TRUE after pushing a tagged integer –1 onto the estack. */
push_quantum (8u);  /* Push variable x */
ki = top_estack;
push_sum (ki, IntegerMinus1Index);
i = top_estack;
did_push_series (i, ki, Integer2Index, NULL_INDEX, TRUE);
```

push_1st_derivative

Declaration: void **push_1st_derivative** (EStackIndex *i*, EStackIndex *j*)

Category(ies): Math

Description: Pushes onto the estack the first derivative of the expression indexed by *i* with respect to the variable indexed by *j*.

If invoked via **push_internal_simplify**, *j* and *i* are simplified to deepest variable. However, if the deepest variable value of *j* has a such-that or STO▸ value, that value is substituted for the deepest variable value after computing the symbolic derivative.

For example, d(x^2, x) | x = 3 -> 6.

Inputs:

i — Indexes the top tag of an internally-simplified algebraic expression, comparison, or aggregate thereof.

j — Indexes the top tag of a variable.

Outputs: None

Assumptions: None

Side Effects: None

Availability: On AMS 2.02 and higher.

TI-89 / TI-92 Plus Differences: None

See Also: **push_nth_derivative**

Example:

```
void push_grad_tail (EStackIndex i, EStackIndex vars)
/* i indexes an algebraic expression.
   vars indexes a tail of variables.
   Pushes the rectangular Cartesian gradient tail of i with respect to vars.
*/
{  if (END_TAG == ESTACK (vars))
      push_quantum (END_TAG);
   else
   {  push_grad_tail (i, next_expression_index (vars));
      push_1st_derivative (i, vars);
   }
}
```

push_abs

Declaration: void **push_abs** (EStackIndex *k*)

Category(ies): Math

Description: Pushes onto the estack the internally-simplified absolute value of the expression indexed by *k*.

For example, if *k* indexes an IM_RE_TAG, pushes the square root of the sum of the squares of the two expressions immediately below *k*.

Inputs: *k* — Indexes the top tag of an internally-simplified algebraic expression, comparison, or aggregate thereof.

Outputs: None

Assumptions: None

Side Effects: None

Availability: On AMS 2.02 and higher.

TI-89 / TI-92 Plus Differences: None

See Also: **push_sign**

Example:

```
push_quantum_as_nonnegative_int (3u);
real_part = top_estack;
push_quantum_as_nonnegative_int (4u);
replace_top2_with_imre (real_part);  /* Pushes an IM_RE_TAG */
push_abs (top_estack);  /* Pushes a tagged integer 5 */
```

push_acos

Declaration: void **push_acos** (EStackIndex *k*)

Category(ies): Math

Description: Pushes onto the estack the internally-simplified principal branch of the inverse cosine of the expression indexed by *k,* measured in radians. Thus the real part of the result is in the range [0, π].

When *k* indexes a square numeric matrix, pushes the Float matrix acos computed via **acos**(eigenvalues(matrix)).

Inputs: *k* — Indexes the top tag of an internally-simplified algebraic expression, comparison, or aggregate thereof.

Outputs: None

Assumptions: None

Side Effects: None

Availability: On AMS 2.02 and higher.

TI-89 / TI-92 Plus Differences: None

See Also: **push_asin, push_atan, push_phase**

Example:

```
push_negate_quantum_as_negint(1u);  /* Pushes tagged integer -1 */
push_acos (top_estack);             /* Pushes PI_TAG or tagged 3.14159 . . . */
```

push_acosh

Declaration: void **push_acosh** (EStackIndex *k*)

Category(ies): Math

Description: Pushes onto the estack the internally-simplified principal branch of the inverse hyperbolic cosine of the expression indexed by *k*. Thus the real part of the result is in the range [0, ∞].

When *k* indexes a square numeric matrix, pushes the Float matrix acosh computed via **acosh**(eigenvalues(matrix)).

Inputs: *k* — Indexes the top tag of an internally-simplified algebraic expression, comparison, or aggregate thereof.

Outputs: None

Assumptions: None

Side Effects: None

Availability: On AMS 2.02 and higher.

TI-89 / TI-92 Plus Differences: None

See Also: **push_asinh, push_atanh**

Example:

```
push_quantum_as_nonnegative_int(1u);  /* Pushes tagged integer 1 */
push_acosh (top_estack);              /* Pushes tagged 0 or 0.0 */
```

push_asin

Declaration: void **push_asin** (EStackIndex *k*)

Category(ies): Math

Description: Pushes onto the estack the internally-simplified principal branch of the inverse sine of the expression indexed by *k,* measured in radians. Thus the real part of the result is in the range [-π/2, π/2].

When *k* indexes a square numeric matrix, pushes the Float matrix asin computed via **asin**(eigenvalues(matrix)).

Inputs: *k* — Indexes the top tag of an internally-simplified algebraic expression, comparison, or aggregate thereof.

Outputs: None

Assumptions: None

Side Effects: None

Availability: On AMS 2.02 and higher.

TI-89 / TI-92 Plus Differences: None

See Also: **push_acos, push_atan, push_phase**

Example:

```
push_quantum_as_nonnegative_int(1u);  /* Pushes tagged integer 1 */
push_asin (top_estack);               /* Pushes symbolic or Float pi/2 */
```

push_asinh

Declaration: void **push_asinh** (EStackIndex *k*)

Category(ies): Math

Description: Pushes onto the estack the internally-simplified inverse hyperbolic sine of the expression indexed by *k*. When *k* indexes a square numeric matrix, pushes the Float matrix asinh computed via **asinh**(eigenvalues(matrix)).

Inputs: *k* — Indexes the top tag of an internally-simplified algebraic expression, comparison, or aggregate thereof.

Outputs: None

Assumptions: None

Side Effects: None

Availability: On AMS 2.02 and higher.

TI-89 / TI-92 Plus Differences: None

See Also: **push_acosh, push_atanh**

Example:

```
push_quantum_as_nonnegative_int(0u);  /* Pushes tagged integer 0 */
push_asinh (top_estack);              /* Pushes tagged 0 or 0.0 */
```

push_atan

Declaration: void **push_atan** (EStackIndex *k*)

Category(ies): Math

Description: Pushes onto the estack the internally-simplified principal branch of the inverse tangent of the expression indexed by *k,* measured in radians. Thus the real part of the result is in the range [-π/2, π/2].

When *k* indexes a square numeric matrix, pushes the Float matrix atan computed via **atan**(eigenvalues(matrix)).

Inputs: *k* — Indexes the top tag of an internally-simplified algebraic expression, comparison, or aggregate thereof.

Outputs: None

Assumptions: None

Side Effects: None

Availability: On AMS 2.02 and higher.

TI-89 / TI-92 Plus Differences: None

See Also: **push_asin, push_acos, push_phase**

Example:

```
push_quantum_as_nonnegative_int(1u);  /* Pushes tagged integer 1 */
push_atan (top_estack);               /* Pushes symbolic or Float pi/4 */
```

push_atanh

Declaration: void **push_atanh** (EStackIndex *k*)

Category(ies): Math

Description: Pushes onto the estack the internally-simplified inverse hyperbolic tangent of the expression indexed by *k*.

When *k* indexes a square numeric matrix, pushes the Float matrix atanh computed via **atanh**(eigenvalues(matrix)).

Inputs: *k* — Indexes the top tag of an internally-simplified algebraic expression, comparison, or aggregate thereof.

Outputs: None

Assumptions: None

Side Effects: None

Availability: On AMS 2.02 and higher.

TI-89 / TI-92 Plus Differences: None

See Also: **push_acosh, push_asinh**

Example:

```
push_quantum_as_nonnegative_int(1u);  /* Pushes tagged integer 1 */
push_atanh(top_estack);               /* Pushes PLUS_INFINITY_TAG */
```

push_ceiling

Declaration: void **push_ceiling** (EStackIndex *k*)

Category(ies): Math

Description: Pushes onto the estack the internally-simplified ceiling of the expression indexed by *k*.

Inputs: *k* — Indexes the top tag of an internally-simplified algebraic expression or aggregate thereof.

Outputs: None

Assumptions: None

Side Effects: None

Availability: On AMS 2.02 and higher.

TI-89 / TI-92 Plus Differences: None

See Also: **push_floor, push_integer_part**

Example:

```
push_Float (1.5);
push_ceiling (top_estack);  /* Pushes a tagged float 2.0 */
```

push_comb

Declaration: void **push_comb** (EStackIndex *i*, EStackIndex *j*)

Category(ies): Math

Description: Let *i* index expression z, and let *j* index expression k. Pushes onto the estack: 0 for integer k < 0; otherwise z ! / (k ! ∗ (z - k!)).

For integer k >= 0 this is z ∗ (z - 1)* . . . *(z - k + 1) / k!. If z is also integer and z >= k, this is the number of combinations of z items taken k at a time.

Reference: Graham, Knuth & Patashnik, “Concrete Mathematics”, section 5.1, Addison-Wesley.

Inputs: *i*, *j* — Indices of the top tags of internally-simplified algebraic expressions or aggregates thereof.

Outputs: None

Assumptions: None

Side Effects: None

Availability: On AMS 2.02 and higher.

TI-89 / TI-92 Plus Differences: None

See Also: **push_perm, push_factorial**

Example:

```
push_quantum_as_nonnegative_int (4u);
i = top_estack;
push_quantum_as_nonnegative_int (2u);
push_comb (i, top_estack);  /* Pushes a tagged integer 6 */
```

push_comdenom

Declaration: void **push_comdenom** (EStackIndex *i*, EStackIndex *vi*)

Category(ies): Math

Description: Pushes an expression equivalent to the one indexed by *i*, with terms combined over a common denominator. If *vi* is NULL_INDEX, the numerator and denominator are fully expanded through all variables, using distributed form. Otherwise the numerator and denominator are fully expanded with distributed form through vi. Expansion is not forced with respect to any variable or kernel if *vi* is more main than any variable or kernel in *i*.

Inputs:
- *i* — Indexes an internally-simplified algebraic expression, algebraic comparison, or aggregate thereof.
- *vi* — NULL_INDEX or indexes an internally-simplified variable or kernel.

Outputs: None

Assumptions: None

Side Effects: None

Availability: On AMS 2.02 and higher.

TI-89 / TI-92 Plus Differences: None

See Also: **push_standardize, push_factor, push_expand, push_make_proper**

Example:

```
void push_trig_expand (EStackIndex i)
/* i indexes an expression. Temporarily sets global trigonometric mode to
   SET_EXPAND_TRIG then pushes a corresponding simplified version of the
   expression onto the estack.
*/
{  Access_AMS_Global_Variables;
   EStackIndex old_top = top_estack;
   CONTROL_BITS old_NG_control = NG_control;

   SET_EXPAND_TRIG;

   push_internal_simplify (i);
   i = top_estack;
   push_comdenom (i, NULL_INDEX);
   NG_control = old_NG_control;
   delete_between (old_top, i);
}
```

push_conj

Declaration: void **push_conj** (EStackIndex *k*)

Category(ies): Math

Description: Pushes onto the estack the internally-simplified complex conjugate of the expression indexed by *k*.

Inputs: *k* — Index of the top tag of an internally-simplified algebraic expression, algebraic comparison, or aggregate thereof.

Outputs: None

Assumptions: None

Side Effects: None

Availability: On AMS 2.02 and higher.

TI-89 / TI-92 Plus Differences: None

See Also: **push_re, push_im, push_abs, push_phase, push_sign**

Example:

```
push_quantum_as_nonnegative_int (4u);
real_part = top_estack;
push_quantum_as_nonnegative_int (2u);
replace_top2_with_imre (real_part);
push_conj (top_estack);  /* Pushes 4-2i */
```

push_constant_terms

Declaration: void **push_constant_terms** (EStackIndex *k*)

Category(ies): Math

Description: Pushes onto the estack the sum of all syntactic terms of the expression indexed by *k* that do not contain variables.

If there are no constant terms, pushes Float0 if IS_ARITH_APPROX is TRUE. Otherwise pushes Integer0.

Inputs: *k* — Index of the top tag of an internally-simplified algebraic expression.

Outputs: None

Assumptions: None

Side Effects: None

Availability: On AMS 2.02 and higher.

TI-89 / TI-92 Plus Differences: None

See Also: **push_nonconstant_terms, push_dependent_terms, push_independent_terms, index_numeric_term,**

Example:

```
push_quantum (PI_TAG);
add1_to_top ();                    /* partial sum = top_estack */
push_quantum (8u);                 /* push variable x */
replace_top2_with_sum (foo);       /* top_estack -> x + pi + 1 */
push_constant_terms (top_estack);  /* Pushes pi + 1 */
```

push_cos

Declaration: void **push_cos** (EStackIndex *k*)

Category(ies): Math

Description: Pushes onto the estack the internally-simplified cosine of the expression indexed by *k*. The only trigonometric tag that internally-simplified results can contain is SIN2_TAG or (if IS_COLLECT_KERNELS) TAN_RAD_TAG, whose arguments are always in radians.

When *k* indexes a square numeric matrix, pushes the Float matrix cosine computed via **cos**(eigenvalues(matrix)).

Inputs: *k* — Index of the top tag of an internally-simplified algebraic expression, algebraic comparison, or aggregate thereof, with angles measured in radians.

Outputs: None

Assumptions: None

Side Effects: None

Availability: On AMS 2.02 and higher.

TI-89 / TI-92 Plus Differences: None

See Also: **push_sin, push_tan, push_sin2**

Example:

```
void push_sec (EStackIndex i)
/* Pushes onto the estack the secant of the expression indexed by i. */
{  if (LIST_TAG == ESTACK (i))
   {  map_tail (push_sec, i - 1u);
      push_quantum (LIST_TAG);
   }
   else
   {  push_cos (i);
      replace_top_with_reciprocal ();
   }
}
```

push_cosh

Declaration: void **push_cosh** (EStackIndex *k*)

Category(ies): Math

Description: Pushes onto the estack the internally-simplified hyperbolic cosine of the expression indexed by *k*.

When *k* indexes a square numeric matrix, pushes the Float matrix hyperbolic cosine computed via **cosh**(eigenvalues(matrix)).

Inputs: *k* — Index of the top tag of an internally-simplified algebraic expression, algebraic comparison, or aggregate thereof.

Outputs: None

Assumptions: None

Side Effects: None

Availability: On AMS 2.02 and higher.

TI-89 / TI-92 Plus Differences: None

See Also: **push_sinh, push_tanh, push_acosh, push_asinh, push_atanh**

Example:

```
void push_sech (EStackIndex i)
/* Pushes onto the estack the hyperbolic secant of the expression indexed by i. */
{  if (LIST_TAG == ESTACK (i))
   {  map_tail (push_sech, i - 1u);
      push_quantum (LIST_TAG);
   }
   else
   {  push_cosh (i);
      replace_top_with_reciprocal ();
   }
}
```

push_def_int

Declaration: void **push_def_int** (EStackIndex *i*, EStackIndex *vi*, EStackIndex *j*, EStackIndex *k*)

Category(ies): Math

Description: Pushes onto the estack the definite integral of the expression indexed by *i*, integrated with respect to the variable indexed by *vi*, from the lower limit indexed by *j* to the upper limit indexed by *k*. If invoked via **push_internal_simplify**, *vi* and *i* are simplified to deepest variable. Moreover, *i* is simplified under the influence of a temporary such that *vi* > *j* and *vi* < *k* when *j* < *k*.

Inputs:

i — Index of the top tag of an internally-simplified algebraic expression, a comparison, or an aggregate thereof.

vi — Index of the top tag of a variable.

j, k — Indices of the top tags of internally-simplified algebraic expressions or aggregates thereof.

Outputs: None

Assumptions: None

Side Effects: None

Availability: On AMS 2.02 and higher.

TI-89 / TI-92 Plus Differences: None

See Also: **did_push_anti_deriv, push_nint, push_arclen**

Example:

```
void push_arclen
               (EStackIndex i, EStackIndex vi, EStackIndex j, EStackIndex k)
{  Access_AMS_Global_Variables;
   EStackIndex m, old_top = top_estack;
   push_quantum_as_nonnegative_int (2u);
   m = top_estack;
   push_1st_derivative (i, vi);
   replace_top2_with_pow (m);
   add1_to_top ();
   i = top_estack;
   push_sqrt (i);
   delete_between (old_top, i);
   i = top_estack;
   push_def_int (i, vi, j, k);
   delete_between (old_top, i);
}
```

push_degrees

Declaration: void **push_degrees** (EStackIndex *i*, EStackIndex *j*, EStackIndex *k*)

Category(ies): Math

Description: Converts the input values, interpreted as degrees, minutes, and seconds, to the currently selected angle measure.

Inputs:
i — EStackIndex of internal tokenized degrees value.
j — EStackIndex of internal tokenized minutes value.
k — EStackIndex of internal tokenized seconds value.

Outputs: Pushes the internal tokenized form of the result of converting the specified degrees, minutes, and seconds to the currently selected angle measure — radians or decimal degrees.

Assumptions: Input values must be numbers, numeric symbols, or expressions that result in numbers or numeric symbols.

Side Effects: May cause estack expansion, heap compression, or throw errors.

Availability: On AMS 1.05 and higher.

TI-89 / TI-92 Plus Differences: None

See Also: **push_radians**

Example:

If the current angle setting is RADIANS, i indexes 30, j indexes 20, and k indexes 0, then

```
push_degrees (i, j, k);
```

pushes the value 91 ∗ π/540 which is the equivalent number of radians onto the estack.

If the current angle setting is DEGREES, i indexes 30, j indexes 20, and k indexes 0, then

```
push_degrees (i, j, k);
```

pushes the value 91/3 which is the equivalent number of degrees onto the estack.

push_dot_exponentiate

Declaration: void **push_dot_exponentiate** (EStackIndex *i*, EStackIndex *j*)

Category(ies): Math, Lists and Matrices

Description: Pushes the internally-simplified result of (expression *i*)^(expression *j*) onto the estack. When the base is a square matrix, this is done by merely distributing the exponent over the elements of the base rather than by (possibly repeated) matrix multiplication and/or inversion.

Inputs: *i*, *j* — Index the top tags of internally-simplified algebraic expressions, algebraic comparisons, or aggregates thereof.

Outputs: None

Assumptions: None

Side Effects: None

Availability: On AMS 2.02 and higher.

TI-89 / TI-92 Plus Differences: None

See Also: **push_exponentiate, raise_to_top, replace_top2_with_pow, push_sqrt, push_square, push_reciprocal, replace_top_with_reciprocal**

Example:

```
/* Push random 2 by 2 matrix: */
push_randmat (Integer2Index, Integer2Index);
/* Push the matrix of the squares of the elements */
push_dot_exponentiate (top_estack, Integer2Index);
```

push_exp

Declaration: void **push_exp** (EStackIndex *i*)

Category(ies): Math

Description: Pushes onto the estack the simplified exponential function of the expression indexed by *i*.

Inputs: *i* — Index of the top tag of an internally-simplified expression.

Outputs: None

Assumptions: *i* points to the estack or some other locked block.

Side Effects: May cause heap compression or throw an error.

Availability: On AMS 2.02 and higher.

TI-89 / TI-92 Plus Differences: None

See Also: **push_exponentiate, raise_to_top, replace_top2_with_pow**

Example:

```
push_quantum_as_nonnegative_int (0u);  /* Push tagged integer 0 */
push_exp (top_estack);                 /* Pushes tagged integer 1 */
```

push_expand

Declaration: void **push_expand** (EStackIndex *i*, EStackIndex *ki*, Boolean *use_part_frac*)

Category(ies): Math

Description: Pushes onto the estack the internally-simplified equivalent of the expression indexed by *i*, expanded with respect to the variable or kernel indexed by *ki*, or with respect to all variables and kernels if *ki* is equal to NULL_INDEX. If *use_part_frac* is FALSE, rational expressions are not expanded beyond proper fractions. Otherwise, partial fraction expansion is also done, using factorization over the real rather than complex numbers.

Inputs:

i — Indexes the top tag of an internally-simplified algebraic expression, algebraic comparison, or aggregate thereof.

ki — Indexes NULL_INDEX or the top tag of an internally-simplified variable or a kernel.

use_part_frac — TRUE if partial fraction expression can be used.

Outputs: None

Assumptions: None

Side Effects: None

Availability: On AMS 2.02 and higher.

TI-89 / TI-92 Plus Differences: None

See Also: None

Example:

```
push_quantum_as_nonnegative_int (2u);
exponent = top_estack;
push_quantum (8u);                        /* Push variable x */
add1_to_top ();                           /* top_estack -> x + 1 */
replace_top2_with_pow (exponent);         /* top_estack -> (x + 1)^2 */
push_expand (top_estack, NULL_INDEX, FALSE);  /* push x^2 + 2x + 1 */
```

push_exponentiate

Declaration: void **push_exponentiate** (EStackIndex *i*, EStackIndex *j*)

Category(ies): Math

Description: Pushes the internally-simplified result of (expression *i*)^(expression *j*) onto the estack. If expression *i* is a square matrix: pushes the same-size identity matrix if *j* indexes a zero; pushes the iterated matrix product if *j* indexes a positive whole number; pushes the inverse matrix or its iterated matrix product if *j* indexes a negative whole number.

Inputs: *i, j* — Indices of the top tags of internally-simplified algebraic expressions, algebraic comparisons, or aggregates thereof.

Outputs: None

Assumptions: None

Side Effects: None

Availability: On AMS 2.02 and higher.

TI-89 / TI-92 Plus Differences: None

See Also: **push_dot_exponentiate, raise_to_top, replace_top2_with_pow, push_sqrt, push_square, push_reciprocal, replace_top_with_reciprocal**

Example:

```
void push_distrib_base_over_tail (EStackIndex bas, EStackIndex tail)
/* tail indexes a sequence of expressions terminated by END_TAG.
   Pushes onto the estack a similar sequence of expression bas raised to
   the expressions in tail.
*/
{  if (END_TAG == ESTACK (tail))
      push_quantum (END_TAG);
   else
      {  push_distrib_base_over_tail (bas, next_expression_index (tail));
         push_exponentiate (bas, tail);
      }
}
```

push_extended_prod

Declaration: void **push_extended_prod** (EStackIndex *i*, EStackIndex *ki*, EStackIndex *j*, EStackIndex *k*)

Category(ies): Math

Description: Pushes onto the estack the product of the expression indexed by *i*, for the variable indexed by *ki* ranging from the value indexed by *j* through the value indexed by *k*. The prodand must be UNSIMPLIFIED for examples such as π (rand(9), . . .) to have the intended effect.

Inputs:

i — Indexes the top tag of a prodand, which can be an algebraic expression, an algebraic comparison, or an aggregate thereof.

ki — Indexes the top tag of a variable.

j, *k* — Indices of algebraic expressions.

Outputs: None

Assumptions: None

Side Effects: None

Availability: On AMS 2.02 and higher.

TI-89 / TI-92 Plus Differences: None

See Also: None

Example:

```
/* Pushes 24u FACTORIAL_TAG  (n!) onto the estack */
push_quantum (24u);
upper_limit = top_estack;    /* Push variable n */
push1();
lower_limit = top_estack;
push_quantum (21u);
index_var = top_estack;      /* Push variable k */
push_extended_prod (index_var, index_var, lower_limit, upper_limit);
```

push_factor

Declaration: void **push_factor** (EStackIndex *i*, EStackIndex *vf*, FCTR_AMOUNT *fctr_amount*)

Category(ies): Math

Description: If *i* indexes a numeric tag then:

If POLY_OR_NMBR_FCTR is equal to *fctr_amount* and NULL_INDEX is equal to *vf* or SQFREE_AND_NMBR_FCTR <= *fctr_amount*, then the factored equivalent is pushed; otherwise the number is pushed. Otherwise the expression is factored through *vf*, or with respect to all of its variables if *vf* is NULL_INDEX.

Complex factors are sought only if IS_DOMAIN_COMPLEX.

Consider 6x^6 - 6x^4 - 6x^2 + 6 with *vf* = NULL_INDEX and IS_DOMAIN_REAL:

REAL_FCTRS_ONLY ->	6 (x - 1)^2 (x + 1)^2
POLY_FCTR ->	6 (x - 1)^2 (x + 1)^2 (x^2 + 1)
POLY_OR_NMBR_FCTR ->	6 (x - 1)^2 (x + 1)^2 (x^2 + 1)
SQFREE_AND_NMBR_FCTR ->	2 ∗ 3 (x^2 - 1)^2 (x^2 + 1)
POLY_AND_NMR_FCTR ->	2 ∗ 3 * (x - 1)^2 (x + 1)^2 (x^2 + 1)

Inputs:

i — Indexes the top tag of an algebraic expression, an algebraic comparison, or an aggregate thereof.

vf — Equal to NULL_INDEX or indexes the top tag of a variable.

fctr_amount — One of REAL_FCTRS_ONLY, POLY_FCTR, POLY_OR_NMBR_FCTR, SQFREE_AND_NMBR_FCTR, or POLY_AND_NMBR_FCTR.

Outputs: None

Assumptions: None

Side Effects: None

Availability: On AMS 2.02 and higher.

TI-89 / TI-92 Plus Differences: None

(continued)

push_factor *(continued)*

See Also: None

Example:

```
push_quantum_as_nonnegative_int (2u);
exponent = top_estack;
push_quantum (8u);                    /* Push variable x */
replace_top2_with_pow;                /* top_estack -> x^2 */
subtract1_from_top ();                /* top_estack -> x^2 - 1 */
push_factor (top_estack, NULL_INDEX, POLY_FCTR);  /* Pushes (x - 1)(x + 1) */
```

push_factorial

Declaration: void **push_factorial** (EStackIndex *i*)

Category(ies): Math

Description: Pushes the internally-simplified factorial of the expression indexed by *i*.

Although z! = Gamma(z - 1) for all complex z, where Gamma is the Euler Gamma function, this implementation of factorial computes a numeric result only for non-negative integer z.

Inputs: *i* — Indexes the top tag of an algebraic expression, an algebraic comparison, or an aggregate thereof.

Outputs: None

Assumptions: None

Side Effects: None

Availability: On AMS 2.02 and higher.

TI‑89 / TI‑92 Plus Differences: None

See Also: None

Example: If *i* indexes a negative integer, then **push_factorial**(*i*) pushes PLUS_OR_MINUS_INFINITY_TAG.

If *i* indexes IM_RE_TAG or PI_TAG, then **push_factorial**(*i*) pushes FACTORIAL_TAG on top of a copy of the expression indexed by *i*.

```
push_quantum_as_nonnegative_int (3u);
push_factorial (top_estack);  /* Pushes a tagged integer 6 */
```

push_floor

Declaration: void **push_floor** (EStackIndex *k*)

Category(ies): Math

Description: Pushes onto the estack the internally-simplified floor of the expression indexed by *k*. For unreal arguments x + *i* * y with x and y real, the result is floor(x) + *i* * floor(y).

Example: floor(-2.1) -> -3

Inputs: *k* — Indexes the top tag of an algebraic expression or an aggregate thereof.

Outputs: None

Assumptions: None

Side Effects: None

Availability: On AMS 2.02 and higher.

TI-89 / TI-92 Plus Differences: None

See Also: None

Example:

```
void push_ceiling (EStackIndex i)
/* i indexes an algebraic expression u or an aggregate thereof.
   Pushes ceiling(u) onto the estack.
   Uses the identity  ceiling(u) = -floor (-u).
*/
{  Access_AMS_Global_Variables;
   EStackIndex old_top = top_estack;
   push_negate (i);
   i = top_estack;
   push_floor (i);
   negate_top ();
   delete_between (old_top, i);
}
```

push_fractional_part

Declaration: void **push_fractional_part** (EStackIndex *k*)

Category(ies): Math

Description: Pushes onto the estack the internally-simplified fractional part of the expression indexed by *k*. For unreal arguments x + *i* * y with x and y real, the result is fPart(x) + *i* * fPart(y).

Inputs: *k* — Indexes the top tag of an algebraic expression or an aggregate thereof.

Outputs: None

Assumptions: None

Side Effects: None

Availability: On AMS 2.02 and higher.

TI-89 / TI-92 Plus Differences: None

See Also: None

Example:

```
push_Float (-2.1);
push_fractional_part (top_estack);  /* Pushes –0.1 */
```

push_gcd_then_cofactors

Declaration: EStackIndex **push_gcd_then_cofactors** (EStackIndex *i*, EStackIndex *j*, EStackIndex * *p*)

Category(ies): Math

Description: Pushes onto the estack the greatest common denominator (gcd) of *i* and *j*, then (expression *j*)/gcd then (expression *i*)/gcd. Stores the index of (expression *j*)/gcd in * *p* then returns the index of the gcd.

Inputs: *i*, *j* — Indices of the top tags of algebraic expressions.

p — The address of an EStackIndex.

Outputs: Index of (expression *j*)/gcd.

Assumptions: None

Side Effects: None

Availability: On AMS 2.02 and higher.

TI-89 / TI-92 Plus Differences: None

See Also: None

Example:

```
EStackIndex index_push_monic_or_prim_pair (EStackIndex i, EStackIndex j)
/* If i indexes a Float, pushes 1.0 then the ratio of expressions j and i.
   Otherwise pushes i/gcd(i, j) then j/gcd(i, j).
   In either case, returns the index of the deeper pushed value.
*/
{  Access_AMS_Global_Variables;
   EStackIndex k;
   if (FLOAT_TAG == ESTACK (i))
      {  push_expression (Float1Index);
         k = top_estack;
         push_ratio (j, i);
      }
   else
      {  EStackIndex old_top = top_estack;
         EStackDisplacement del =
            deleted_between (old_top, push_gcd_then_cofactors (j, i, &k));
         k -= del;
      }
   return k;
}
```

push_im

Declaration: void **push_im** (EStackIndex *k*)

Category(ies): Math

Description: Pushes onto the estack the internally-simplified imaginary part of the expression indexed by *k*.

Inputs: *k* — Index of the top tag of an internally-simplified algebraic expression, algebraic comparison, or aggregate thereof.

Outputs: None

Assumptions: None

Side Effects: None

Availability: On AMS 2.02 and higher.

TI-89 / TI-92 Plus Differences: None

See Also: **push_re, push_phase, push_abs, push_conj**

Example:

```
push_Float (2.3);
real_part = top_estack;
push_Float (3.5);
replace_top2_with_imre (real_part);
push_im (top_estack);   /* Pushes tagged float 3.5 */
```

push_integer_part

Declaration: void **push_integer_part** (EStackIndex *k*)

Category(ies): Math

Description: Pushes onto the estack the internally-simplified integer part of the expression indexed by *k*. For unreal arguments x + i * y with x and y real, the result is iPart(x) + i * iPart(y).

Inputs: *k* — Indexes the top tag of an algebraic expression or an aggregate thereof.

Outputs: None

Assumptions: None

Side Effects: None

Availability: On AMS 2.02 and higher.

TI-89 / TI-92 Plus Differences: None

See Also: **push_integer_quotient, push_integer_remainder, push_floor, push_mod, push_ceiling**

Example:

```
push_Float (-2.1);
push_integer_part (top_estack);  /* Pushes –2.0 */
```

push_integer_quotient

Declaration: void **push_integer_quotient** (EStackIndex *i*, EStackIndex *j*)

Category(ies): Math

Description: Pushes onto the estack the internally-simplified integer quotient intDiv of *i* and *j*, truncated toward zero. For all integers m and $n \neq 0$, $m = n * \text{intDiv}(m, n) + \text{remain}(m, n)$.

Inputs: *i*, *j* — Indices of the top tags of internally-simplified algebraic expressions or aggregates thereof.

Outputs: None

Assumptions: None

Side Effects: None

Availability: On AMS 2.02 and higher.

TI-89 / TI-92 Plus Differences: None

See Also: **push_integer_remainder, push_integer_part, push_floor, push_mod, push_ceiling**

Example:

```
push_quantum_as_nonnegative_int (3u);
j = top_estack;
push_quantum_as_nonnegative_int (5u);
push_integer_quotient (top_estack, j);  /* Pushes a tagged integer 1 */
```

push_integer_remainder

Declaration: void **push_integer_remainder** (EStackIndex *i*, EStackIndex *j*)

Category(ies): Math

Description: Pushes onto the estack their internally-simplified integer remainder. For all integers m and n ≠ 0, m = n ∗ intDiv (m, n) + remain (m, n).

Inputs: *i*, *j* — Indices of the top tags of internally-simplified algebraic expressions or aggregates thereof.

Outputs: None

Assumptions: None

Side Effects: None

Availability: On AMS 2.02 and higher.

TI-89 / TI-92 Plus Differences: None

See Also: **push_integer_quotient, push_integer_part, push_floor, push_ceiling, push_mod**

Example:

```
push_quantum_as_nonnegative_int (3u);
j = top_estack;
push_quantum_as_nonnegative_int (5u);
push_integer_remainder (top_estack, j);  /* Pushes a tagged integer 2 */
```

push_left

Declaration: void **push_left** (EStackIndex *i*, EStackIndex *j*)

Category(ies): Math

Description: Returns the leftmost portion of the input argument.

Inputs: *i* — EStackIndex of a list, a string, or a relational expression.

j — EStackIndex of a non-negative integer or whole number float.

Outputs: Pushes the leftmost *j* elements of *i* onto the estack.

Assumptions: If *i* is a list or string, and *j* is NULL_INDEX, all of *i* is pushed onto the estack.

Side Effects: May cause estack expansion, heap compression, or throw errors if arguments are invalid.

Availability: On AMS 2.00 and higher.

TI-89 / TI-92 Plus Differences: None

See Also: **push_mid, push_right, push_rotate, push_shift**

(continued)

push_left *(continued)*

Example:

If m indexes the bolded tag in the list {a, 0, b, -1} as follows
END_TAG 1 1 NEGATIVE_INTEGER_TAG B_VAR_TAG 0 NONNEGATIVE_INTEGER_TAG A_VAR_TAG **LIST_TAG**

and n indexes the bolded tag in the integer 3 as follows
3 1 **NONNEGATIVE_INTEGER_TAG**

then
```
push_left (m, n);
```

pushes the list {a, 0, b} onto the estack such that **top_estack** points to the bolded tag as follows.
END_TAG B_VAR_TAG 0 NONNEGATIVE_INTEGER_TAG A_VAR_TAG **LIST_TAG**

If m indexes the bolded tag in the string “hello” as follows
0 h e l l o 0 **STR_DATA_TAG**

and n indexes the bolded tag in the tagged floating-point number 2. as follows
0x40 0x00 0x20 0x00 0x00 0x00 0x00 0x00 0x00 **FLOAT_TAG**

then
```
push_left (m, n);
```

pushes the string “he” onto the estack such that **top_estack** points to the bolded tag as follows.
0 h e 0 **STR_DATA_TAG**

If m indexes the bolded tag in the relational expression x < 3 as follows
3 1 NONNEGATIVE_INTEGER_TAG X_VAR_TAG **LT_TAG**

and n is NULL_INDEX

then
```
push_left (m, n);
```

pushes the left side of the expression which is x onto the estack such that **top_estack** points to the bolded tag as follows.
X_VAR_TAG

push_lim

Declaration: void **push_lim** (EStackIndex *i*, EStackIndex *ki*, EStackIndex *j*, EStackIndex *direction*)

Category(ies): Math

Description: Pushes the limit of expression *i*, as variable *ki* approaches expression *j.* Subject to overriding for consistency with a signed zero or infinity for point *j*, the limit is from the right if the expression indexed by *direction* is positive, versus from the left if negative, or from both directions otherwise. If invoked via **push_internal_simplify**, *ki* and *i* are simplified to deepest variable. Moreover, the simplification of *i* and the computation of the limit are done in the context of a "such that" constraining *ki* to a small (but finite) neighborhood of point *j*, from the direction(s) specified by direction.

Inputs:

i — Indexes the top tag of an internally-simplified algebraic expression, algebraic comparison, or aggregate thereof.

ki — Indexes the top tag of a variable.

j, *direction* — Indexes to internally-simplified algebraic expressions.

Outputs: None

Assumptions: None

Side Effects: None

Availability: On AMS 2.02 and higher.

TI-89 / TI-92 Plus Differences: None

See Also: **push_substitute_simplify, push_subst_no_simp**

Example:

```
void push_subst_or_lim (EStackIndex i, EStackIndex ki, EStackIndex j)
/* Pushes the result of substituting j for ki in i, unless it yields a
   nonunique result and the two-sided limit does not, in which case it
   pushes the latter instead.
*/
{  Access_AMS_Global_Variables;
   EStackIndex k,  old_top = top_estack;
   push_substitute_simplify (i, ki, j);
   k = top_estack;
   if (is_undefined (k))
      {  push_lim (i, ki, j, Integer0Index);
         if (is_undefined (top_estack))
            top_estack = k;
         else delete_between (old_top, k);
      }
} /* end push_subst_or_lim */
```

push_ln

Declaration: void **push_ln** (EStackIndex *k*)

Category(ies): Math

Description: Pushes onto the estack the natural logarithm of the expression indexed by *k*. When *k* indexes a square numeric matrix, pushes the Float matrix ln computed via ln(eigenvalues).

Inputs: *k* — Index of the top tag of an internally-simplified algebraic expression, algebraic comparison, or aggregate thereof.

Outputs: None

Assumptions: None

Side Effects: None

Availability: On AMS 2.02 and higher.

TI-89 / TI-92 Plus Differences: None

See Also: **push_log10, push_exp**

Example:

```
push_quantum (8u);  /* Push variable x */
push_quantum (EXP_TAG);
push_ln (top_estack);  /* Pushes a copy of the variable x. */
```

push_log10

Declaration: void **push_log10** (EStackIndex *k*)

Category(ies): Math

Description: Pushes onto the estack the base-10 logarithm of the expression indexed by *k*. When *k* indexes a square numeric matrix, pushes the Float matrix ln computed via **log10** (eigenvalues).

Inputs: *k* — Index of the top tag of an internally-simplified algebraic expression, algebraic comparison, or aggregate thereof.

Outputs: None

Assumptions: None

Side Effects: None

Availability: On AMS 2.02 and higher.

TI-89 / TI-92 Plus Differences: None

See Also: **push_ln, push_exp**

Example:

```
push_Float (100.0);
push_log10 (top_estack);  /* Pushes a tagged float 2.0 */
```

push_make_proper

Declaration: void **push_make_proper** (EStackIndex *k*)

Category(ies): Math

Description: Pushes onto the estack the equivalent internally-simplified sum of the quotient and proper rational term, with the latter term shallowest.

Inputs: *k* — Indexes an internally-simplified improper rational expression.

Outputs: None

Assumptions: None

Side Effects: None

Availability: On AMS 2.02 and higher.

TI-89 / TI-92 Plus Differences: None

See Also: **is_term_improper**

Example:

```
push_quantum (8u);
numerator = top_estack;  /* Push variable x */
push_sum (numerator, Integer1Index);         /* Push x + 1 */
replace_top2_with_ratio (numerator);         /* top_estack -> x/(x + 1) */
if (is_term_improper (top_estack))
   push_make_proper (top_estack);            /* Pushes 1 – 1/(x + 1) */
```

push_max

Declaration: void **push_max** (EStackIndex *i*, EStackIndex *vi*)

Category(ies): Math

Description: If invoked via **push_internal_simplify**, *vi* and *i* are simplified to deepest variable. Pushes onto the estack a Boolean expression specifying where expression *i* achieves its global maximum with respect to variable *vi*, subject to any current **NG_such_that** constraint.

Inputs:
i — Indexes the top tag of an internally-simplified algebraic expression.
vi — Indexes the top tag of a variable.

Outputs: None

Assumptions: None

Side Effects: None

Availability: On AMS 2.02 and higher.

TI-89 / TI-92 Plus Differences: None

See Also: **push_min, did_push_approx_inflection_point, push_min1, push_min2, push_max1, push_max2**

Example:

```
push_quantum_as_nonnegative_int (2u);
exponent = top_estack;
push_quantum (8u);                /* Push variable x */
replace_top2_with_pow (exponent);
negate_top ();
i = top_estack;   /* i -> -x^2 */
push_quantum (8u);
push_max (i, top_estack);         /* Pushes x=0 */
```

push_max1

Declaration: void **push_max1** (EStackIndex *i*)

Category(ies): Math

Description: If *i* indexes a matrix, returns a one-row matrix of the maximums in each column. Otherwise if *i* indexes a list, returns the maximum of the elements. Otherwise returns expression *i*.

Inputs: *i* — Indexes the top tag of an internally-simplified algebraic expression or an aggregate thereof.

Outputs: None

Assumptions: None

Side Effects: None

Availability: On AMS 2.02 and higher.

TI-89 / TI-92 Plus Differences: None

See Also: **push_min1, push_min2, push_min, push_max2, push_max**

Example:

```
push_quantum (END_TAG);
push_Float (3.2);
push_Float (5.7);
push_quantum (LIST_TAG);
push_max1 (top_estack);  /* Pushes a tagged float 5.7 */
```

push_max2

Declaration: void **push_max2** (EStackIndex *i*, EStackIndex *j*)

Category(ies): Math

Description: Pushes the expression indexed by *i* if *j* is equal to NULL_INDEX. Otherwise pushes onto the estack the internally-simplified maximum of the two expressions indexed by *i* and *j*.

Inputs:

i — Index of the top tag of an internally-simplified algebraic expression or aggregate thereof.

j — NULL_INDEX or an index of the top tag of an internally-simplified algebraic expression or aggregate thereof.

Outputs: None

Assumptions: None

Side Effects: None

Availability: On AMS 2.02 and higher.

TI-89 / TI-92 Plus Differences: None

See Also: **push_min1, push_min2, push_min, push_max1, push_max**

Example:

```
push_Float (3.2);
j = top_estack;
push_Float (5.7);
push_max2 (top_estack, j);  /* Pushes a tagged float 5.7 */
```

push_mid

Declaration: void **push_mid** (EStackIndex *i*, EStackIndex *j*, EStackIndex *k*)

Category(ies): Math

Description: Returns the middle portion of the input argument.

Inputs:

i — EStackIndex of a list or a string.

j — EStackIndex of a non-negative integer or whole number float representing the starting position.

k — EStackIndex of a non-negative integer or whole number float representing the number of elements to push.

Outputs: Pushes the middle *k* elements of *i* beginning with element *j* onto the estack.

Assumptions: If *k* is NULL_INDEX, all elements of *i* beginning with element *j* are pushed.

Side Effects: May cause estack expansion, heap compression, or throw errors if arguments are invalid.

Availability: On AMS 2.00 and higher.

TI-89 / TI-92 Plus Differences: None

See Also: **push_left, push_right, push_rotate, push_shift**

(continued)

push_mid *(continued)*

Example:

If m indexes the bolded tag in the list {a, 0, b, -1} as follows
END_TAG 1 1 NEGATIVE_INTEGER_TAG B_VAR_TAG 0 NONNEGATIVE_INTEGER_TAG A_VAR_TAG **LIST_TAG**

and n indexes the bolded tag in the integer 2 as follows
2 1 **NONNEGATIVE_INTEGER_TAG**

then

```
push_mid (m, n, n);
```

pushes the list {0, b} onto the estack such that **top_estack** points to the bolded tag as follows.
END_TAG B_VAR_TAG 0 NONNEGATIVE_INTEGER_TAG **LIST_TAG**

If m indexes the bolded tag in the string “hello” as follows
0 h e l l o 0 **STR_DATA_TAG**

and n indexes the bolded tag in the tagged floating-point number 2. as follows
0x40 0x00 0x20 0x00 0x00 0x00 0x00 0x00 0x00 **FLOAT_TAG**

and p indexes the bolded tag in the integer 3 as follows
3 1 **NONNEGATIVE_INTEGER_TAG**

then

```
push_mid (m, n, p);
```

pushes the string “ell” onto the estack such that **top_estack** points to the bolded tag as follows.
0 e l l 0 **STR_DATA_TAG**

push_min

Declaration: void **push_min** (EStackIndex *i*, EStackIndex *vi*)

Category(ies): Math

Description: If invoked via **push_internal_simplify**, *vi* and *i* are simplified to deepest variable. Pushes onto the estack a Boolean expression specifying where expression *i* achieves its global minimum with respect to variable *vi*, subject to any current **NG_such_that** constraint.

Inputs:
i — Indexes the top tag of an internally-simplified algebraic expression.
vi — Indexes the top tag of a variable.

Outputs: None

Assumptions: None

Side Effects: None

Availability: On AMS 2.02 and higher.

TI-89 / TI-92 Plus Differences: None

See Also: **push_max, did_push_approx_inflection_point, push_min1, push_min2, push_min, push_max2, push_max1**

Example:

```
push_quantum_as_nonnegative_int (2u);
exponent = top_estack;
push_quantum (8u);                /* Push variable x */
replace_top2_with_pow (exponent);
i = top_estack;  /* top_estack -> x^2 */
push_quantum (8u);
push_min (i, top_estack);         /* Pushes x=0 */
```

push_min1

Declaration: void **push_min1** (EStackIndex *i*)

Category(ies): Math

Description: If *i* indexes a matrix, returns a one-row matrix of the minimums in each column. Otherwise if *i* indexes a list, returns the minimum of the elements. Otherwise returns expression *i*.

Inputs: *i* — Indexes the top tag of an internally-simplified algebraic expression or an aggregate thereof.

Outputs: None

Assumptions: None

Side Effects: None

Availability: On AMS 2.02 and higher.

TI-89 / TI-92 Plus Differences: None

See Also: **push_max1, push_min2, push_min, push_max2, push_max**

Example:

```
push_quantum (END_TAG);
push_Float (3.2);
push_Float (5.7);
push_quantum (LIST_TAG);
push_min1 (top_estack);  /* Pushes a tagged float 3.2 */
```

push_min2

Declaration: void **push_min2** (EStackIndex *i*, EStackIndex *j*)

Category(ies): Math

Description: Pushes the expression indexed by *i* if *j* is equal to NULL_INDEX. Otherwise pushes onto the estack the internally-simplified minimum of the two expressions indexed by *i* and *j*.

Inputs:

i — Index of the top tag of an internally-simplified algebraic expression or aggregate thereof.

j — NULL_INDEX or an index of the top tag of an internally-simplified algebraic expression or aggregate thereof.

Outputs: None

Assumptions: None

Side Effects: None

Availability: On AMS 2.02 and higher.

TI-89 / TI-92 Plus Differences: None

See Also: **push_min1, push_max2, push_min, push_max1, push_max**

Example:

```
push_quantum (END_TAB);
push_Float (3.2);
j = top_estack;
push_Float (5.7);
push_min2 (top_estack, j);  /* Pushes a tagged float 3.2 */
```

push_mod

Declaration: void **push_mod** (EStackIndex *i*, EStackIndex *j*)

Category(ies): Math

Description: Pushes onto the estack the expression indexed by *i* numeric-module the expression indexed by *j*. For positive *j* this is the non-negative rather than centered residue. As suggested by Graham, Knuth & Patashnik, “Concrete Mathematics”, Addison-Wesley, section 3.4: mod(x,0) simplifies to x to preserve the property that mod(x, y) always differs from x by a multiple of y.

Inputs: *i*, *j* — Indices of the top tags of internally-simplified algebraic expressions or aggregates thereof.

Outputs: None

Assumptions: None

Side Effects: None

Availability: On AMS 2.02 and higher.

TI-89 / TI-92 Plus Differences: None

See Also: **push_integer_quotient, push_integer_part, push_floor, push_ceiling, push_integer_remainder**

Example:

```
push_quantum_as_nonnegative_int (3u);
j = top_estack;
push_negate_quantum_as_negint (5u);
push_mod (top_estack, j);  /* Pushes a tagged integer 2 */
```

push_next_arb_int

Declaration: void **push_next_arb_int** (void)

Category(ies): Math

Description: Pushes onto the estack the next arbitrary integer variable. Wraps to 0 suffix when **ARb_int_count** exceeds one quantum.

Inputs: None

Outputs: None

Assumptions: None

Side Effects: None

Availability: On AMS 1.05 and higher.

TI-89 / TI-92 Plus Differences: None

See Also: **push_next_arb_real**

Example:

```
push_next_arb_int();  /* Pushes an ARB_INT_TAG on top of a quantum whose
                         value is history-dependent. */
```

push_next_arb_real

Declaration: void **push_next_arb_real** (void)

Category(ies): Math

Description: Pushes onto the estack the next arbitrary real variable. Wraps to 0 suffix when **ARb_real_count** exceeds one quantum.

Inputs: None

Outputs: None

Assumptions: None

Side Effects: None

Availability: On AMS 1.05 and higher.

TI-89 / TI-92 Plus Differences: None

See Also: **push_next_arb_int**

Example:

```
push_next_arb_real();  /* Pushes an ARB_REAL_TAG on top of a quantum
                          whose value is history-dependent. */
```

push_nint

Declaration: void **push_nint** (EStackIndex *i*, EStackIndex *vi*, EStackIndex *j*, EStackIndex *k*)

Category(ies): Math

Description: Pushes onto the estack the definite integral of the expression indexed by *i* with respect to the variable indexed by *vi* going from the expression indexed by *j* through the expression indexed by *k*, computed via quadrature. If invoked via **push_internal_simplify**, *vi* and *i* are simplified to deepest variable, and the simplification of *i* is done under the temporary influence of SET_PARTIAL_SIMPLIFY to avoid costly polynomial expansions, polynomial GCDs, etc.

Inputs:

i — Indexes the top tag of an internally-simplified algebraic expression or an aggregate thereof.

vi — Indexes the top tag of a variable.

j, k — Indices of the top tags of internally-simplified algebraic expressions.

Outputs: None

Assumptions: None

Side Effects: None

Availability: On AMS 2.02 and higher.

TI-89 / TI-92 Plus Differences: None

See Also: **did_push_anti_deriv, push_def_int, push_arclen**

Example:

```
push1();
k = top_estack;
push0();
j = top_estack;
push_quantum (8u);  /* Push integration variable and integrand x */
push_nint (top_estack, top_estack, j, k);   /* Pushes 0.5 */
```

push_nth_derivative

Declaration: void **push_nth_derivative** (EStackIndex *i*, EStackIndex *j*, EStackIndex *n*)

Category(ies): Math

Description: If *n* is zero, pushes expression *i*. Pushes the (perhaps iterated) antiderivative of *i* with respect to *j* if *n* is negative. Otherwise pushes the *n*th derivative of expression *i* with respect to expression *j*. If invoked via **push_internal_simplify**, *j* and *i* are simplified to deepest variable. However, if the deepest variable value of *j* has a such-that or STO▸ value, that value is substituted for the deepest variable value after computing the symbolic derivative.

For example, *d* (x^2, x, -1) | x = 3 -> 9/2.

Inputs:

i — Indexes the top tag of an internally-simplified algebraic expression, algebraic comparison, or aggregate thereof.

j — Indexes the top tag of a variable.

n — Indexes the top tag of a whole number or an expression that can have only whole number values, such as @n7.

Outputs: None

Assumptions: None

Side Effects: None

Availability: On AMS 2.02 and higher.

TI-89 / TI-92 Plus Differences: None

See Also: **push_1st_derivative**

Example:

```
push_quantum_as_nonnegative_int (3u);
exponent = top_estack;
push_quantum (8u);   /* Push variable x */
replace_top2_with_pow (exponent);
i = top_estack;   /* i -> x^3 */
push_quantum_as_nonnegative_int (2u);
n = top_estack;
push_quantum (8u);   /* Push variable x */
push_nth_derivative (i, top_estack, n);   /* Pushes 3 * x^2 */
```

push_perm

Declaration: void **push_perm** (EStackIndex *i*, EStackIndex *j*)

Category(ies): Math

Description: Let *i* index expression z, and let *j* index expression k. Pushes z!/k! onto the estack. For integer k >= 0, this is the Pochhammer symbol (z)[k], or the "falling factorial power" z ∗ (z - 1) ∗ . . . ∗ (z - k). If z is also an integer and z >= k, this is the number of permutations of z items taken k at a time.

Inputs: *i*, *j* — Indices of the top tags of internally-simplified algebraic expressions or aggregates thereof.

Outputs: None

Assumptions: None

Side Effects: None

Availability: On AMS 2.02 and higher.

TI-89 / TI-92 Plus Differences: None

See Also: None

Example:

```
push_quantum_as_nonnegative_int (4u);
i = top_estack;
push_quantum_as_nonnegative_int (2u);
push_perm (i, top_estack);   /* Pushes a tagged integer 12 */
```

push_phase

Declaration: void **push_phase** (EStackIndex *i*)

Category(ies): Math

Description: Pushes onto the estack the internally-simplified phase angle of the expression indexed by *i*, measured counter-clockwise in radians from the x axis in the range (-π, π].

Inputs: *i* — Index of the top tag of an internally-simplified algebraic expression, algebraic comparison, or aggregate thereof.

Outputs: None

Assumptions: None

Side Effects: None

Availability: On AMS 2.02 and higher.

TI-89 / TI-92 Plus Differences: None

See Also: None

Example:

```
push0 ();
real_part = top_estack;
push1 ();
replace_top2_with_imre (real_part);
push_phase (top_estack);   /* Pushes pi/2 */
```

push_poly_qr

Declaration: EStackIndex **push_poly_qr** (EStackIndex *i*, EStackIndex *j*, EStackIndex *gv*, Int *choice*)

Category(ies): Math

Description: According to *choice*, pushes onto the estack the remainder, and/or then the quotient of polynomial *i* divided by polynomial *j*, with respect to *gv*. Returns the index of the first expression pushed.

Top-level calls should be made via the macros PUSH_POLY_QUOTIENT, PUSH_POLY_REMAINDER, or PUSH_POLY_REMAINDER_THEN_QUOTNT.

Inputs:

i, *j* — Indices of the top tags of internally-simplified polynomials, generalized to allow non-negative fractional exponents.

gv — Indexes the top tag of an internally-simplified variable or kernel.

choice — One of PUSH_QUOTIENT, PUSH_REMAINDER, or PUSH_BOTH.

Outputs: The index of the first expression pushed.

Assumptions: None

Side Effects: None

Availability: On AMS 2.02 and higher.

TI‑89 / TI‑92 Plus Differences: None

See Also: **push_gcd_then_cofactors**

Example:

```
/* Pushes a tagged integer 2, then the polynomial x + 1, then returns
   the index of the pushed tagged integer 2.
*/
push_quantum (2u);
exponent = top_estack;
push_quantum (8u);   /* Push variable x */
replace_top2_with_pow (exponent);   /* top_estack -> x^2 */
add1_to_top ();
i = top_estack;   /* i -> x^2 + 1 */
push_quantum (8u);  /* Push variable x */
subtract1_from_top ();
j = top_estack;   /* j -> x + 1 */
push_quantum (8u);
gv = top_estack;   /* gv -> variable x */
push_poly_qr (i, j, gv, PUSH_BOTH);
```

push_r_cis

Declaration: void **push_r_cis** (EStackIndex *r*, EStackIndex *t*)

Category(ies): Math

Description: Pushes internally-simplified $r\cos(t) + i\, r\sin(t)$ onto the estack, with *t* measured in radians.

Inputs: *r*, *t* — Indices of the top tags of internally-simplified real algebraic expressions.

Outputs: None

Assumptions: None

Side Effects: None

Availability: On AMS 2.02 and higher.

TI-89 / TI-92 Plus Differences: None

See Also: **push_phase, push_abs**

Example:

```
/* Pushes the internal representation of sqrt(-1) onto the estack */
push_pi_on_quantum (2u);
t = top_estack;  /* t -> pi/2 */
push1 ();
push_r_cis (top_estack, t);
```

push_rand

Declaration: void **push_rand** (EStackIndex *num_idx*)

Category(ies): Math

Description: Pushes a random number onto the estack. If *num_idx* indexes the END_TAG then a floating-point number between zero and one is generated. If the number indexed by *num_idx* is a positive integer then a number between one and that number is generated. If the number indexed by *num_idx* is a negative integer then a number between that integer and minus one is generated.

Inputs: *num_idx* — Indexes the input number (may index an END_TAG).

Outputs: None

Assumptions: None

Side Effects: May expand expression stack, cause heap compression, or throw an error.

Availability: On AMS 1.05 and higher.

TI-89 / TI-92 Plus Differences: None

See Also: **push_randmat, push_randpoly, push_randnorm**

Example: This example pushes a list of ten random numbers in the range [0.0, 1.0] on the estack.

```
EStackIndex esi;
short i;

push_quantum( END_TAG );
esi = top_estack;
for (i = 1; i <= 10; i++)
   push_rand( esi );  /* index END_TAG (so get numbers in range 0 . . . 1) */
push_quantum( LIST_TAG );
```

push_radians

Declaration: void **push_radians** (EStackIndex *i*)

Category(ies): Math

Description: Converts the input value, interpreted as radians, to the currently selected angle measure.

Inputs: *i* — EStackIndex of internal tokenized radians value.

Outputs: Pushes the internal tokenized form of the result of converting the specified radians to the currently selected angle measure — radians or decimal degrees.

Assumptions: None

Side Effects: May cause estack expansion, heap compression, or throw errors.

Availability: On AMS 2.00 and higher.

TI-89 / TI-92 Plus Differences: None

See Also: **push_degrees**

Example:

If the current angle setting is RADIANS and i indexes π/2, then

```
push_radians (i);
```

pushes the value π/2 onto the estack.

If the current angle setting is DEGREES and i indexes π/2, then

```
push_radians (i);
```

pushes the value 90 which is the equivalent number of degrees onto the estack.

push_randpoly

Declaration: void **push_randpoly** (EStackIndex *var_idx*, EStackIndex *order_idx*)

Category(ies): Math

Description: Pushes onto the estack a polynomial of the specified order with random integer coefficients between -9 and 9 and with the leading coefficient nonzero. The caller must specify an independent variable.

Inputs:

var_idx — Indexes the name of the independent variable to use in the output polynomial.

order_idx — Indexes the order of the output polynomial.

Outputs: None

Assumptions: None

Side Effects: May expand expression stack, cause heap compression, or throw an error.

Availability: On AMS 1.05 and higher.

TI-89 / TI-92 Plus Differences: None

See Also: **push_rand, push_randmat, push_randnorm**

Example: This example pushes a fourth order polynomial in “z” on the estack.

```
BYTE zName = ENCODE_LETTER('z');  /* single letter variables are stored specially */

push_ushort_to_integer( 4 );  /* 4th order polynomial */
push_randpoly( zName, top_estack );
```

push_re

Declaration: void **push_re** (EStackIndex *i*)

Category(ies): Math

Description: Pushes onto the estack the internally-simplified real part of the expression indexed by *i*.

Inputs: *i* — Index of the top tag of an internally-simplified algebraic expression, algebraic comparison, or aggregate thereof.

Outputs: None

Assumptions: None

Side Effects: None

Availability: On AMS 2.02 and higher.

TI‑89 / TI‑92 Plus Differences: None

See Also: **push_im, push_phase, push_abs, push_conj**

Example:

```
push_Float (2.3);
real_part = top_estack;
push_Float (3.5);
replace_top2_with_imre (real_part);
push_re (top_estack);   /* Pushes tagged float 2.3 */
```

push_rec_to_angle

Declaration: void **push_rec_to_angle** (EStackIndex *i*, EStackIndex *j*)

Category(ies): Math

Description: Expression *i* indexes an x coordinate and expression *j* indexes a y coordinate. Pushes onto the estack the internally-simplified radian angle in the range ($^{-}\pi$, π], measured counter-clockwise from the positive x axis. If *i* and *j* both index zeros, **push_rec_to_angle**(*i, j*) pushes REC_TO_ANGLE_TAG on top of the two zeros representing the open-closed interval ($^{-}\pi$, π].

Inputs: *i*, *j* — Indices of the top tags of internally-simplified algebraic expressions, algebraic comparisons, or aggregates thereof.

Outputs: None

Assumptions: None

Side Effects: None

Availability: On AMS 2.02 and higher.

TI-89 / TI-92 Plus Differences: None

See Also: **push_atan, push_phase**

Example:

```
push_quantum_as_nonnegative_int (11u);
push_rec_to_angle (top_estack, top_estack);  /* Pushes pi/4 */
```

push_right

Declaration: void **push_right** (EStackIndex *i*, EStackIndex *j*)

Category(ies): Math

Description: Returns the rightmost portion of the input argument.

Inputs: *i* — EStackIndex of a list, a string, or a relational expression.

j — EStackIndex of a non-negative integer or whole number float.

Outputs: Pushes the rightmost *j* elements of *i* onto the estack.

Assumptions: If *i* is a list or string, and *j* is NULL_INDEX, all of *i* is pushed onto the estack.

Side Effects: May cause estack expansion, heap compression, or throw errors if arguments are invalid.

Availability: On AMS 2.00 and higher.

TI-89 / TI-92 Plus Differences: None

See Also: **push_left, push_mid, push_rotate, push_shift**

(continued)

push_right *(continued)*

Example:

If m indexes the bolded tag in the list {a, 0, b, -1} as follows
END_TAG 1 1 NEGATIVE_INTEGER_TAG B_VAR_TAG 0 NONNEGATIVE_INTEGER_TAG A_VAR_TAG **LIST_TAG**

and n indexes the bolded tag in the integer 3 as follows
3 1 **NONNEGATIVE_INTEGER_TAG**

then
```
push_right (m, n);
```

pushes the list {0, b, -1} onto the estack such that **top_estack** points to the bolded tag as follows.
END_TAG 1 1 NEGATIVE_INTEGER_TAG B_VAR_TAG 0 NONNEGATIVE_INTEGER_TAG **LIST_TAG**

If m indexes the bolded tag in the string “hello” as follows
0 h e l l o 0 **STR_DATA_TAG**

and n indexes the bolded tag in the floating-point number 2. as follows
0x40 0x00 0x20 0x00 0x00 0x00 0x00 0x00 0x00 **FLOAT_TAG**

then
```
push_right (m, n);
```

pushes the string “lo” onto the estack such that **top_estack** points to the bolded tag as follows.
0 l o 0 **STR_DATA_TAG**

If m indexes the bolded tag in the relational expression x < 3 as follows
3 1 NONNEGATIVE_INTEGER_TAG X_VAR_TAG **LT_TAG**

and n is NULL_INDEX

then
```
push_right (m, n);
```

pushes the right side of the expression which is 3 onto the estack such that **top_estack** points to the bolded tag as follows.
3 1 **NONNEGATIVE_INTEGER_TAG**

push_rotate

Declaration: void **push_rotate** (EStackIndex *i*, EStackIndex *j*)

Category(ies): Math

Description: Rotates the elements of an integer, a list, or a string to the left or right.

Inputs:
i — EStackIndex of an integer, a list, or a string.

j — EStackIndex of an integer or whole number float.

Outputs: Pushes a rotated copy of *i* onto the estack.

If *i* is an integer, then the bits are rotated.

If *i* is a list, the elements of the list are rotated.

If *i* is a string, the characters of the string are rotated.

If *j* is positive, the rotation is that number of places to the left.

If *j* is negative, the rotation is that number of places to the right.

Each time the input is rotated left, the leftmost element (integer bit, list element, string character) that is pushed out is moved to the rightmost position.

Each time the input is rotated right, the rightmost element that is pushed out is moved to the leftmost position.

Assumptions: None

Side Effects: May expand the estack, cause heap compression, or throw errors if arguments are invalid

Availability: On AMS 2.00 and higher.

TI-89 / TI-92 Plus Differences: None

See Also: **push_left, push_mid, push_right, push_shift**

(continued)

push_rotate *(continued)*

Example:

If m indexes the bolded tag in the integer 256 (0b100000000) as follows
0 1 2 **NONNEGATIVE_INTEGER_TAG**

and n indexes the bolded tag in the integer -1 as follows
1 1 **NEGATIVE_INTEGER_TAG**

then
```
push_rotate (m, n);
```

pushes 128 (0b10000000) onto the estack such that **top_estack** points to the bolded tag as follows.
128 1 **NONNEGATIVE_INTEGER_TAG**

If m indexes the bolded tag in the list {a, b, c} as follows
END_TAG C_VAR_TAG B_VAR_TAG A_VAR_TAG **LIST_TAG**

and n indexes the bolded tag in the integer one as follows
1 1 **NONNEGATIVE_INTEGER_TAG**

then
```
push_rotate (m, n);
```

pushes the left rotated list {b, c, a} onto the estack such that **top_estack** points to the bolded tag as follows.
END_TAG A_VAR_TAG C_VAR_TAG B_VAR_TAG **LIST_TAG**

If m indexes the bolded tag in the string “hello” as follows
0 h e l l o 0 **STR_DATA_TAG**

and n indexes the bolded tag in the floating-point number -3. as follows
0xC0 0x00 0x30 0x00 0x00 0x00 0x00 0x00 0x00 **FLOAT_TAG**

then
```
push_rotate (m, n);
```

pushes the right rotated string “llohe“ onto the estack such that **top_estack** points to the bolded tag as follows.
0 l l o h e 0 **STR_DATA_TAG**

push_round

Declaration: void **push_round** (EStackIndex *val_idx*, EStackIndex *num_dig_idx*)

Category(ies): Math

Description: Pushes onto the estack the argument rounded to the specified number of digits.

Inputs:

val_idx — Indexes the input number, list, or matrix.

num_dig_idx — Indexes the number of digits after the decimal place to round to or if NULL then uses the default value from the MODE settings.

Outputs: None

Assumptions: None

Side Effects: May expand expression stack, cause heap compression, or throw an error.

Availability: On AMS 2.00 and higher.

TI-89 / TI-92 Plus Differences: None

See Also: None

Example: This example creates a static column vector and a static integer then creates a unit vector from the column vector and finally rounds that result to four decimal places.

```
/* static column vector */
static const Quantum colVec[] =    {END_TAG,
   END_TAG,3,1,NONNEGATIVE_INTEGER_TAG,LIST_TAG,
   END_TAG,2,1,NONNEGATIVE_INTEGER_TAG,LIST_TAG,
   END_TAG,1,1,NONNEGATIVE_INTEGER_TAG,LIST_TAG,
   LIST_TAG
};
#define colVecIndex ((EStackIndex) (colVec+sizeof(colVec)-1))
static const Quantum Integer4 [] = {4u, 1u, NONNEGATIVE_INTEGER_TAG};
#define Integer4Index ((EStackIndex) (Integer4+2))

push_unitv( colVecIndex );

push_round( top_estack, Integer4Index );
```

push_sequence

Declaration: void **push_sequence** (EStackIndex *i*, EStackIndex *j*, EStackIndex *k*, EStackIndex *m*, EStackIndex *n*)

Category(ies): Math, Lists and Matrices

Description: Pushes a sequence of internally-simplified values into a list on the estack.

Inputs:

i — EStackIndex of the sequence generator expression in either internal or external tokenized form.

j — EStackIndex of the independent variable.

k — EStackIndex of the starting value of the independent variable in internal tokenized form.

m — EStackIndex of the ending value of the independent variable in internal tokenized form.

n — EStackIndex of the step value of the independent variable in internal tokenized form.

Outputs: Pushes a sequence as a list on the estack. The sequence is generated by incrementing variable *j* from start value *k* through end value *m* by an increment of step *n*, and evaluating the sequence generator expression *i* for each step. The resulting sequence is in internal tokenized form.

Assumptions: None

Side Effects: May expand the estack, cause heap compression, or throw a variety of evaluation errors depending upon the generator expression

Availability: On AMS 2.00 and higher.

TI-89 / TI-92 Plus Differences: None

See Also: None

(continued)

push_sequence *(continued)*

Example:

If i indexes the bolded tag in the expression n^2 as follows
2 1 NONNEGATIVE_INTEGER_TAG N_VAR_TAG **EXPONENTIATION_TAG**

and j indexes the bolded tag in the variable n as follows
N_VAR_TAG

and k indexes the bolded tag in the integer 1 as follows
1 1 **NONNEGATIVE_INTEGER_TAG**

and m indexes the bolded tag in the integer 4 as follows
4 1 **NONNEGATIVE_INTEGER_TAG**

and n indexes the bolded tag in the integer 1 as follows
1 1 **NONNEGATIVE_INTEGER_TAG**

then

```
push_sequence (i, j, k, m, n);
```

pushes the sequence {1, 4, 9, 16} onto the estack such that **top_estack** points to the bolded tag as follows.
END_TAG 16 1 NONNEGATIVE_INTEGER_TAG 9 1 NONNEGATIVE_INTEGER_TAG 4 1 NONNEGATIVE_INTEGER_TAG 1 1 NONNEGATIVE_INTEGER_TAG **LIST_TAG**

push_shift

Declaration: void **push_shift** (EStackIndex *i*, EStackIndex *j*)

Category(ies): Math

Description: Shifts the elements of an integer, a list, or a string to the left or right.

Inputs:
i — EStackIndex of an integer, a list, or a string.
j — EStackIndex of an integer or whole number float.

Outputs: Pushes a shifted copy of *i* onto the estack.

If *i* is an integer, then the bits are shifted.

If *i* is a list, the elements of the list are shifted.

If *i* is a string, the characters of the string are shifted.

If *j* is positive, the shift is that number of places to the left.

If *j* is negative, the shift is that number of places to the right.

Each time an integer is shifted left, the leftmost bit is dropped, and a zero bit is introduced on the right. Each time an integer is shifted right, the rightmost bit is dropped, and the original leftmost bit (0 or 1) is copied on the left.

Each time a list is shifted left or right, the appropriate element is dropped, and the symbol **undef** is introduced at the opposite end of the list.

Each time a string is shifted left or right, the appropriate character is dropped, and a space is introduced at the opposite end of the list.

Assumptions: None

Side Effects: May expand the estack, cause heap compression, or throw errors if arguments are invalid.

Availability: On AMS 2.00 and higher.

TI-89 / TI-92 Plus Differences: None

See Also: **push_left, push_mid, push_right, push_rotate**

(continued)

push_shift *(continued)*

Example:

If m indexes the bolded tag in the integer 256 (0b100000000) as follows
0 1 2 **NONNEGATIVE_INTEGER_TAG**

and n indexes the bolded tag in the integer one as follows
1 1 **NONNEGATIVE_INTEGER_TAG**

then

```
push_shift (m, n);
```

pushes 512 (0b1000000000) onto the estack such that **top_estack** points to the bolded tag as follows.
0 2 2 **NONNEGATIVE_INTEGER_TAG**

If m indexes the bolded tag in the list {a, b, c} as follows
END_TAG C_VAR_TAG B_VAR_TAG A_VAR_TAG **LIST_TAG**

and n indexes the bolded tag in the integer –1 as follows
1 1 **NEGATIVE_INTEGER_TAG**

then

```
push_shift (m, n);
```

pushes the right shifted list {undef, a, b} onto the estack such that **top_estack** points to the bolded tag as follows.
END_TAG B_VAR_TAG A_VAR_TAG UNDEFINED_TAG **LIST_TAG**

If m indexes the bolded tag in the string “hello” as follows
0 h e l l o 0 **STR_DATA_TAG**

and n indexes the bolded tag in the floating-point number 3. as follows
0x40 0x00 0x30 0x00 0x00 0x00 0x00 0x00 0x00 **FLOAT_TAG**

then

```
push_shift (m, n);
```

pushes the left shifted string “lo ” onto the estack such that **top_estack** points to the bolded tag as follows.
0 l o ␣ ␣ ␣ 0 **STR_DATA_TAG**

push_simult

Declaration: void **push_simult** (EStackIndex *mat_idx*, EStackIndex *vec_idx*, EStackIndex *tol_idx*)

Category(ies): Math, Lists and Matrices

Description: Pushes onto the estack a column vector containing the solution of a system of equations. Throws ER_SINGULARMAT if no solution can be found.

Inputs:

mat_idx — Indexes the input matrix which must be square.

vec_idx — Indexes the input column vector.

tol_idx — If not NULL then indexes a tolerance factor. Any matrix element is treated as zero if its absolute value is less than the tolerance.

Outputs: None

Assumptions: None

Side Effects: May expand expression stack, cause heap compression, or throw an error.

Availability: On AMS 2.00 and higher.

TI-89 / TI-92 Plus Differences: None

See Also: None

(continued)

push_simult *(continued)*

Example: This example solves a system of equations of order two (although it could solve any order). The input matrix is stored in ‘A’ and the column vector is stored in ‘B’. The result is left on the estack.

```
short i, j, order = 2;
EStackIndex mat_idx;
float A[] =  {1.0,2.0,3.0,4.0};
float B[] =  {5.0,6.0};

/* push A onto estack */
push_quantum (END_TAG);
for (i = order-1; i >= 0; i--) {
   push_quantum (END_TAG);
   for (j = order-1; j >= 0; j--)
      push_Float (A[i*order + j]);
   push_quantum (LIST_TAG);
}
push_quantum (LIST_TAG);
mat_idx = top_estack;

/* push B onto estack */
push_quantum (END_TAG);
for (i = order-1; i >= 0; i--) {
   push_quantum (END_TAG);
   push_Float (B[i]);
   push_quantum (LIST_TAG);
}
push_quantum (LIST_TAG);

/* Solve */
push_simult (mat_idx, top_estack, NULL_INDEX);
```

push_sin

Declaration: void **push_sin** (EStackIndex *k*)

Category(ies): Math

Description: Pushes onto the estack the internally-simplified sine of the expression indexed by *k*. The only trigonometric tag that internally-simplified results can contain is SIN2_TAG or (if IS_COLLECT_KERNELS) TAN_RAD_TAG, whose arguments are always in radians. When *k* indexes a square numeric matrix, pushes the Float matrix sine computed via sin(eigenvalues).

Inputs: *k* — Indexes the top tag of an internally-simplified algebraic expression, algebraic comparison, or aggregate thereof, with angles measured in radians.

Outputs: None

Assumptions: None

Side Effects: None

Availability: On AMS 2.02 and higher.

TI-89 / TI-92 Plus Differences: None

See Also: **push_sin2, push_cos, push_tan, push_trig**

Example:

```
push_pi_on_quantum (6u);  /* Push pi/6 */
push_sin (top_estack);  /* Pushes fraction 1/2 */
```

push_sin2

Declaration: void **push_sin2** (EStackIndex *i*, EStackIndex *j*)

Category(ies): Math

Description: Pushes onto the estack sine (expression *i* + (expression *j*) ∗ π/2).
sin (z) == sin2 (z, 0)
cos (z) == sin2 (z, 1)

The second argument saves code space by unifying rules for sine and cosine, while making the "constant multiple of π" term efficiently accessible.

When *i* indexes a square numeric matrix, pushes the Float matrix sin2 computed via sin2(eigenvalues(*i*), *j*).

Example: If *i* indexes 0 and *j* indexes 1/3, pushes 1/2.

Inputs:

i — Indexes the top tag of an internally-simplified algebraic expression, algebraic comparison, or aggregate thereof, with angles measured in radians.

j — Indexes a number denoting a multiple of π/2.

Outputs: None

Assumptions: None

Side Effects: None

Availability: On AMS 2.02 and higher.

TI-89 / TI-92 Plus Differences: None

See Also: **push_sin2, push_cos, push_tan, push_trig**

Example:

```
push_quantum_as_nonnegative_int (0u);
i = top_estack;
push_reciprocal_of_quantum (3u);
push_sin2 (i, top_estack);  /* Pushes fraction 1/2 */
```

push_sinh

Declaration: void **push_sinh** (EStackIndex *k*)

Category(ies): Math

Description: Pushes onto the estack the internally-simplified hyperbolic sine of the expression indexed by *k*. When *k* indexes a square numeric matrix, pushes the Float matrix sinh computed via sinh(eigenvalues).

Inputs: *k* — Indexes the top tag of an internally-simplified algebraic expression, algebraic comparison, or aggregate thereof, with angles measured in radians.

Outputs: None

Assumptions: None

Side Effects: None

Availability: On AMS 2.02 and higher.

TI-89 / TI-92 Plus Differences: None

See Also: **push_cosh, push_tanh, push_acosh, push_asinh, push_atanh, push_exp**

Example:

```
push_quantum_as_nonnegative_int(0u);   /* Push tagged integer 0 */
push_sinh (top_estack);   /* Pushes a tagged integer 0 */
```

push_sqrt

Declaration: void **push_sqrt** (EStackIndex *i*)

Category(ies): Math

Description: Pushes onto the estack the internally-simplified square root of the expression indexed by *i*.

Inputs: *i* — Index to the top tag of an internally-simplified algebraic expression, algebraic comparison, or aggregate thereof.

Outputs: None

Assumptions: None

Side Effects: None

Availability: On AMS 2.02 and higher.

TI-89 / TI-92 Plus Differences: None

See Also: **push_exponentiate, raise_to_top, push_square, push_reciprocal, replace_top2_with_pow, replace_top_with_reciprocal, push_dot_exponentiate**

Example:

```
void push_arclen (EStackIndex i, EStackIndex vi, EStackIndex j, EStackIndex k)
/* j and k index expressions, vi indexes a variable, and i indexes an
   expression simplified through variable vi.
   Pushes onto the estack the arc displacement of expression i with respect
   to vi going from j through k.
*/
{  Access_AMS_Global_Variables;
   EStackIndex m, old_top = top_estack;
   push_quantum_as_nonnegative_int (2u);
   m = top_estack;
   push_1st_derivative (i, vi);
   replace_top2_with_pow (m);
   add1_to_top ();
   i = top_estack;
   push_sqrt (i);
   delete_between (old_top, i);
   i = top_estack;
   push_def_int (i, vi, j, k);
   delete_between (old_top, i);
}
```

push_square

Declaration: void **push_square** (EStackIndex *i*)

Category(ies): Math

Description: Pushes onto the estack the internally-simplified value of the square of the expression indexed by *i*.

Inputs: *i* — Index to the top tag of an internally-simplified algebraic expression, algebraic comparison, or aggregate thereof.

Outputs: None

Assumptions: None

Side Effects: None

Availability: On AMS 2.02 and higher.

TI-89 / TI-92 Plus Differences: None

See Also: **push_exponentiate, raise_to_top, push_sqrt, push_reciprocal, replace_top2_with_pow, replace_top_with_reciprocal, push_dot_exponentiate**

Example:

```
void push_quadratic_discriminant (EStackIndex a, EStackIndex b, EStackIndex c)
/* Pushes onto the estack  b^2 - 4 a c. */
{  Access_AMS_Global_Variables;
   EStackIndex old_top = top_estack;
   push_negate_quantum_as_negint (4u);
   times_top (a);
   times_top (c);
   a = top_estack;

   push_square (b);
   replace_top2_with_sum (a);
}
```

push_standardize

Declaration: void **push_standardize** (EStackIndex *i*)

Category(ies): Math

Description: Pushes onto the estack an equivalent expression in which for all sums at all recursive levels:

- The sums contain no negative exponents or monomial factors.
- The leading coefficient is positive.
- If the numeric factors of the terms are all exactly-representable whole numbers, then the gcd these numeric factors is 1. Otherwise the leading coefficient is 1.0. This is the minimum level of factorization. It includes forming common denominators. To within further factorization of sums, it standardizes expressions so that similar factors are more likely to be recognized.

Examples:
x^-1 ∗ (y^2 ∗ -6 + y ∗ 4) + 5 -> (x ∗ 5 + (y ∗ 3 + -2) ∗ y ∗ -2) ∗ x^-1
x^-1 ∗ (y^2 ∗ -6.0 + y ∗ 4) + 5 -> (x ∗ 5 + (y ∗ 3.0 + -2) ∗ y ∗ -2) ∗ x^-1
x^-1 ∗ (y^2 ∗ -1.2 + y ∗ 4) + 5 -> (x + (y + -3.333 . . .) ∗ y ∗ -.24) ∗ x^-1 ∗ 5.0

Inputs: *i* — Indexes the top tag of an internally-simplified algebraic expression.

Outputs: None

Assumptions: None

Side Effects: None

Availability: On AMS 2.02 and higher.

TI-89 / TI-92 Plus Differences: None

See Also: **push_factor, push_comdenom**

```
push_quantum_as_nonnegative_int(2u);
exponent = top_estack;
push_quantum (8u);     /* Push variable x */
replace_top2_with_pow (exponent);
power = top_estack;    /* x^2 */
push_quantum (8u);     /* Push variable x */
replace_top2_with_sum (power);   /* top_estack -> x^2 + x */
push_standardize (top_estack);   /* Pushes (x + 1)*x */
```

push_summation

Declaration: void **push_summation** (EStackIndex *i*, EStackIndex *ki*, EStackIndex *j*, EStackIndex *k*)

Category(ies): Math

Description: Expression *i* is simplified under the influence of a temporary "| $ki \geq j$ and $ki \leq k$." Pushes $\Sigma(i, ki, j, k)$ onto the estack. The summand *i* must be unsimplified for examples such as Σ(rand(9), . . .) to have the intended effect.

Inputs:

i — Index of the top tag of an algebraic expression, algebraic comparison, or aggregate thereof.

ki — Index of the top tag of a variable.

j, k — Indices of the top tags of internally-simplified expressions.

Outputs: None

Assumptions: None

Side Effects: None

Availability: On AMS 2.02 and higher.

TI-89 / TI-92 Plus Differences: None

See Also: **push_extended_product**

Example:

```
/* Pushes Σ(x, x, 1, y) -> y * (y + 1)/2 */
push_quantum (9u);
k = top_estack;   /* k -> variable y */
push_quantum_as_nonnegative_int (1u);
j = top_estack;   /* j -> 1 */
push_quantum (8u);   /* top_estack -> variable x */
push_summation (top_estack, top_estack, j, k);
```

push_tan

Declaration: void **push_tan** (EStackIndex *k*)

Category(ies): Math

Description: Pushes onto the estack the internally-simplified tangent of the expression indexed by *k*. The only trigonometric tag that internally-simplified results can contain is SIN2_TAG or (if IS_COLLECT_KERNELS) TAN_RAD_TAG, both of whose arguments are always in radians. When *k* indexes a square numeric matrix, pushes the Float matrix tan computed via tan(eigenvalues).

Inputs: *k* — Indexes the top tag of an internally-simplified algebraic expression, algebraic comparison, or aggregate thereof, with angles measured in radians.

Outputs: None

Assumptions: None

Side Effects: None

Availability: On AMS 2.02 and higher.

TI-89 / TI-92 Plus Differences: None

See Also: **push_sin2, push_cos, push_sin, push_trig**

Example:

```
push_pi_on_quantum (4u);   /* Push pi/4 */
push_tan (top_estack);   /* Pushes tagged integer 1 */
```

push_tanh

Declaration: void **push_tanh** (EStackIndex *k*)

Category(ies): Math

Description: Pushes onto the estack the internally-simplified hyperbolic tangent of the expression indexed by *k*. When *k* indexes a square numeric matrix, pushes the Float matrix tanh computed via tanh(eigenvalues).

Inputs: *k* — Indexes the top tag of an internally-simplified algebraic expression, algebraic comparison, or aggregate thereof.

Outputs: None

Assumptions: None

Side Effects: None

Availability: On AMS 2.02 and higher.

TI-89 / TI-92 Plus Differences: None

See Also: **push_sinh, push_cosh**

Example:

```
push_quantum (PLUS_INFINITY_TAG);   /* Push +∞ */
push_tanh (top_estack);   /* Pushes tagged integer 1 */
```

push_trig

Declaration: void **push_trig** (void (* *proc*) (EStackIndex), EStackIndex *i*)

Category(ies): Math

Description: Pushes the corresponding internally-simplified result onto the estack. The only trigonometric tag that internally-simplified results can contain is SIN2_TAG or (if IS_COLLECT_KERNELS) TAN_RAD_TAG, both of whose first arguments are always in radians.

Inputs:

proc — The address of **push_sin**, **push_cos**, or some other such radian mode trig pusher subroutine of one EStackIndex argument.

i — Indexes the top tag of an internally-simplified algebraic expression, algebraic comparison or aggregate thereof, with angles measured according to IS_DEGREES versus IS_RADIANS.

Outputs: None

Assumptions: None

Side Effects: None

Availability: On AMS 2.02 and higher.

TI-89 / TI-92 Plus Differences: None

See Also: **push_sin, push_cos, push_tan, push_sin2**

Example: If IS_DEGREES and not IS_ARITH_APPROX, and *i* indexes integer 60, then **push_trig** (**push_sin**, *i*) pushes 1/2.

```
/* Depending on the arithmetic mode: Pushes tagged 0.5 if IS_DEGREES;
   Otherwise pushes –3.0481062110209 or symbolic sin(60).
*/
push_Float (60.0);
push_trig (push_sin, top_estack);
```

raise_to_top

Declaration: void **raise_to_top** (EStackIndex *i*)

Category(ies): Math

Description: Replaces the top expression on the estack with the simplified result of (expression *i*)^(top expression). If expression *i* is a square matrix: pushes the same-size identity matrix if **top_estack** indexes a zero; pushes the iterated matrix product if **top_estack** indexes a positive whole number; pushes the inverse matrix or its iterated matrix product if **top_estack** indexes a negative whole number.

Inputs: *i, j* — Indexes to the top tags of internally-simplified algebraic expressions, algebraic comparisons, or aggregates thereof.

Outputs: None

Assumptions: None

Side Effects: None

Availability: On AMS 2.02 and higher.

TI-89 / TI-92 Plus Differences: None

See Also: **push_dot_exponentiate, push_exponentiate, replace_top2_with_pow, push_sqrt, push_square, push_reciprocal, replace_top_with_reciprocal**

Example:

```
void push_increment_degree (EStackIndex power, EStackIndex inc)
/* Pushes onto the estack factor_base (power) ^ (inc + factor_deg(power)) */
{  push_sum (inc, factor_exponent_index (power));
   raise_to_top (factor_base_index (power));
}
```

replace_top2_with_pow

Declaration: void **replace_top2_with_pow** (EStackIndex *i*)

Category(ies): Math

Description: Replaces the top two expressions of the estack with (top expression)^(expression *i*). If expression *i* is a square matrix, pushes the same-size identity matrix. If **top_estack** indexes a zero, pushes the iterated matrix product. If **top_estack** indexes a positive whole number, pushes the inverse matrix or else its iterated matrix product if **top_estack** indexes a negative whole number.

Inputs: *i* — Index to the top tag of the deeper of the top two expressions of the estack. These top two expressions are internally-simplified algebraic expressions, algebraic comparisons, or aggregates thereof.

Outputs: None

Assumptions: None

Side Effects: None

Availability: On AMS 2.02 and higher.

TI-89 / TI-92 Plus Differences: None

See Also: **push_dot_exponentiate, raise_to_top, push_exponentiate, push_sqrt, push_square, push_reciprocal, replace_top_with_reciprocal**

Example:

```
void push_arclen(EStackIndex i,EStackIndex vi,EStackIndex j,EStackIndex k)
/* j and k index expressions, vi indexes a variable, and i indexes an
   expression simplified through variable vi.
   Pushes onto the estack the arc displacement of expression i with respect
   to vi going from j through k.
*/
{  Access_AMS_Global_Variables;
   EStackIndex m,  old_top = top_estack;
   push_quantum_as_nonnegative_int (2u);
   m = top_estack;
   push_1st_derivative (i, vi);
   replace_top2_with_pow (m);
   add1_to_top ();
   i = top_estack;
   push_sqrt (i);
   delete_between (old_top, i);
   i = top_estack;
   push_def_int (i, vi, j, k);
   delete_between (old_top, i);
}
```

Appendix A: System Routines — Memory Management

See Also:

HeapAlloc

Declaration: HANDLE **HeapAlloc** (DWORD *Hlen*)

Category(ies): Memory Management

Description: Allocate a block of heap memory of the given size and return its handle. Use **HeapDeref** to dereference the handle and get a pointer to the actual memory. Note that a pointer to the heap is valid only as long as heap compression is not done.

Inputs: *Hlen* — Length of block of memory to allocate (all odd sizes are rounded up to be even).

Outputs: HANDLE of memory block allocated, H_NULL if not enough memory.

Assumptions: *Hlen* may not exceed 65520 bytes and the minimum block size is eight bytes.

Side Effects: May cause heap compression.

Availability: All versions of the TI‑89 / TI‑92 Plus.

TI‑89 / TI‑92 Plus Differences: None

See Also: **HeapAllocThrow, HeapDeref, HeapFree, HeapLock, HeapRealloc, HeapUnlock**

Example:

```
BYTE *bPtr1, *bPtr2, *bPtr3;
HANDLE hBlock1, hBlock2, hBlock3;
if (hBlock1 = HeapAlloc(1000)) {
   bPtr1 = HeapDeref( hBlock1 );
   /* use bPtr1 */
   if (hBlock2 = HeapAlloc(500)) {
      bPtr1 = HeapDeref( hBlock1 );  /* hBlock1 may have moved because of HeapAlloc */
      bPtr2 = HeapDeref( hBlock2 );
      /* can now user bPtr1 and bPtr2 */
      HeapLock( hBlock1 );  /* hBlock1 will NOT move */
      if (hBlock3 = HeapAlloc( 750 )) {
         bPtr2 = HeapDeref( hBlock2 );  /* hBlock2 may have moved, hBlock1 will not */
         bPtr3 = HeapDeref( hBlock3 );
         /* can now user bPtr1, bPtr2, and bPtr3 */
         HeapFree( hBlock3 );
      }
      HeapFree( hBlock2 );
   }
   HeapFree( hBlock1 );
}
```

HeapAllocHigh

Declaration: HANDLE **HeapAllocHigh** (DWORD *Hlen*)

Category(ies): Memory Management

Description: Allocate a block of heap memory at the high end of the heap, lock it and return its handle. This routine should NOT be used as a general heap allocation routine. It also compresses the heap first to (hopefully) move all used (unlocked) blocks of memory down. Blocks of memory that are locked for long periods of time should be moved high in memory so that they do not interfere as much with rest of the system.

Inputs: *Hlen* — Length of block of memory to allocate (all odd sizes are rounded up to be even).

Outputs: HANDLE of memory block allocated, H_NULL if not enough memory.

Assumptions: *Hlen* may not exceed 65520 bytes and the minimum block size is eight bytes.

Side Effects: This routine ALWAYS compresses the heap before it tries to allocate the requested memory and so is much slower than the standard **HeapAlloc** routine. Locking memory may cause the system to run out of useable memory sooner than if memory is kept unlocked.

Availability: All versions of the TI‑89 / TI‑92 Plus.

TI‑89 / TI‑92 Plus Differences: None

See Also: **HeapAlloc, HeapDeref, HeapFree**

Example:

```
HANDLE hPermament;
SYS_STRUCT *pSysStruct;
if (hPermament = HeapAllocHigh(500)) {
   /* hPermament is the handle to a block of memory that is needed through-out the
      life of this app and must be kept locked at all times */
   pSysStruct = HeapDeref( hPermament );  /* pSysStruct is always valid */
   /* . . . pSysStruct is used here . . . */
   HeapFree( hPermament );
}
```

HeapAllocHighThrow

Declaration: HANDLE **HeapAllocHighThrow** (DWORD *Hlen*)

Category(ies): Memory Management

Description: Allocate a block of heap memory at the high end of the heap, lock it and return its handle. This routine should NOT be used as a general heap allocation routine. It also compresses the heap first to (hopefully) move all used (unlocked) blocks of memory down. Blocks of memory that are locked for long periods of time should be moved high in memory so that they do not interfere as much with rest of the system.

Inputs: *Hlen* — Length of block of memory to allocate (all odd sizes are rounded up to be even).

Outputs: HANDLE of memory block allocated, throws an ER_MEMORY error if not enough memory (**HeapAllocHigh** just returns H_NULL in that case).

Assumptions: *Hlen* may not exceed 65520 bytes and the minimum block size is eight bytes.

Side Effects: This routine ALWAYS compresses the heap before it tries to allocate the requested memory and so is much slower than the standard **HeapAlloc** routine. Locking memory may cause the system to run out of useable memory sooner than if memory is kept unlocked.

This routine may throw an ER_MEMORY error.

Availability: All versions of the TI-89 / TI-92 Plus.

TI-89 / TI-92 Plus Differences: None

See Also: **HeapAllocHigh, HeapAllocThrow, HeapFree, HeapFreeIndir**

Example: See **HeapAllocThrow** substituting **HeapAllocHighThrow** for **HeapAllocThrow**.

HeapAllocThrow

Declaration: HANDLE **HeapAllocThrow** (DWORD *Hlen*)

Category(ies): Memory Management

Description: Allocate a block of heap memory of the given size and return its handle throwing a ER_MEMORY error if there is not enough memory (compared with **HeapAlloc** which returns H_NULL if there is not enough memory). Use **HeapDeref** to dereference the handle and get a pointer to the actual memory. Note that a pointer to the heap is valid only as long as heap compression is not done.

Inputs: *Hlen* — Length of block of memory to allocate (all odd sizes are rounded up to be even).

Outputs: HANDLE of memory block allocated, throws an ER_MEMORY error if not enough memory (**HeapAlloc** just returns H_NULL in that case).

Assumptions: *Hlen* may not exceed 65520 bytes and the minimum block size is eight bytes.

Side Effects: May cause heap compression.

This routine may throw an ER_MEMORY error.

Availability: All versions of the TI‑89 / TI‑92 Plus.

TI‑89 / TI‑92 Plus Differences: None

See Also: **HeapAlloc, HeapFree, HeapFreeIndir**

Example:

```
HANDLE hBlock1=0, hBlock2=0;  /* set to 0 for HeapFreeIndir */
TRY
   hBlock1 = HeapAllocThrow( 1000 );
   hBlock2 = HeapAllocThrow( 2000 );
   /* . . .   use hBlock1, hBlock2, hBlock 3 . . . */
   HeapFreeIndir( &hBlock1 );
   HeapFreeIndir( &hBlock2 );
ONERR
   /* Free memory we may have allocated, Note: HeapFreeIndir checks * HANDLE first */
   HeapFreeIndir( &hBlock1 );
   HeapFreeIndir( &hBlock1 );
   PASS;  /* pass error on up */
ENDTRY
}
```

HeapAvail

Declaration: DWORD **HeapAvail** (void)

Category(ies): Memory Management

Description: Return the total amount of free bytes in the heap (the sum of all of the individual blocks of memory).

Inputs: None

Outputs: The total number of bytes available in the heap.

Assumptions: None

Side Effects: None

Availability: All versions of the TI‑89 / TI‑92 Plus.

TI‑89 / TI‑92 Plus Differences: None

See Also: **HeapAlloc, HeapAllocThrow, HeapMax, HeapCompress**

Example:

```
static void AP_app(pFrame self, PEvent e) {

switch (e->command) {
   case CM_START:
      if (HeapAvail() < 20000) {
         myErrorMsg( "Not enough memory to start app" );
         EV_quit();
      }
   break;
```

HeapCompress

Declaration: void **HeapCompress** (void)

Category(ies): Memory Management

Description: Compress the heap.

Inputs: None

Outputs: None

Assumptions: None

Side Effects: This routine is called automatically by the system whenever it is needed and normally should not be called by applications.

Availability: All versions of the TI‑89 / TI‑92 Plus.

TI‑89 / TI‑92 Plus Differences: None

See Also: **HeapAlloc, HeapAllocThrow, HeapMax , HeapAvail**

Example:

```
/* This example is from the MEM key code. That code compresses the heap before
   calling HeapAvail since heap compression may combine some blocks of memory thus
   slightly changing the total amount available.
*/
DWORD TotalFree;
HeapCompress();
TotalFree = HeapAvail();
```

HeapDeref

Declaration: void * **HeapDeref** (HANDLE *h*)

Category(ies): Memory Management

Description: Dereferences a HANDLE and returns a pointer to the block of memory defined by that handle. The heap allocation routines return a HANDLE which is an identifier for a block of memory allocated in the heap. In order to use that memory, the handle must be dereferenced. Once a handle is dereferenced, that pointer is valid as long as nothing else is done to cause the heap to be compressed. If the heap is compressed the handle can be redereferenced to make it valid again. If a HANDLE is locked, then the pointer that references that block of memory is valid even after the heap is compressed (since locking a handle means the heap manager will never move the memory associated with that handle).

Inputs: *h* — Handle created with a heap allocation routine like **HeapAlloc** or **HeapAllocThrow**.

Outputs: Pointer to the block of memory defined by the given handle.

Assumptions: None

Side Effects: None

Availability: All versions of the TI-89 / TI-92 Plus.

TI-89 / TI-92 Plus Differences: None

See Also: **HeapAlloc, HeapAllocHigh, HeapAllocThrow, HeapAllocHighThrow, HeapLock, HeapUnlock, HeapFree, HeapFreeIndir**

Example: See **HeapAlloc**.

HeapFree

Declaration: void **HeapFree** (HANDLE *handle*)

Category(ies): Memory Management

Description: Frees the memory allocated for a handle.

Inputs: *handle* — Handle created with a heap allocation routine like **HeapAlloc** or **HeapAllocThrow**.

Outputs: None

Assumptions: None

Side Effects: None

Availability: All versions of the TI‑89 / TI‑92 Plus.

TI‑89 / TI‑92 Plus Differences: None

See Also: **HeapAlloc, HeapAllocHigh, HeapAllocThrow, HeapAllocHighThrow, HeapFreeIndir**

Example: See **HeapAlloc**.

HeapFreeIndir

Declaration: void **HeapFreeIndir** (HANDLE * *PtrHandle*)

Category(ies): Memory Management

Description: **HeapFreeIndir** is like **HeapFree** only you pass the address of a handle. If the handle that *PtrHandle* points to is not H_NULL then frees that handle and sets the handle that is pointed to by *PtrHandle* to H_NULL.

Inputs: *PtrHandle* — A pointer to a handle created with a heap allocation routine like **HeapAlloc** or **HeapAllocThrow**.

Outputs: None

Assumptions: None

Side Effects: None

Availability: All versions of the TI-89 / TI-92 Plus.

TI-89 / TI-92 Plus Differences: None

See Also: **HeapAlloc, HeapAllocHigh, HeapAllocThrow, HeapAllocHighThrow, HeapFree**

Example: See **HeapAllocThrow**.

HeapGetLock

Declaration: BOOL **HeapGetLock** (HANDLE *handle*)

Category(ies): Memory Management

Description: Return locked/unlocked status of *handle.*

Inputs: *handle* — Handle created with a heap allocation routine like **HeapAlloc** or **HeapAllocThrow**.

Outputs: Nonzero if *handle* references a block of locked heap memory or zero if that memory is free to move on the next heap compression.

Assumptions: None

Side Effects: None

Availability: All versions of the TI‑89 / TI‑92 Plus.

TI‑89 / TI‑92 Plus Differences: None

See Also: **HeapLock, HeapUnlock**

Example:

HeapLock

Declaration: void **HeapLock** (HANDLE *handle*)

Category(ies): Memory Management

Description: Lock the block of heap memory associated with *handle* so that it will not move on the next heap compression.

Inputs: *handle* — Handle created with a heap allocation routine like **HeapAlloc** or **HeapAllocThrow**.

Outputs: None

Assumptions: Memory allocated with **HeapAllocHigh** and **HeapAllocHighThrow** is locked to begin with.

Side Effects: Locking memory may cause the system to run out of useable memory sooner than if memory is kept unlocked.

Availability: All versions of the TI‑89 / TI‑92 Plus.

TI‑89 / TI‑92 Plus Differences: None

See Also: **HeapUnlock, HeapGetLock, HeapMoveHigh, HeapAllocHigh**

Example:

```
HANDLE hBlock;
BYTE *pBlock;
if (hBlock = HeapAlloc(100)) {
   HeapLock( hBlock );
   pBlock = HeapDeref( hBlock );
   /* pBlock may now be used even if the heap is compressed */
   DlgMessage(  "HeapLock", "pBlock points to locked memory" );
   /* can still use pBlock here */
   HeapUnlock( hBlock );
   DlgMessage( "HeapUnlock", "pBlock may now be invalid" );
   /* Cannot use pBlock here as the memory it points to may have moved */
   pBlock = HeapDeref( hBlock );
   /* Redereferenced hBlock so now pBlock is useable until next heap compression. */
   HeapFree( hBlock );
}
```

HeapMax

Declaration: DWORD **HeapMax** (void)

Category(ies): Memory Management

Description: Return the largest block of memory available to allocate (calls **HeapCompress** first).

Inputs: None

Outputs: The largest block of memory that may be allocated with one of the heap allocation routines (like **HeapAlloc**, **HeapAllocThrow**, . . .). This will be in the range 0 . . . 65520.

Assumptions: None

Side Effects: Compresses the heap.

Availability: All versions of the TI‑89 / TI‑92 Plus.

TI‑89 / TI‑92 Plus Differences: None

See Also: **HeapAlloc, HeapAllocThrow, HeapAvail, HeapCompress**

Example:

```
if (HeapMax() < 20000) {
   myErrorMsg( "NOT enough memory to allocate large array" );
   return FALSE;
}
```

HeapMoveHigh

Declaration: HANDLE **HeapMoveHigh** (HANDLE *handle*)

Category(ies): Memory Management

Description: Try to reallocate a block of heap memory as high in memory as possible. The block must not be locked. Use **HeapAllocHigh** if a block must be allocated high in memory when it is first allocated. **HeapMoveHigh** moves an existing block of memory to high memory. Blocks of memory that are locked for long periods of time should be moved high in memory so that they do not interfere as much with the rest of the system.

Inputs: None

Outputs: If successful, *handle* is returned. If the block cannot be moved then H_NULL is returned (the block is still in the same place as before, so no memory is lost).

Assumptions: None

Side Effects: Will cause heap compression.

Availability: All versions of the TI-89 / TI-92 Plus.

TI-89 / TI-92 Plus Differences: None

See Also: **HeapAlloc, HeapAllocThrow**

Example:

```
HANDLE hBlock;
void *vPtr;
if (hBlock = HeapAllocThrow(1000)) {
   /* . . . use hBlock . . . */
      Now if we need to keep a locked version around, move it high in memory
      and lock it.
   */
   HeapMoveHigh( hBlock );
   vPtr = HLock( hBlock );  /* LOCK and dereference block */
   /* . . . vPtr can now be used even if the heap is compressed . . .*/
   return( hBlock );  /* in this case, caller will free the block */
}
```

HeapPtrToHandle

Declaration: HANDLE **HeapPtrToHandle** (void const * *ptr*)

Category(ies): Memory Management

Description: Find the handle associated with a dereferenced pointer by searching the table of handle pointers for the given pointer.

Inputs: *ptr* — A dereferenced pointer to the heap.

Outputs: HANDLE of dereferenced pointer or H_NULL if not found.

Assumptions: The heap has not been compressed since the dereferenced pointer was originally obtained or the block it points to is locked.

Side Effects: This routine searches the entire handle table and so should be used accordingly.

Availability: All versions of the TI-89 / TI-92 Plus.

TI-89 / TI-92 Plus Differences: None

See Also: None

Example:

```
HANDLE h1, h2;
void *Ptr;
if (h1 = HeapAlloc(1000)) {
   Ptr = HLock( h1 );   /* Lock handle, return pointer to memory */
   h2 = HeapPtrToHandle( Ptr );   /* h2 better be same as h1 */
   if (h1 != h2)
      DlgMessage("ERROR", "Pointers do not match", PDB_CANCEL, 0 );
   HeapFree( h1 );   /* do not have to unlock it to free it */
}
```

HeapRealloc

Declaration: HANDLE **HeapRealloc** (HANDLE *handle*, DWORD *NewHsize*)

Category(ies): Memory Management

Description: Try to reallocate the given heap block to a new size. If *handle* is H_NULL, then just calls **HeapAlloc**. Return H_NULL if there is not enough memory (will try to compress the heap first) or the new size is invalid; otherwise return *handle* (will not change). If the block is LOCKED then it will not be moved for reallocation! The contents of the object will be unchanged up to the lesser of the new and old size. If the new size is larger the value of the newly allocated portion of the object is indeterminate.

Inputs:

handle — Handle of a heap block allocated with one of the heap allocation routines.

NewHsize — Length of block of memory to allocate (all odd sizes are rounded up to be even).

Outputs: Return *handle* if successful or H_NULL if not enough memory or invalid *NewHsize*.

Assumptions: None

Side Effects: May cause heap compression.

Availability: All versions of the TI‑89 / TI‑92 Plus.

TI‑89 / TI‑92 Plus Differences: None

See Also: **HeapAlloc, HeapAllocThrow, HeapFree**

Example:

```
HANDLE hBlock;
DWORD size = 1000;

if (hBlock = HeapAlloc( size )) {
   /* . . . use hBlock . . . */
   size = 100;  /* shrink block down */
   HeapRealloc( hBlock, size );  /* making smaller so will always succeed */
   return( hBlock );  /* assume caller will free it */
}
```

HeapShuffle

Declaration: void **HeapShuffle** (void)

Category(ies): Memory Management

Description: Shuffle the blocks in the heap. **HeapShuffle** is a debugging tool for tracking down problems with handles in an app. When a nonlocked handle is dereferenced, the resulting pointer is only valid until a heap compression is done. Heap compression is done internally when a heap allocation fails in order to try to coalesce free heap blocks together, then the heap allocation is retried and that result is returned to the caller. This routine helps track down such problems by forcing all dereferenced handles to nonlocked blocks of memory to change instead of the app relying on memory full cases.

Inputs: None

Outputs: None

Assumptions: None

Side Effects: The blocks of memory in the heap are rearranged. All dereferenced handles to nonlocked blocks of memory in the heap must be redereferenced.

Availability: On AMS 2.00 and higher.

TI-89 / TI-92 Plus Differences: None

See Also: **HeapWalk**

Example:

```
BYTE *bPtr1, *bPtr2;
char buf[128];
HANDLE hBlock1;

if (hBlock1 = HeapAlloc(1000)) {
   bPtr1 = HeapDeref( hBlock1 );
   HeapShuffle();
   bPtr2 = HeapDeref( hBlock1 );
   sprintf( buf, "Pointer before/after shuffle: %08lX, %08lX", bPtr1, bPtr2 );
   DlgNotice( "HeapShuffle", buf );
}
```

HeapSize

Declaration: DWORD **HeapSize** (HANDLE *handle*)

Category(ies): Memory Management

Description: Return the number of bytes allocated for the given heap block. Due to word alignment and minimum block size, this may not be the amount it was allocated with. Note that heap compression itself can in rare circumstances increase the size of a heap block by a couple of bytes; therefore it is best not to rely on **HeapSize** to determine the true size of a block of heap memory.

Inputs: *handle* — Handle of a heap block allocated with one of the heap allocation routines.

Outputs: Size of block of heap memory for *handle*.

Assumptions: None

Side Effects: None

Availability: All versions of the TI-89 / TI-92 Plus.

TI-89 / TI-92 Plus Differences: None

See Also: **HeapAlloc, HeapAllocThrow, HeapRealloc**

Example:

```
HANDLE h1, h2;
DWORD s1, s2;
h1 = HeapAlloc( 4 );
s1 = HeapSize( h1 );  /* will be 8 since that is the minimum block size */
h2 = HeapAlloc( 11 );
s2 = HeapSize( h2 );  /* will be 12 since all blocks must be even in length */
```

HeapUnlock

Declaration: void **HeapUnlock** (HANDLE *handle*)

Category(ies): Memory Management

Description: Unlock the block of heap memory associated with *handle* so that it may move on the next heap compression.

Inputs: *handle* — Handle created with a heap allocation routine like **HeapAlloc** or **HeapAllocThrow**.

Outputs: None

Assumptions: Memory allocated with **HeapAlloc** and **HeapAllocThrow** is unlocked to begin with.

Side Effects: None

Availability: All versions of the TI-89 / TI-92 Plus.

TI-89 / TI-92 Plus Differences: None

See Also: **HeapLock, HeapGetLock**

Example: See **HeapLock**.

HeapWalk

Declaration: BOOL **HeapWalk** (WORD *Function*)

Category(ies): Memory Management

Description: Walk the heap to verify it is valid, print the status of the heap, print the size of each heap block and its handle, or print the symbol table. The **HeapWalk** routine uses **LIO_SendData** to send the output to the link cable. You can connect the TI-89 or TI-92 Plus through the gray-link cable to a program like HyperTerminal to view the output.

Inputs: H_WALK_VERIFY, H_WALK_STATUS, H_WALK_DUMP, H_WALK_SYM

Outputs: TRUE is returned for functions H_WALK_VERIFY, H_WALK_STATUS, and H_WALK_DUMP if the heap is valid or FALSE if there is an error in the heap.

The resulting print to the link port depends on the input.

H_WALK_VERIFY	—	Prints nothing.
H_WALK_STATUS	—	Prints the total free space, maximum free block, number of used and free blocks, and the number of locked blocks.
H_WALK_DUMP	—	Prints the heap status and the size of the heap block for each handle in the system.
H_WALK_SYM	—	Prints the entire symbol table.

Assumptions: None

Side Effects: None

Availability: On AMS 2.00 and higher. H_WALK_SYM is only available on 2.04 and above. See **SymFindFoldername** for the code that does H_WALK_SYM.

TI-89 / TI-92 Plus Differences: None

See Also: **HeapShuffle**

(continued)

HeapWalk *(continued)*

Example: This is example output from calling **HeapWalk**(H_WALK_STATUS) and **HeapWalk**(H_WALK_SYM)

```
Total Free space:175332      Max FreeBlock:172086
Total Blocks:67      Free:9   Used:58  (0 Locked)

Name/Flags/hVal (dec)
FOLDER: main       0080  21
   main\pic4       0000  37
   main\str1       0000  39
   main\var1       0000  40
   main\xxx        0000  43
FOLDER: zfolder1   0080  28
zfolder1\f1        0000  24
```

HLock

Declaration: void * **HLock** (HANDLE *handle*)

Category(ies): Memory Management

Description: Lock the block of heap memory associated with *handle* so that it will not move on the next heap compression and then return the dereferenced value of the handle.

Inputs: *handle* — Handle created with a heap allocation routine like **HeapAlloc** or **HeapAllocThrow**.

Outputs: **HeapDeref** (**HeapLock** (*handle*))

Assumptions: Memory allocated with **HeapAllocHigh** and **HeapAllocHighThrow** is locked to begin with.

Side Effects: Locking memory may cause the system to run out of useable memory sooner than if memory is kept unlocked.

Availability: All versions of the TI‑89 / TI‑92 Plus.

TI‑89 / TI‑92 Plus Differences: None

See Also: **HeapUnlock, HeapGetLock, HeapMoveHigh, HeapAllocHigh**

Example:

```
HANDLE hBlock;
void *pBlock;
if (hBlock = HeapAlloc(100)) {
   pBlock = HLock( hBlock );
   /* . . . pBlock may now be used even if the heap is compressed . . . */
   HeapFree( hBlock );
}
```

memcpy

Declaration: void * **memcpy** (void * *dest*, const void * *source*, size_t *length*)

Category(ies): Memory Management

Description: Copies *length* bytes from *source* to *dest*. If some regions of *source* or *dest* overlap, the behavior of **memcpy** is undefined (use **memmove** for overlapping regions).

Inputs:
dest — Destination of move.
source — Source of move.
length — Size of move in bytes.

Outputs: *dest* is returned.

Assumptions: The source and destination regions must not overlap.

Side Effects: None

Availability: All versions of the TI-89 / TI-92 Plus.

TI-89 / TI-92 Plus Differences: None

See Also: **strncpy, memmove**

Example: See **strlen**, **MakeWinRect**, and **CalcBitmapSize**.

memmove

Declaration: void * **memmove** (void * *dest*, const void * *source*, size_t *length*)

Category(ies): Memory Management

Description: Copies *length* bytes from *source* to *dest*. Note that **memmove**, unlike **memcpy**, works properly even if some regions of *source* or *dest* overlap.

Inputs:
dest — Destination of move.
source — Source of move.
length — Size of move in bytes.

Outputs: *dest* is returned.

Assumptions: None

Side Effects: None

Availability: All versions of the TI-89 / TI-92 Plus.

TI-89 / TI-92 Plus Differences: None

See Also: **strncpy, memcpy**

Example:

```
char serno[11];

/* Assuming serno contains a 10 digit serial number, split it into two 5-digit
   groups. Must use memmove since the two regions overlap
*/
memmove( &serno[6], &serno[5], 6 );
serno[5] = ' ';
```

memset

Declaration: void * **memset** (void * *dest*, int *value*, size_t *count*)

Category(ies): Memory Management

Description: Sets the first *count* bytes of *dest* to the byte in *value.*

Inputs:
dest — Location to set.
value — Byte value to store.
count — Number of bytes to store.

Outputs: *dest* is returned.

Assumptions: None

Side Effects: None

Availability: All versions of the TI-89 / TI-92 Plus.

TI-89 / TI-92 Plus Differences: None

See Also: **memcpy**

Example: This example zeros out a FILES structure.

```
FILES *fsPtr;

memset( fsPtr, 0, sizeof(FILES) );
```

Appendix A: System Routines — Menus

See Also:

DynMenuAdd

Declaration: HANDLE **DynMenuAdd** (HANDLE *hMenu*, SWORD *ParentId*, const void * *Name*, short *Id*, WORD *Flags*);

Category(ies): Menus

Description: Add a new entry to a dynamic menu.

Inputs:

hMenu — Handle to a dynamic menu created with **MenuNew** or **MenuLoad**.

ParentId — ID of parent if adding a child entry (0 if adding top-level entry).

Name — Pointer to string (if DMF_TEXT set in *Flags*) or ICON (if DMF_ICON set in *Flags*) or BITMAP (if DMF_BITMAP set in *Flags*).

Id — ID of new entry. If *Id* is 0 then use sequentially numbered Ids. IDs are limited to the range 0x0001 . . . 0xFFF (12 bits).

> **NOTE:** If you are adding to a menu created with the resource compiler (using **MenuLoad**) do NOT use the range F00 . . . FFF.

Flags — One of the following flags must be set:

- DMF_TEXT — *Name* points to a text string.
- DMF_ICON — *Name* points to an ICON.
- DMF_BITMAP — *Name* points to a BITMAP.

Also, one of the following flags may be set:

- DMF_TOP — New top-level entry that cannot be a parent.
- DMF_TOP_SUB — New top-level entry that can have children.
- DMF_CHILD — Child of Parent (as specified by *ParentId*).
- DMF_CILD_SUB — Child of Parent (*ParentId*) that can also have children.

Outputs: *hMenu* if successful, H_NULL if out of memory or error in parameters (*ParentId* not found, *ParentId* found but it was not a possible parent or max-items in a menu exceeded).

Note that if there is an error adding the new entry, the MF_ERROR bit in **MenuFlags**(*hMenu*) is set.

(continued)

DynMenuAdd *(continued)*

Assumptions: See **MenuNew** for details on creating and using dynamic menus.

Side Effects: May cause heap compression.

Availability: On AMS 2.00 and higher.

TI-89 / TI-92 Plus Differences: None

See Also: **MenuNew, MenuLoad, DynMenuChange**

Example:

```
HANDLE hNewMenu, hDrawMenu;
WORD key;
static const WORD IconPencil[] = {0x0000, 0x0000, 0x0018, 0x0024, 0x0074, 0x0098,
   0x0110, 0x0220, 0x0440, 0x0880, 0x1900, 0x1E00, 0x1C00, 0x1800, 0x1000, 0x0000};
if (!(hNewMenu = MenuNew( 0, 240, 0 )))
   ER_throw(ER_MEMORY);
DynMenuAdd( hNewMenu ,0, "TOP 1", 1, DMF_TEXT );
DynMenuAdd( hNewMenu, 0, "TOP 2", 2, DMF_TEXT );
DynMenuAdd( hNewMenu, 1, "SUB 1", 11, DMF_CHILD | DMF_TEXT);
DynMenuAdd( hNewMenu, 1, IconPencil, 12, DMF_CHILD | DMF_ICON);
if (MenuFlags(hNewMenu) & MF_ERROR) {
   HeapFree( hNewMenu );
   ER_throw(ER_MEMORY);
}
if (!(hDrawMenu = MenuBegin( HLock(hNewMenu), -1, 0, TRUE )))  {
   HeapFree( hNewMenu );
   ER_throw(ER_MEMORY);
}
key = MenuKey( hDrawMenu, KB_F1 );
MenuEnd( hDrawmenu );
HeapFree( hNewMenu );
```

DynMenuChange

Declaration: HANDLE **DynMenuChange** (HANDLE *hMenu*, short *Id*, const void * *newName*, WORD *Flags*);

Category(ies): Menus

Description: Change a previous entry in a dynamic menu.

Inputs:

hMenu — Handle to a dynamic menu created with **MenuNew** or **MenuLoad**.

Id — ID of entry to change.

newName — Pointer to new text string, ICON or BITMAP.

Flags — One of the following flags must be set:
- DMF_TEXT — *Name* points to a text string.
- DMF_ICON — *Name* points to an ICON.
- DMF_BITMAP — *Name* points to a BITMAP.

Outputs: *hMenu* if successful, H_NULL if out of memory or *Id* not found.

Note that if there is an error adding the new entry, the MF_ERROR bit in **MenuFlags**(*hMenu*) is set.

Assumptions: None

Side Effects: May cause heap compression.

Availability: On AMS 2.00 and higher.

TI-89 / TI-92 Plus Differences: None

See Also: **MenuNew, MenuLoad, DynMenuAdd**

Example: See **PopupBegin**.

FKeyI_H

Declaration: WORD **FKeyI_H** (HANDLE *MenuH*, WORD *Key*)

Category(ies): Menus

Description: For the given function key, return its index relative to KB_F1 (KB_F1 -> 0, . . . , F8 -> 7) or M_NOTMENUKEY if not a function key.

Inputs: *hMenu* — Handle returned from **MenuBegin**.

Key — Key code.

Outputs: 0 . . . 7 if menu key or M_NOTMENUKEY. (Note that M_NOTMENUKEY may be returned for function keys not defined for the given menu handle.)

Assumptions: None

Side Effects: None

Availability: On AMS 2.02 and higher.

TI-89 / TI-92 Plus Differences: None

See Also: **MenuBegin, MenuKey**

Example:

```
HANDLE hMenuDraw;
WORD key, menuSelect  = 0;
const MENU ToolBox2;
if (hMenuDraw = MenuBegin(&ToolBox2, 0,0, 0 )) {
   key = GKeyIn( NULL, 0 );
   if (M_NOTMENUKEY == FKeyI_H(hMenuDraw, key))
      Disp("Key not on this menu");
   else
      menuSelect = MenuKey( hMenuDraw, key );
   MenuEnd( hMenuDraw );
   return menuSelect;
}
```

MenuAddIcon

Declaration: HANDLE **MenuAddIcon** (HANDLE *hMenu*, SWORD *ParentId*, const WORD * *Icon*, short *Id*, WORD *Flags*);

Category(ies): Menus

Description: Add an icon to a dynamic menu. *Icon* points to an array of 16 words that defines a 16x16 icon.

Inputs:

hMenu — Handle to a dynamic menu created with **MenuNew** or **MenuLoad**.

ParentId — ID of parent if adding a child entry (0 if adding top-level entry).

Icon — Pointer to ICON.

Id — ID of new entry. If *Id* is 0 then use sequentially numbered Ids. IDs are limited to the range 0x0001 . . . 0xFFF (12 bits).

> **NOTE:** If you are adding to a menu created with the resource compiler (using **MenuLoad**) do NOT use the range F00 . . . FFF.

Flags — Must be one of the following:

- DMF_TOP — New top-level entry that cannot be a parent.
- DMF_TOP_SUB — New top-level entry that can have children.
- DMF_CHILD — Child of parent (as specified by *ParentId*).
- DMF_CILD_SUB — Child of parent (*ParentId*) that can also have children.

Outputs: *hMenu* if successful, H_NULL if out of memory or error in parameters.

Note that if there is not enough memory to add the new entry, the MF_ERROR bit in **MenuFlags**(*hMenu*) is set.

Assumptions: None

Side Effects: May compress the heap.

Availability: All versions of the TI-89 / TI-92 Plus.

TI-89 / TI-92 Plus Differences: None

See Also: **MenuNew, MenuLoad, DynMenuAdd, DynMenuChange**

Example: See **MenuAddText**.

MenuAddText

Declaration: HANDLE **MenuAddText** (HANDLE *hMenu*, SWORD *ParentId*, const char * *Name*, short *Id*, WORD *Flags*);

Category(ies): Menus

Description: Add a text entry to a dynamic menu.

Inputs:

hMenu — Handle to a dynamic menu created with **MenuNew** or **MenuLoad**.

ParentId — ID of parent if adding a child entry (0 if adding top-level entry).

Name — Pointer to ICON.

Id — ID of new entry. If *Id* is 0 then use sequentially numbered Ids. IDs are limited to the range 0x0001 . . . 0xFFF (12 bits).

> **NOTE:** If you are adding to a menu created with the resource compiler (using **MenuLoad**) do NOT use the range F00 . . . FFF.

Flags — Must be one of the following:

- DMF_TOP — New top-level entry that cannot be a parent.
- DMF_TOP_SUB — New top-level entry that can have children.
- DMF_CHILD — Child of parent (as specified by *ParentId*).
- DMF_CILD_SUB — Child of parent (*ParentId*) that can also have children.

Outputs: *hMenu* if successful, H_NULL if out of memory or error in parameters.

Assumptions: None

Side Effects: May compress the heap.

Availability: All versions of the TI-89 / TI-92 Plus.

TI-89 / TI-92 Plus Differences: None

See Also: **MenuNew, MenuLoad, DynMenuAdd, DynMenuChange**

(continued)

MenuAddText *(continued)*

Example:

```
HANDLE hNewMenu, hDrawMenu;
WORD key;
static const WORD IconEraser[] = {0x0000, 0x0000, 0x007E, 0x0086, 0x010A, 0x0212,
   0x0422, 0x0844, 0x1088, 0x2110, 0x7E20, 0x7E40, 0x7E80, 0x7F00, 0x0000, 0x0000};
if (!(hNewMenu = MenuNew( 0, 240, 0 )))
   ER_throw(ER_MEMORY);
MenuAddText( hNewMenu ,0, "TOP 1", 1, 0 );
MenuAddText( hNewMenu, 0, "TOP 2", 2, 0 );
MenuAddText( hNewMenu, 1, "SUB 1", 11, DMF_CHILD);
MenuAddIcon( hNewMenu, 1, &IconEraser[0], 12, DMF_CHILD);
if (MenuFlags(hNewMenu) & MF_ERROR) {
   HeapFree( hNewMenu );
   ER_throw(ER_MEMORY);
}
if (!(hDrawMenu = MenuBegin(NULL, -1, 0, MBF_HMENU, hNewMenu )))  {
   HeapFree( hNewMenu );
   ER_throw(ER_MEMORY);
}
key = MenuKey( hDrawMenu, KB_F1 );
MenuEnd( hDrawmenu );  /* will also free hNewMenu since we set MBF_HMENU */
```

MenuBegin

Declaration: HANDLE **MenuBegin** (const MENU * *Menu*, short *x0*, short *y0*, SINT Flags, . . .)

Category(ies): Menus

Description: Open the top level menu for the given menu structure, allocate any internal data and return a handle to the menu-draw structure or H_NULL if there is not enough memory to open the menu.

Inputs:

Menu — Pointer to a MENU structure (as defined by the resource compiler or a dynamic menu).

x0, y0 — Screen coordinates of the upper left corner of the menu. If *x0* is equal to -1 then the menu is centered horizontally, if *y0* is equal to -1 then the menu is centered vertically on the screen.

Flags — MBF_REDEF

Allow for the top-level menu items (special text/icon combination) to be redefined with the **MenuTopRedef** function.

> **Note:** This only works for static menus created with the resource compiler (using the MF_ALT_ICONS flag in the TOOLBOX definition) and cannot be used for dynamic menus.

MBF_MAX_MENU_WIDTH

The parameter after *Flags* (SINT) will be the maximum field width to use for the menu (by default it is the screen width) — this maximum field width is only used if the menu was given a 0 width in the resource file (or in a dynamic menu).

MBF_STRIKEOUT

Use strikeout (line drawn through text or ICON fields) instead of gray-out to indicate inactive menu items.

MBF_HMENU

The parameter after *Flags* (HANDLE) is the handle of a dynamically created menu. *Menu* is ignored and this handle is locked and dereferenced and used instead of *Menu*. The handle is saved and calling **MenuEnd** on the handle returned by **MenuBegin** will also free this handle.

(continued)

MenuBegin *(continued)*

Inputs: ***(continued)***

MBF_NO_DRAWTOP
Setup the menu-draw structure and return a handle to it but do not draw the menu (caller must call **MenuOn** to draw it).

> **NOTE:** If both MAX_MENU_WIDTH and MBF_HMENU are set, the first parameter after *Flags* is the maximum menu width and the second parameter is the handle to a dynamic menu.

Outputs: HANDLE of the newly created menu-draw structure. (Note that this handle is passed to menu routines like **MenuCheck**, **MenuKey**, **MenuOn**, **MenuTopStat**.) It is separate from the handle returned by **MenuNew** for dynamic menus. H_NULL is returned if there is not enough memory to create the new structure.

Assumptions: None

Side Effects: May cause heap compression.

Availability: All versions of the TI‑89 / TI‑92 Plus.

TI‑89 / TI‑92 Plus Differences:

	TI‑89	TI‑92 Plus
Max menu width	159	239
Top level font	F_4x6	F_6x8
Sub level font	F_6x8	F_6x8

See Also: **MenuEnd, MenuKey, MenuCheck, MenuSubStat, MenuTopStat, MenuOn, MenuOff, MenuTopRedef**

Example:

```
HANDLE hMenuDraw;
WORD key, menuSelect;
const MENU testMenu;  /* defined by resource compiler */
if (hMenuDraw = MenuBegin(&testMenu, 0,0, 0 )) {
   key = GKeyIn( NULL, 0 );
   menuSelect = MenuKey( hMenuDraw, key );
   MenuEnd( hMenuDraw );
   return menuSelect;
}
```

MenuCheck

Declaration: short **MenuCheck** (HANDLE *MenuHandle*, SINT *MenuId*, short *Cmd*)

Category(ies): Menus

Description: Set, clear, flip, or return the status of a checkmark for a menu item.

Inputs:

MenuHandle — Handle returned from **MenuBegin**.

MenuId — ID of menu item.

Cmd — For the menu item associated with **MenuId**.

MC_CHECK	—	Display a check-mark next to the menu item.
MC_UNCHECK	—	Remove check-mark.
MC_STATUS	—	Return status of check-mark (zero: not checked, nonzero: checked).
MC_FLIP	—	Invert the status of the checkmark.

Outputs: For MC_STATUS return the current status of the checkmark. For the other *Cmd*s return TRUE if *MenuId* found, FALSE if not.

Assumptions: *MenuHandle* was created by **MenuBegin**.

Side Effects: None

Availability: All versions of the TI‑89 / TI‑92 Plus.

TI‑89 / TI‑92 Plus Differences: None

See Also: **MenuBegin**

Example: See **MenuTopStat**.

MenuEnd

Declaration: void **MenuEnd** (HANDLE *MenuHandle*)

Category(ies): Menus

Description: Terminate the given menu-draw handle. If the MBF_NO_DRAWTOP flag was NOT passed to **WinBegin** then clear the area underneath the menu. If MBF_HMENU (and a dynamic menu/pop-up handle) was passed to **MenuBegin** then that dynamic handle is freed. Finally, the memory allocated to *MenuHandle* is freed.

Inputs: *MenuHandle* — Handle returned from **MenuBegin**.

Outputs: None

Assumptions: None

Side Effects: None

Availability: All versions of the TI-89 / TI-92 Plus.

TI-89 / TI-92 Plus Differences: None

See Also: **MenuBegin, MenuKey**

Example: See **MenuBegin**.

MenuFlags

Declaration: SINT **MenuFlags** (HANDLE *mH*)

Category(ies): Menus

Description: Return the flag word for a dynamic menu/pop-up structure.

Inputs: *mH* — Handle returned by **MenuNew** or **PopupNew**.

Outputs: The only useful flag bit is MF_ERROR which is cleared when the menu structure is created and set if adding or changing a menu entry causes a memory error.

Assumptions: None

Side Effects: None

Availability: On AMS 2.00 and higher.

TI-89 / TI-92 Plus Differences: None

See Also: **MenuBegin, MenuKey**

Example: See **MenuAddText**.

MenuGetTopRedef

Declaration: SINT **MenuGetTopRedef** (HANDLE *MenuHandle*, SWORD *Index*)

Category(ies): Menus

Description: Return the current value of a redefinable top-level menu item (0 if not redefined or redefinable).

Inputs: *MenuHandle* — Handle returned from **MenuBegin**.

Index — 0 . . . number of top-level items -1.

Outputs: Menu ID (as defined in the MENU structure) for the selected menu item.

Assumptions: *MenuHandle* refers to a menu-draw handle with the MBF_REDEF flag bit set when it was started with **MenuBegin**. Redefinable menus may only be created with the resource compiler.

Side Effects: None

Availability: All versions of the TI-89 / TI-92 Plus.

TI-89 / TI-92 Plus Differences: None

See Also: **MenuBegin, MenuTopRedef, MenuTopSelect**

Example: See **MenuTopRedef** for an example and further details.

MenuItemDef

Declaration: void * **MenuItemDef** (HANDLE *mH*, WORD *Id*, WORD * *type*)

Category(ies): Menus

Description: Given a menu *Id*, return a pointer to the text, ICON, or BITMAP defining it; NULL if *Id* not found (* *type* set to 0) for a given menu handle. Return its type in *type*.

Inputs: *mH* — Handle returned from **MenuBegin** for menu to search.

Id — Menu ID.

Outputs: Pointer to item's definition.

* *type* — DMF_TEXT, DMF_ICON, DMF_BITMAP, or 0 if *Id* not found.

Assumptions: None

Side Effects: None

Availability: On AMS 2.00 and higher.

TI-89 / TI-92 Plus Differences: None

See Also: **MenuBegin, DynMenuAdd, DynMenuChange**

Example:

```
HANDLE mH;
WORD Id, type;
void *vPtr;

.
.
.
if (Id = MenuKey(mH, KB_F1)) {
   if (vPtr = MenuItemDef( mH, Id, &type )) {
      /* vPtr points to item selected */
      /* type will be DMF_TEXT, DMF_ICON or DMF_BITMAP depending on the type
         of the menu item */
      .
      .
      .
   }
}
```

MenuKey

Declaration: WORD **MenuKey** (HANDLE *MenuHandle*, WORD *Key*)

Category(ies): Menus

Description: Handle a *Key* for a menu returning the menu item selected.

Inputs:

MenuHandle	—	Menu-draw handle returned by **MenuBegin**.
Key	—	KB_F1 . . . KB_F8

Outputs:

0	—	Nothing selected (ESC pressed) or not enough memory to display menu.
1 . . . 0xFFF	—	Menu ID of item selected.
M_NOTMENUKEY	—	*Key* is not in range for this menu.

Assumptions: None

Side Effects: May cause heap compression.

Availability: All versions of the TI-89 / TI-92 Plus.

TI-89 / TI-92 Plus Differences: None

See Also: **MenuBegin**

Example: See **MenuBegin**.

MenuLoad

Declaration: HANDLE **MenuLoad** (const MENU * *BaseMenu*, WORD *Size*)

Category(ies): Menus

Description: Begin a dynamically created menu, using a ROM based menu as the starting point (the menu's flags, width, and height are stored in the ROM structure). Return the HANDLE of the new dynamically created menu which may be used in **DynMenuAdd** or **DynMenuChange** and then passed to **MenuBegin** to draw the menu.

Inputs:

BaseMenu — The address of a MENU structure created by the resource compiler.

Size — The number of bytes in the MENU (the resource compiler always defines a label with the base name of the MENU and "_end" appended which may be used to determine the length — see example below).

Outputs: A dynamic menu handle or H_NULL if not enough memory.

Assumptions: None

Side Effects: May cause heap compression.

Availability: On AMS 2.00 and higher.

TI-89 / TI-92 Plus Differences: None

See Also: **MenuBegin, MenuNew, DynMenuAdd, DynMenuChange**

Especially see **MenuNew** for restrictions on creating and using dynamic menus.

(continued)

MenuLoad *(continued)*

Example:

```
HANDLE hMenu, hDraw;
const MENU AddToMenu_end, AddToMenu;

if (hMenu = MenuLoad( &AddToMenu, (const BYTE *) &AddToMenu_end - (const BYTE *)
    &AddToMenu )) {
   DynMenuAdd(hMenu, 0, "NEW TOP", 30, DMF_TEXT | DMF_TOP_SUB );
   DynMenuAdd(hMenu, 30, "NEW SUB1", 31, DMF_TEXT | DMF_CHILD );
   DynMenuAdd(hMenu, 30, "NEW SUB2", 32, DMF_TEXT | DMF_CHILD );
   DynMenuAdd(hMenu, 20, "ADDED TO TOP2", 21, DMF_TEXT | DMF_CHILD );
   if (!(MenuFlags(hMenu) & MF_ERROR)) {
      if (hDraw = MenuBegin( 0, 0, 0, MBF_HMENU, hMenu )) {
         MenuKey( hDraw, KB_F1 );
         MenuEnd( hDraw );
         return;
      }
   }
   HeapFree( hMenu );
}

APPR.R:
#include "tiams.h"
TOOLBOX AddToMenu, RC_NO_IDS, 0, 0 {
   "TOP 1", 10 {
      "SUB 1", 11
      "SUB 2", 12
   }
   "TOP 2", 20 {
   }
}
```

MenuNew

Declaration: HANDLE **MenuNew** (SINT *Flags*, SINT *Width*, SINT *Height*)

Category(ies): Menus

Description: Begin a dynamically created menu.

Inputs:
Flags — Always pass 0.
Width — 0 (calculate) or requested width of menu bar.
Height — 0 (use default) or requested height of menu bar.

Outputs: Returns a handle to an empty MENU structure or H_NULL if not enough memory.

Assumptions:

Note:	There is a STRICT method of using dynamic menus.

1. Create an empty, dynamic menu structure with **MenuNew** or **MenuLoad**.
2. Build the menu with **DynMenuAdd** or **DynMenuChange** using the handle returned by **MenuNew** or **MenuLoad** (each of these routines returns H_NULL if not enough memory, or check **MenuFlags** when done with all of the additions/changes).
3. Call **MenuBegin** setting the MBF_HMENU flag and passing the handle returned by **MenuNew** or **MenuLoad** as the argument after the Flags parameter (**MenuBegin** has a variable number of arguments). NULL can be passed as the pointer to the MENU structure (since the dereferenced handle points to the MENU structure).
4. This will lock the handle returned from **MenuNew** and save it.
5. Using the handle returned from **MenuBegin** (this is a separate handle!) you may then call all of the normal menu functions (**MenuCheck**, **MenuKey**, **MenuOn**, **MenuTopStat**, **MenuTopSelect**).
6. When done with the menu, call **MenuEnd** on the handle returned from **MenuBegin**. This will free the handle returned from **MenuBegin** AND the handle returned from **MenuNew**.

Do not forget that once you call **MenuBegin** you may not unlock the handle returned from **MenuNew** or call **DynMenuAdd** or **DynMenuChange**!

(continued)

MenuNew *(continued)*

Side Effects: May cause heap compression.

Availability: All versions of the TI-89 / TI-92 Plus.

TI-89 / TI-92 Plus Differences: None

See Also: **MenuBegin, MenuKey, MenuLoad, DynMenuAdd, DynMenuChange**

Example:

```
void MenuTestA( void )
{
   HANDLE hDynMenu, hMenuDraw;

   if (hDynMenu = MenuNew(0, 0, 0)) {
      DynMenuAdd( hDynMenu, 0, "TOP1", 10, DMF_TOP_SUB | DMF_TEXT );
      DynMenuAdd( hDynMenu, 10, "SUB1-A", 11, DMF_CHILD | DMF_TEXT );
      DynMenuAdd( hDynMenu, 10, "SUB1-B", 12, DMF_CHILD | DMF_TEXT );
      DynMenuAdd( hDynMenu, 10, "SUB1-C", 13, DMF_CHILD_SUB | DMF_TEXT );
      DynMenuAdd( hDynMenu, 0, "TOP2", 20, DMF_TOP | DMF_TEXT );
      DynMenuAdd( hDynMenu, 13, "CHILD SUB-A", 131, DMF_CHILD | DMF_TEXT );
      DynMenuAdd( hDynMenu, 13, "CHILD SUB-B", 132, DMF_CHILD | DMF_TEXT );
      if (!(MenuFlags(hDynMenu) & MF_ERROR)) {
         if (hMenuDraw = MenuBegin( NULL, 0, 0, MBF_HMENU, hDynMenu )) {
            MenuKey( hMenuDraw, KB_F1 );
            MenuEnd( hMenuDraw );
            return;
         }
      }
      HeapFree( hDynMenu );
      ER_throwVar( ER_MEMORY );
   }
}
```

MenuOff

Declaration: void **MenuOff** (HANDLE *MenuHandle*)

Category(ies): Menus

Description: Gray-out the top-level of the menu defined by *MenuHandle*.

Inputs: *MenuHandle* — Handle returned from **MenuBegin**.

Outputs: None

Assumptions: None

Side Effects: None

Availability: On AMS 2.00 and higher.

TI-89 / TI-92 Plus Differences: None

See Also: **MenuBegin, MenuOn**

Example:

```
if (mode == DISABLE_MENU) {
   MenuOff( hMenuHandle );  /* Disable entire menu */
   .
   .
   .
   MenuOn( hMenuHandle);  /* turn menu back on in case it was off */
}
```

MenuOn

Declaration: void **MenuOn** (HANDLE *MenuHandle*)

Category(ies): Menus

Description: Draws the menu defined by *MenuHandle*. Note that items that were disabled with **MenuTopStat** remain disabled and therefore shaded.

Inputs: *MenuHandle* — Handle returned from **MenuBegin**.

Outputs: None

Assumptions: None

Side Effects: None

Availability: All versions of the TI‑89 / TI‑92 Plus.

TI‑89 / TI‑92 Plus Differences: None

See Also: **MenuBegin, MenuOff, MenuTopStat**

Example: See **MenuTopStat**.

MenuPopup

Declaration: WORD **MenuPopup** (const MENU * *Popup*, short *x0*, short *y0*, short *Offset*)

Category(ies): Menus

Description: Execute a static POPUP as defined by the resource compiler, returning the item selected.

Inputs:

Menu	—	Pointer to a MENU structure of a POPUP as defined by the resource compiler.
x0, *y0*	—	Screen coordinates of the upper left corner of the menu. If *x0* is equal to -1 then the pop-up is centered horizontally, if *y0* is equal to -1 then the pop-up is centered vertically.
Offset	—	Menu ID of initially selected item (0 defaults to first item).

Outputs:

0	—	Nothing selected (ESC pressed) or not enough memory to display pop-up.
1 . . . 0xFFF	—	Menu ID of item selected.

Assumptions: **MenuPopup** is only used for static POPUPs created by the resource compiler.

Side Effects: May cause heap compression.

Availability: All versions of the TI-89 / TI-92 Plus.

TI-89 / TI-92 Plus Differences: None

See Also: **PopupDo** (dynamic popups)

Example:

```
#include "mpopup.h"

WORD select;
if (MID_CORRECT == (select = MenuPopup( &mPopupTest, -1, -1, 3 )))
   Disp("CORRECT");

// MPOPUP.R
POPUP mPopupTest, 0, 0
{
   "POPUP 1", MID_1
   "POPUP 2", MID_2
   "SELECT THIS", MID_CORRECT
}
```

MenuSubStat

Declaration: void **MenuSubStat** (HANDLE *MenuHandle*, SINT *Index*, SINT *Status*)

Category(ies): Menus

Description: Enable/Disable a sublevel menu item.

Inputs:

MenuHandle — Menu-draw handle from **MenuBegin**.

Index — Menu ID for the sublevel menu item to enable/disable.

Status — If *Status* is TRUE the item is enabled (normal text/icon, can be selected) otherwise it is disabled (grayed-out text/icon, item cannot be selected).

Outputs: None

Assumptions: None

Side Effects: None

Availability: All versions of the TI‑89 / TI‑92 Plus.

TI‑89 / TI‑92 Plus Differences: None

See Also: **MenuTopStat** (for top-level menu items)

Example: See **MenuTopStat**.

MenuTopRedef

Declaration: void **MenuTopRedef** (HANDLE *MenuHandle*, SWORD *Index*, SINT *MenuId*)

Category(ies): Menus

Description: Redefine a top-level menu item Icon for a menu that was started with the MBF_REDEF flag set when **MenuBegin** was called and created with the resource compiler (using the MF_ALT_ICONS flag). The selected top-level Icon is redrawn with the Icon of one of its submenus.

Inputs:

MenuHandle — Handle returned from **MenuBegin**.

Index — 0 . . . number of top-level items – 1.

MenuId — New menu ID for the top-level item specified by *Index*. Note that *MenuId* must be the menu ID for one of the submenus for the top-level item selected.

Outputs: None

Assumptions: *MenuHandle* refers to a menu-draw handle with the MBF_REDEF flag bit set when it was started with **MenuBegin**. Redefinable menus may only be created with the resource compiler. The redefined item must have text and an icon defined for it (see example below). The original top-level item may be defined with the special DUMMY keyword in which case there is no default selection or it may have a default icon specified and then it will not have to be initialized with **MenuTopRedef**.

Side Effects: None

Availability: All versions of the TI-89 / TI-92 Plus.

TI-89 / TI-92 Plus Differences: None

See Also: **MenuBegin, MenuGetTopRedef, MenuTopSelect**

(continued)

MenuTopRedef *(continued)*

Example: This example uses an include file, a resource file, and an icon file as specified after the example.

```
#include "appr.h"
void TestRedef( void ) {
   HANDLE hMenu;
   SINT select, fTop;
   char buf[100];

   if (hMenu = MenuBegin( &RedefMenu, 0, 0, MBF_REDEF )) {
      MenuTopRedef( hMenu, 0, MID_ERASER );
      MenuTopRedef( hMenu, 1, MID_ALPHA );
      MenuTopSelect( hMenu, 0 );
      do {
         select = MenuKey( hMenu, KB_F1 );
         switch (select) {
            case MID_ERASER: case MID_PENCIL: case MID_RBBOX: fTop = 0; break;
            case MID_GRAPH: case MID_ALPHA: fTop = 1; break;
            default: MenuEnd(hMenu); return;
         }
         MenuTopRedef( hMenu, fTop, select );
         MenuTopSelect( hMenu, fTop );
         sprintf( buf, "You selected %d", MenuGetTopRedef( hMenu, fTop ) );
         DlgNotice( "TEST", buf );
      } while (1);
   }
}

// app.h
#define MID_TOOLS     1
#define MID_CURSORS   2
#define MID_EXIT      3
#define MID_ERASER    4
#define MID_PENCIL    5
#define MID_RBBOX     6
#define MID_GRAPH     7
#define MID_ALPHA     8

// appr.r
#include "app.h"
#include "tiams.h"
```

(continued)

MenuTopRedef *(continued)*

```
TOOLBOX RedefMenu, RC_NO_IDS | MF_ALT_ICONS, 0, 0 {
   DUMMY {
      "ERASER", MID_ERASER, *appr.ico, ICON_ERASER
      "PENCIL", MID_PENCIL, *appr.ico, ICON_PENCIL
      "BOX", MID_RBBOX,  *appr.ico, ICON_RBBOX
   }
   DUMMY {
      "GRAPH", MID_GRAPH, *appr.ico, ICON_GRAPH
      "ALPHA", MID_ALPHA, *appr.ico, ICON_ALPHA
   }
   "EXIT", MID_EXIT
}

// appr.ico
[0x0000, 0x0000, 0x0000, 0x7777, 0x4001, 0x4001, 0x0000, 0x4001, 0x4001, 0x4001,
   0x0000, 0x4001, 0x4001, 0x7777, 0x0000, 0x0000], ICON_RBBOX
[0x0000,0x0000,0x0018,0x0024,0x0074,0x0098,0x0110,0x0220, 0x0440, 0x0880, 0x1900,
   0x1E00, 0x1C00, 0x1800, 0x1000, 0x0000], ICON_PENCIL
[0x0000, 0x0000, 0x007E, 0x0086, 0x010A, 0x0212, 0x0422, 0x0844, 0x1088, 0x2110,
   0x7E20, 0x7E40, 0x7E80, 0x7F00, 0x0000, 0x0000], ICON_ERASER
[0x0000, 0x0000, 0x0000, 0x0100, 0x0100, 0x0100, 0x0100, 0x0100, 0x7FFC, 0x0100,
   0x0100, 0x0100, 0x0100, 0x0100, 0x0000, 0x0000], ICON_GRAPH
[0x0000, 0x0000, 0x0180, 0x0180, 0x03C0, 0x03C0, 0x0660, 0x0660, 0x0C30,
   0x0FF0, 0x1FF8, 0x1818, 0x380C, 0x700C, 0xF81E, 0x0000], ICON_ALPHA
```

MenuTopSelect

Declaration: void **MenuTopSelect** (HANDLE *MenuHandle*, SWORD *Index*)

Category(ies): Menus

Description: Select a top-level menu item (only one may be selected at a time) by drawing a thick box around the menu item. Use a value of -1 for *Index* to force no top-level menu item to be selected.

Inputs:

MenuHandle — Handle returned from **MenuBegin**.

Index — 0 to the number of top-level items minus 1 or -1 to deselect the currently highlighted item.

Outputs: None

Assumptions: *MenuHandle* refers to a menu-draw handle returned from **MenuBegin**. **MenuTopSelect** is normally used with redefinable menus.

Side Effects: None

Availability: All versions of the TI-89 / TI-92 Plus.

TI-89 / TI-92 Plus Differences: None

See Also: **MenuBegin, QMenuTopSelect**

Example: See **MenuTopRedef** for an example of using redefinable menus and **MenuTopSelect**.

MenuTopStat

Declaration: void **MenuTopStat** (HANDLE *MenuHandle*, SINT *Index*, SINT *Status*)

Category(ies): Menus

Description: Enable/Disable a top-level menu item.

Inputs:

MenuHandle — Menu-draw handle from **MenuBegin**.

Index — 0 . . . number of top-level menu items less one (not all items may be displayed at once depending on how many there are, their size and the size of the screen).

Status — If Status is TRUE the item is enabled (normal text/icon, can be selected) otherwise it is disabled (grayed-out text/icon, cannot be selected).

Outputs: None

Assumptions: The effects of enabling or disabling a top-level menu item are not drawn until **MenuOn** is called to redraw the menu. Note in the following example that **MenuOn** is called after **MenuTopStat** (there could have been several **MenuTopStat** calls) and before **MenuKey**.

Side Effects: None

Availability: All versions of the TI-89 / TI-92 Plus.

TI-89 / TI-92 Plus Differences: None

See Also: **MenuSubStat** (for sublevel items)

Example:

```
HANDLE hMenu;
if (hMenu = MenuBegin( &SelectMenu, 0, 0, 0 )) {
   MenuCheck( hMenu, MID_SUB1, MC_CHECK );  /* Check SUB1 */
   MenuKey( hMenu, KB_F1 );
   MenuTopStat( hMenu, 1, FALSE );  /* Disable TOP item "SECOND" */
   MenuSubStat( hMenu, MID_SUB2, FALSE );  /* Disable submenu item "SUB2" */
   MenuCheck( hMenu, MID_SUB1, MC_FLIP );  /* Uncheck SUB1 */
   MenuOn( hMenu );  /* so top-level redrawn */
   MenuKey( hMenu, KB_F1 );
   MenuEnd( hMenu );
}

TOOLBOX SelectMenu, 0, 0, 240 {
  "FIRST", MID_1ST_TOP {
    "SUB 1", MID_SUB1
    "SUB 2", MID_SUB2
  }
  "SECOND", MID_2ND_TOP {
    "SUB A", MID_SUBA
  }
}
```

PopupAddText

Declaration: HANDLE **PopupAddText** (HANDLE *hPopup*, SWORD *ParentId*, const char * *Name*, short *Id*)

Category(ies): Menus

Description: **PopupAddText** is an older version of **DynMenuAdd**. **DynMenuAdd** is more flexible and should be used in general.

Inputs:

hPopup — HANDLE returned from **PopupNew**.

ParentId —
- 0 — Top-level entry that can have children.
- -1 — Low-level menu item.
- > 0 — The given *Id* is the child of this parent.

Name — Pointer to text string to add.

Id — Menu ID for this entry (1 . . . 0xFFF).

Outputs: If successful, *hPopup*. H_NULL if out of memory or error in parameters (*ParentId* not found, *ParentId* found but it was not a possible parent or the maximum number of items in a menu was exceeded).

Note that if there is an error adding the new entry, the MF_ERROR bit in **MenuFlags** (*hPopup*) is set.

Assumptions: **DynMenuAdd** is more general purpose and should generally be used instead of **PopupAddText**.

Side Effects: May cause heap compression.

Availability: All versions of the TI-89 / TI-92 Plus.

TI-89 / TI-92 Plus Differences: None

See Also: **DynMenuAdd, DynMenuChange, PopupNew**

Example: See **PopupDo**.

PopupBegin

Declaration: HANDLE **PopupBegin** (HANDLE *PopupHandle*, SINT *Flags*)

Category(ies): Menus

Description: Allocate a menu-draw structure for a dynamic pop-up so that the pop-up items can have the enable / disable or checkmark features of menus.

Inputs:
PopupHandle — HANDLE created by **PopupNew**.
Flags — None currently used.

Outputs: Menu-draw HANDLE that is then passed to **PopupBeginDo**.

Assumptions:

> **NOTE:** There is a STRICT method of using dynamic popups that use **PopupBegin**.

1. Create an empty, dynamic pop-up structure with **PopupNew**.
2. Build the menu with **DynMenuAdd** (or **PopupAddText**).
3. Pass the handle returned by **PopupNew** to **PopupBegin**.
4. The handle returned by **PopupBegin** can now be passed to **MenuSubStat** (**PopupSubStat** is #defined to this) to enable/disable individual items or **MenuCheck** (**PopupCheck**) to turn on/off or test the status of checkmarks for individual items.
5. Pass the handle returned from **PopupBegin** to **PopupBeginDo** to actually execute the pop-up.
6. When done with the menu, call **MenuEnd** on the handle returned from **PopupBegin**. This will free that handle AND the handle returned from **PopupNew**.

Do not forget that once you call **PopupBegin** you may not unlock the handle returned from **PopupNew** or call **DynMenuAdd** or **DynMenuChange** (or **PopupAddText**).

Side Effects: May cause heap compression.

Availability: On AMS 2.00 and higher.

TI-89 / TI-92 Plus Differences: None

See Also: **PopupNew, PopupBeginDo, MenuSubStat, MenuCheck, MenuEnd**

(continued)

PopupBegin *(continued)*

Example:

```
WORD v;
HANDLE pnH, pbH;

if (!(pnH = PopupNew( "PopupBegin POPUP", 0 )))
   ER_THROW( ER_MEMORY );
DynMenuAdd( pnH, 0, "NORMAL", 1, DMF_TEXT | DMF_TOP );
DynMenuAdd( pnH, 0, "WILL BE CHECKED", 2, DMF_TEXT | DMF_TOP );
DynMenuAdd( pnH, 0, "WILL BE DISABLED", 3, DMF_TEXT | DMF_TOP );
DynMenuAdd( pnH, 0, "PARENT", 4, DMF_TEXT | DMF_TOP_SUB );
DynMenuAdd( pnH, 4, "CHILD 1 (to be checked)", 5, DMF_TEXT | DMF_CHILD);
DynMenuAdd( pnH, 4, "CHILD 2 (to be disabled)", 6, DMF_TEXT | DMF_CHILD);
DynMenuAdd( pnH, 4, "CHILD 3 (to change)", 7, DMF_TEXT | DMF_CHILD);
if (MenuFlags(pnH) & MF_ERROR) {
   HeapFree( pnH );
   ER_THROW( ER_MEMORY );
}
v = PopupDo( pnH, -1, -1, 0 );
if (!DynMenuChange(pnH, 7, "NEW CHILD 3", DMF_TEXT)) {
   HeapFree( pnH );
   ER_THROW( ER_MEMORY );
}
if (!(pbH = PopupBegin( pnH, 0 )))  {
   HeapFree( pnH );
   ER_THROW( ER_MEMORY );
}
PopupSubStat( pbH, 3, FALSE ); /* PopupSubStat is same as MenuSubStat */
PopupSubStat( pbH, 6, FALSE );
PopupCheck( pbH, 2, MC_CHECK ); /* PopupCheck is same as MenuCheck */
PopupCheck( pbH, 5, MC_FLIP );
v = PopupBeginDo( pbH, -1, -1, 0 );
MenuEnd( pbH );  /* will also free pnH */
```

PopupBeginDo

Declaration: WORD **PopupBeginDo** (HANDLE *pH*, short *x0*, short *y0*, short *Offset0*)

Category(ies): Menus

Description: Execute a dynamically allocated pop-up using the handle returned by **PopupBegin**.

Inputs:

pH	—	HANDLE returned from **PopupBegin**.
x0, *y0*	—	Screen coordinates of the upper left corner of the menu. If *x0* is equal to -1 then the pop-up is centered horizontally, if *y0* is equal to -1 then the pop-up is centered vertically.
Offset0	—	Menu ID of initially selected item (0 defaults to first item).

Outputs:

0	—	Nothing selected (ESC pressed) or not enough memory to display pop-up.
1 . . . 0xFFF	—	Menu ID of item selected.

Assumptions: Do NOT pass handles from **PopupNew**.

Side Effects: May cause heap compression.

Availability: On AMS 2.00 and higher.

TI-89 / TI-92 Plus Differences: None

See Also: **PopupBegin, MenuEnd**

Example: See **PopupBegin**.

PopupClear

Declaration: HANDLE **PopupClear** (HANDLE *pH*)

Category(ies): Menus

Description: Clear all entries of a dynamically created pop-up. Return the existing handle.

Inputs: *pH* — HANDLE created by **PopupNew**.

Outputs: The existing handle is always returned.

Assumptions: The memory allocated to the *pH* HANDLE is not released until the next call to **DynMenuAdd** (or **PopupAddText**).

Side Effects: None

Availability: All versions of the TI‑89 / TI‑92 Plus.

TI‑89 / TI‑92 Plus Differences: None

See Also: **PopupNew, DynMenuAdd, DynMenuChange**

Example:

```
/* In the VarOpen dialog box, when the user selects a new folder the "drop-down"
   (pop-up in a dialog box or menu) for the available variables must be repopulated.
   Since the dialog box code keeps this handle, a new one cannot be created. So the
   old pop-up is cleared and the new variables are added to it
*/
FolderName = (BYTE *) PopupText(hOpenFolder, VarOptList[1]);
PopupClear( hOpenVar );  /* empty pop-up, keep same handle */
AddSymsToOpenPopup( hOpenVar, StrToTokN( FolderName, TokenizedName ) );
return DB_REDRAW;  /* changed drop-down, must redraw */
```

PopupDo

Declaration: WORD **PopupDo** (HANDLE *pH*, short *x0*, short *y0*, short *Offset*)

Category(ies): Menus

Description: Execute a dynamic POPUP created by **PopupNew**.

Inputs:

pH	—	HANDLE returned from **PopupNew**.
x0, *y0*	—	Screen coordinates of the upper left corner of the menu. If *x0* is equal to -1 then the pop-up is centered horizontally, if *y0* is equal to -1 then the pop-up is centered vertically.
Offset	—	Menu ID of initially selected item (0 defaults to first item).

Outputs:

0	—	Nothing selected (ESC pressed) or not enough memory to display pop-up.
1 . . . 0xFFF	—	Menu ID of item selected.

Assumptions: Do NOT pass handles from **PopupBegin**!

Side Effects: May cause heap compression.

Availability: All versions of the TI-89 / TI-92 Plus.

TI-89 / TI-92 Plus Differences: None

See Also: **PopupNew, MenuPopup (static POPUPs)**

Example:

```
volatile HANDLE h = NULL;
volatile WORD opt = 0;

TRY
   if (!(h = PopupNew( "TITLE", 0 )))
      ER_throw(ER_MEMORY);
   if (!PopupAddText( h, 0, "QUIT", 1 ) || !PopupAddText( h, 0, "SELECT TO CHANGE", 2 ))
      ER_throw(ER_MEMORY);
   if (2 == PopupDo( h, -1, -1, 0 )) {
      if (!PopupChangeText( h, 2, "THIS WAS CHANGED" ))
         ER_throw(ER_MEMORY);
      opt = PopupDo( h, -1, -1, 2 );
}
PopupFree( h );
ONERR
   if (h)
      PopupFree( h );
ENDTRY
return opt;
```

PopupNew

Declaration: HANDLE **PopupNew** (const char * *Title*, short *MaxHeight*)

Category(ies): Menus

Description: Begin a dynamically created pop-up returning a handle to an empty pop-up structure or H_NULL if not enough memory. This empty pop-up structure can be modified with **DynMenuAdd** or **DynMenuChange**. The pop-up can then be executed with **PopupDo** or if menu features (checkmarks, enabling or disabling entries) are needed then **PopupBegin**/**PopupBeginDo** can be used.

Inputs:

Title — Pointer to string for title of pop-up or NULL if no title.

MaxHeight — Set to 0 for default height (as high as will fit onto the screen). If > 0 then use it as the maximum height of the pop-up.

Outputs: HANDLE to empty POPUP structure or H_NULL if not enough memory.

Assumptions: None

Side Effects: May cause heap compression.

Availability: All versions of the TI‑89 / TI‑92 Plus.

TI‑89 / TI‑92 Plus Differences: None

See Also: **PopupDo, PopupBegin, PopupClear, DynMenuAdd, DynMenuChange**

Example: See **PopupDo** (normal pop-up) or **PopupBegin** (pop-up with menu like features — enable/disable and checkmarks) for an example.

PopupText

Declaration: char * **PopupText** (HANDLE *pH*, SINT *PopupId*)

Category(ies): Menus

Description: Return a pointer to the text of a dynamically created pop-up.

Inputs: *pH* — HANDLE to a pop-up from **PopupNew**.

PopupId — Pop-up ID to search for (1 . . . 0xFFF).

Outputs: Pointer to text (or icon or bitmap).

Assumptions: Only works for dynamically created popups. For menus, use **MenuItemDef**.

Side Effects: None

Availability: All versions of the TI‑89 / TI‑92 Plus.

TI‑89 / TI‑92 Plus Differences: None

See Also: **PopupNew, PopupDo, PopupAddText**

Example:

```
/* Assume hFolderPopup is the handle of a dynamically created pop-up,
   buf is a character buffer, opt is a SINT.
*/
if (opt = PopupDo(hFolderPopup,0,0,0)) {
   sprintf( buf, "You selected %s", PopupText( hFolderPopup, opt ));
   Disp( buf );
}
```

QMenuTopSelect

Declaration: WORD **QMenuTopSelect** (HANDLE *MenuHandle*)

Category(ies): Menus

Description: Return the currently selected top-level menu item (only one may be selected at a time) or ‑1 if none is selected.

Inputs: *MenuHandle* — Handle returned from **MenuBegin**.

Outputs: Top-level menu index (0 . . . number of top-level items – 1).

Assumptions: *MenuHandle* refers to a menu-draw handle returned from **MenuBegin**. **MenuTopSelect** is normally used with redefinable menus.

Side Effects: None

Availability: On AMS 2.00 and higher.

TI‑89 / TI‑92 Plus Differences: None

See Also: **MenuBegin, MenuTopSelect**

Example: See **MenuTopRedef** for an example of using **MenuTopSelect**. Note that **QMenuTopSelect** just returns the value set by **MenuTopSelect**.

Appendix A: System Routines — Mode Screen Settings

MO_currentOptions

Declaration: void **MO_currentOptions** (void)

Category(ies): Mode Screen Settings

Description: Loads current mode settings into global array **MO_option**. See **MO_option** for details of what values you can find in this array.

Mode settings from various OS subsystems are collected and placed into **MO_option** for easy reference.

Inputs: None

Outputs: None

Assumptions: **MO_option** is a system global array, but it only reflects current mode settings immediately after **MO_currentOptions** has been called.

Side Effects: None

Availability: All versions of the TI-89 / TI-92 Plus.

TI-89 / TI-92 Plus Differences: None

See Also: **MO_digestOptions**, **MO_option**

Example:

```
Access_AMS_Global_Variables;
WORD angleMode;
   .
   .
   .
MO_currentOptions();
angleMode = MO_option[MO_OPT_ANGLE]; /* Get current angle mode setting */
```

MO_digestOptions

Declaration: void **MO_digestOptions** (void)

Category(ies): Mode Screen Settings

Description: Changes mode settings to reflect values stored in global array **MO_option**.

This is how you change system mode settings from your app. First call **MO_currentOptions** to load the current mode settings into **MO_option**. Then, change mode settings you are concerned with in array **MO_option**. Finally, call **MO_digestOptions** to process your changes.

See **MO_option** for details of what values you can store in this array.

Inputs: defaultFolder — This should always be zero (0) when called by an app. The MODE dialog box calls this routine with a nonzero value to indicate which folder name is highlighted in the "Current Folder" pop-up menu.

Outputs: None

Assumptions: None

Side Effects: This routine sends a CM_MODE_CHANGE message to each app, which may in turn cause heap compression, window repaints, and other side effects known only to the apps.

Availability: All versions of the TI‑89 / TI‑92 Plus.

TI‑89 / TI‑92 Plus Differences: None

See Also: **MO_currentOptions**, **MO_option**

Example:

```
Access_AMS_Global_Variables;              /* so we can access MO_option */
MO_currentOptions();                      /* get current mode settings */
MO_option[MO_OPT_ANGLE] = D_ANGLE_RAD;    /* Change to radians mode */
MO_digestOptions(0);
```

Appendix A: System Routines — Operating System

See Also:

EV_captureEvents

Declaration: EV_entryPoint **EV_captureEvents** (EV_entryPoint *callback*)

Category(ies): Operating System

Description: Redirects all events to *callback*. Use this routine to capture events if you are creating your own dialog or menu managers.

Save the returned value and use it to restore event processing when you are finished capturing events.

Inputs: *callback* — The address of a routine which accepts one parameter, an event pointer.

Outputs: Returns pointer to previously captured event handler.

Assumptions: None

Side Effects: None

Availability: All versions of the TI‑89 / TI‑92 Plus.

TI‑89 / TI‑92 Plus Differences: None

See Also: Not applicable.

Example: **MyDialog** opens a window and captures events. All subsequent events are sent to **myEventHandler**. Routine **myEventHandler** calls cancel when it is finished handling events. Routine **cancel** closes the dialog window and restores normal event dispatching.

```
WINDOW myWindow;
EV_eventPoint oldEventHandler;
EV_FLAGS oldPaintState;

void MyDialog(void)
/* Initialize my display */
{
   static const WIN_RECT r = {
      (W_MAX_X - WINDOW_WIDTH*LF_WIDTH(' '))/2,
      WINDOW_TOP,
      (W_MAX_X + WINDOW_WIDTH*LF_WIDTH(' '))/2 + 1,
      WINDOW_TOP+W_TITLE_Y+WINDOW_HEIGHT*LF_HEIGHT + 1
   };
   oldPaintState = EV_suspendPainting();
```

(continued)

EV_captureEvents *(continued)*

```
   /* Display my window */
   if (! WinOpen(&myWindow, &r, WF_SAVE_SCR|WF_ROUNDEDBORDER|WF_TITLE|WF_TTY,
         "MY WINDOW"))
   {
      EV_restorePainting(oldPaintState);
      return;
   }

   WinActivate(&myWindow);
   WinBeginPaint(&myWindow);
   WinAttr(&myWindow, A_REPLACE);
   myWindow.Flags &= ~WF_DIRTY;          /* I will repaint this window myself */
   myWindow.TaskId = AP_NONE;            /* no task owns this window */
   paint();

   /* Install event handler */
   oldEventHandler = EV_captureEvents(myEventHandler);
}

static void myEventHandler(PEvent e)
{
   if (e->command == CM_KEY_PRESS)
   {
      switch (e->info.keyInfo.keyCode)
      {
         case KB_ENTER:
         case KB_ESC:
            cancel();                  /* cancel dialog */
            break;

         case KB_SWITCH:
         case KB_VARLINK:
         case KB_CHAR:
         case KB_MATH:
         case KB_MODE:
         case KB_MENU:
         case KB_MEM:
         case KB_QUIT:
         case KB_HOME:
         case KB_YEQ:
         case KB_RANGE:
         case KB_GRAPH:
         case KB_TBLSET:
         case KB_TABLE:
         case KB_OFF:
            cancel();                  /* cancel dialog */
            EV_defaultHandler(e);      /* allow default handling */
            break;
```

(continued)

EV_captureEvents *(continued)*

```
         case KB_ON+KB_OPTION:
            EV_defaultHandler(e);      /* allow default handling */
            break;
      }
   }
}

static void cancel(void)
{
   WinEndPaint(&myWindow);
   WinClose(&myWindow);                /* close my window */
   EV_captureEvents(oldEventHandler);  /* re-establish previous event handler */
   EV_restorePainting(oldPaintState);
}
```

EV_defaultHandler

Declaration: void **EV_defaultHandler** (Event * *event*)

Category(ies): Operating System

Description: Defines the default behavior of all events.

Your application interacts with the user by responding to events, primarily CM_KEY_PRESS and CM_WPAINT events. But there are many more event types (see section **9.3. Commands**) the majority of which your application need not concern itself. Call **EV_defaultHandler** to deal with any event messages your application does not explicitly handle.

Inputs: *event* — Pointer to event message from the OS.

Outputs: None

Assumptions: Most events are either acted upon by your application or passed to **EV_defaultHandler** for default behavior. Some events may need both app processing and default processing. The CM_ACTIVATE event usually requires such handling. When your application receives a CM_ACTIVATE event, it usually calls **WinActivate** to activate its window, then calls **EV_defaultHandler** to get automatic menu activation. See section **9.4. Starting and Stopping an Application** for details.

Side Effects: There can be any number of side effects depending on which event was forwarded to **EV_defaultHandler**. See section **9.9. Default Event Handler** for default actions taken by each event type.

Availability: All versions of the TI-89 / TI-92 Plus.

TI-89 / TI-92 Plus Differences: None

See Also: Not applicable.

(continued)

EV_defaultHandler *(continued)*

Example:

```
AP_myApp(Event *event)
{
   switch (event->command)
   {
   case CM_START:
      . /* start application */
      .
      .
      break;

   case CM_KEY_PRESS:
      . /* a key was pressed */
      .
      .
      break;

   case CM_WPAINT:
      . /* repaint window */
      .
      .
      break;

   default:
      EV_defaultHandler(event);
      break;
   }
}
```

EV_getc

Declaration: USHORT **EV_getc** (UINT *indic*, Event * *event*)

Category(ies): Operating System

Description: Gets next character from keyboard. Puts the calculator into low power mode until a character is available.

Inputs: *indic* — ST_IDLE or ST_PAUSE for the kind of busy indicator to display on the status line while waiting for a keypress.

Outputs: *event* — Return event message containing either a CM_KEY_PRESS or CM_CURSOR_FLASH command. The event contains the keyboard modifier keys when the key was pressed.

Returns either a character number or zero (0) if the cursor blink timer timed out.

Assumptions: This routine will automatically power down the calculator if a key has not been pressed after a few minutes.

Side Effects: None

Availability: All versions of the TI‑89 / TI‑92 Plus.

TI‑89 / TI‑92 Plus Differences: None

See Also: Not applicable.

Example:

```
Event e;
USHORT ch;

/* Get a keypress from the keyboard */
while ((ch = EV_getc(ST_PAUSE, &e)) == 0)
   ;
```

EV_restorePainting

Declaration: EV_FLAGS **EV_restorePainting** (EV_FLAGS *newState*)

Category(ies): Operating System

Description: Restores the paint message state. Use this routine to turn paint messages back on after turning them off with **EV_suspendPainting**.

Inputs: *newState* — State returned from previous call to **EV_suspendPainting**.

Outputs: Returns the state of paint messages before **EV_restorePainting** was called.

Assumptions: None

Side Effects: None

Availability: All versions of the TI‑89 / TI‑92 Plus.

TI‑89 / TI‑92 Plus Differences:

See Also: **EV_suspendPainting**

Example: See **EV_captureEvents**.

EV_sendEvent

Declaration: void **EV_sendEvent** (AppID *destApp*, Event * *event*)

Category(ies): Operating System

Description: Sends an event message to an application. The destination app receives and processes the event in its event handling entry point.

Inputs:

destApp — ID of the destination application. Use **EV_getAppID** to find an app's ID given its internal name.

event — Pointer to event message.

Outputs: None

Assumptions: The destination app does not have to be started or active to receive messages. You should first start the app (**EV_startApp**) if it is important that the app be on the screen before it receives *event*.

Applications may communicate with each other by sending interapplication messages, but this means of interfacing between apps is not recommended. Interapplication messaging is largely replaced with an object-oriented approach. Applications should communicate with each other through their frame interface.

Apps can use this routine to send or forward OS events to other apps.

Side Effects: None

Availability: All versions of the TI‑89 / TI‑92 Plus.

TI‑89 / TI‑92 Plus Differences: None

See Also: Not applicable

Example:

```
/* Send keypress to another application */
void sendChar(AppID destApp, USHORT ch)
{
   Event e;
   e.command = CM_KEY_PRESS;                /* send keypress command */
   e.info.keyInfo.keyCode = ch;             /* stuff character to send */
   EV_sendEvent(destApp, &e);               /* send event to dest app */
}
```

EV_setCmdCheck

Declaration: void **EV_setCmdCheck** (UINT *cmd*, enum CheckCmds *checkCmd*)

Category(ies): Operating System

Description: Sets or clears checkmark on a menu item.

Inputs:

cmd — Menu item ID — the command number given to the item when the menu resource was created.

checkCmd —
- MC_CHECK — Add checkmark.
- MC_UNCHECK — Remove checkmark.
- MC_FLIP — Toggle checkmark.

Outputs: None

Assumptions: This routine uses the menu handle in application frame attribute OO_APP_DEFAULT_MENU_HANDLE. See section **9.6. Menu Processing** for guidelines on setting up a menu where default event handling can find it.

Side Effects: None

Availability: All versions of the TI-89 / TI-92 Plus.

TI-89 / TI-92 Plus Differences: None

See Also: **EV_setCmdState, EV_setFKeyState**

Example:

```
EV_setCmdCheck(currentFontSize, MC_UNCHECK);
EV_setCmdCheck(newFontSize, MC_CHECK);
currentFontSize = newFontSize;
```

EV_setCmdState

Declaration: void **EV_setCmdState** (UINT *cmd*, BOOL *state*)

Category(ies): Operating System

Description: Enables or disables a menu item. A disabled menu item is dimmed and cannot be selected by the user.

Inputs:
cmd — menu item ID — the command number given to the item when the menu resource was created.

state — TRUE — Enable command.

FALSE — Disable command (dim menu item).

Outputs: None

Assumptions: This routine uses the menu handle in application frame attribute OO_APP_DEFAULT_MENU_HANDLE. See section **9.6. Menu Processing** for guidelines on setting up a menu where default event handling can find it.

Side Effects: None

Availability: All versions of the TI‑89 / TI‑92 Plus.

TI‑89 / TI‑92 Plus Differences: None

See Also: **EV_setCmdCheck, EV_setFKeyState,** section **9.6. Menu Processing**

Example:

```
EV_setCmdState(CM_SAVE_AS, FALSE);          /* disable Save As . . . command */
```

EV_setFKeyState

Declaration: void **EV_setFKeyState** (USHORT *fkey*, BOOL *state*, BOOL *redraw*)

Category(ies): Operating System

Description: Enables or disables a menu function key. A disabled function key is dimmed in the menu bar and cannot be selected by the user.

Avoid menu redraw flicker when enabling/disabling several function keys by specifying

Inputs:

fkey — Function key number 1 . . . 8.

state — TRUE — Enable function key.

FALSE — Disable function key (dim menu title).

redraw — TRUE — Redraw menu bar.

FALSE — Do not redraw menu bar.

Avoid menu redraw flicker when enabling/disabling several function keys by specifying FALSE for *redraw* in all but the last call to this routine.

Outputs: None

Assumptions: This routine uses the menu handle in application frame attribute OO_APP_DEFAULT_MENU_HANDLE. See section **9.6. Menu Processing** for guidelines on setting up a menu where default event handling can find it.

Side Effects: None

Availability: All versions of the TI-89 / TI-92 Plus.

TI-89 / TI-92 Plus Differences: None

See Also: **EV_setCmdCheck, EV_setCmdState,** section **9.6. Menu Processing**

Example:

```
USHORT f;
for (f = 1; f <= 8; f += 1)                 /* disable all menus */
   EV_setFKeyState(f, FALSE, FALSE);        /* do not redraw the menu bar, yet */
EV_setFKeyState(1, TRUE, TRUE);             /* enable F1 and redraw the menu bar */
```

EV_startApp

Declaration: void **EV_startApp** (AppID *app*, USHORT *startCode*)

Category(ies): Operating System

Description: Starts another application. Activates *app* if it is already running.

Inputs:

app — ID of app to start.

startCode —

AP_START_CURRENT	—	Use current data from last time app ran.
AP_START_OPEN	—	Prompt user to open existing variable.
AP_START_NEW	—	Prompt user for new variable name.
AP_START_ERROR	—	Display data where error occurred.

These start codes correspond to Current, Open, and New seen on some apps in the [APPS] menu.

Outputs: None

Assumptions: Under normal circumstances, you should start another app with AP_START_CURRENT.

Side Effects: Your application will be deactivated and may be terminated to start the other app.

Availability: All versions of the TI-89 / TI-92 Plus.

TI-89 / TI-92 Plus Differences: None

See Also: **EV_quit**

Example:

```
EV_startApp(otherApp, AP_START_CURRENT);
```

EV_suspendPainting

Declaration: EV_FLAGS **EV_suspendPainting** (void)

Category(ies): Operating System

Description: Tells the OS to quit sending CM_WPAINT messages when it finds dirty windows. This is useful when you are creating your own dialog or menu managers. This prevents paint messages from being sent to background windows while you are processing events.

Inputs: None

Outputs: Returns the state of paint messages before **EV_suspendPainting** was called.

Assumptions: Save the returned state of paint messages from this call, and use the saved state to restore painting later with a call to **EV_restorePainting**.

Side Effects: None

Availability: All versions of the TI-89 / TI-92 Plus.

TI-89 / TI-92 Plus Differences: None

See Also: **EV_restorePainting**

Example: See **EV_captureEvents**.

EV_switch

Declaration: void **EV_switch** (void)

Category(ies): Operating System

Description: Switches screen side. Activates the application on the other side of a split screen, or restarts the application which last ran before the current app.

Inputs: None

Outputs: None

Assumptions: None

Side Effects: Your application will be deactivated and may be terminated to start the other app.

Availability: All versions of the TI-89 / TI-92 Plus.

Versions prior to AMS 2.00 only switch between split-screen apps. The ability to switch to the previous app was introduced in AMS 2.00.

TI-89 / TI-92 Plus Differences: None

See Also; Not applicable.

Example:

```
EV_switch();
```

EX_getBasecodeParmBlock

Declaration: BASECODE_PARM_BLOCK const * **EX_getBasecodeParmBlock** (void)

Category(ies): Operating System

Description: Get a pointer to the base code parameter block.

Inputs: None

Outputs: The base code parameter block contains version information about the AMS Operating System: the major and minor version number of the AMS Operating System and date the OS was built.

```
typedef struct
{
   unsigned short len;  /* length of parameter block */
   unsigned short version_number;  /* 1-byte major, 1-byte minor */
   unsigned long version_date;  /* 2-byte yr, 1-byte mo, 1-byte day *
} BASECODE_PARM_BLOCK;
```

Assumptions: None

Side Effects: None

Availability: On AMS 2.04 and higher.

TI-89 / TI-92 Plus Differences: None

See Also: Not applicable.

Example:

```
BASECODE_PARM_BLOCK const *bpb = EX_getBasecodeParmBlock();
```

FL_getHardwareParmBlock

Declaration: HARDWARE_PARM_BLOCK * **FL_getHardwareParmBlock** (void)

Category(ies): Operating System

Description: Get a pointer to the hardware parameter block.

Inputs: None

Outputs: The hardware parameter block contains a description of the calculator hardware.

```
typedef struct {
   unsigned short len;                // length of parameter block
   unsigned long hardwareID;          // 1 = TI-92 Plus, 3 = TI-89
   unsigned long hardwareRevision;    // hardware revision number
   unsigned long bootMajor;           // boot code version number
   unsigned long bootRevision;        // boot code revision number
   unsigned long bootBuild;           // boot code build number
   unsigned long gateArray;           // gate array version number
   unsigned long physDisplayBitsWide; // display width
   unsigned long physDisplayBitsTall; // display height
   unsigned long LCDBitsWide;         // visible display width
   unsigned long LCDBitsTall;         // visible display height
} HARDWARE_PARM_BLOCK;
```

The TI‑89 / TI‑92 Plus allocate the same amount of memory for the LCD. However, the TI‑89 cannot display as much as the TI‑92 Plus. *LCDBitsWide* and *LCDBitsTall* reflect how much of LCD memory the calculator user can see.

> **Note:** Some fields of the HARDWARE_PARM_BLOCK are not available in earlier versions of the calculator. The fields up through *bootBuild* are available in all boot code versions. It is important to check the value of *len* before accessing any values after *bootBuild*. See OFFSETOF in the example below to learn how to determine if a particular field is present.

Assumptions: You cannot change any values in the parameter block. The pointer returned by this routine points directly to a parameter block in ROM.

Side Effects: None

Availability: All versions of the TI‑89 / TI‑92 Plus.

TI‑89 / TI‑92 Plus Differences: None

See Also: Not applicable.

(continued)

FL_getHardwareParmBlock *(continued)*

Example:

```
HARDWARE_PARM_BLOCK *hpb = FL_getHardwareParmBlock();
   .
   .
   .
/* Is LCDBitsTall field available in Hardware Parameter Block? */
if (hpb->len >= OFFSETOF(HARDWARE_PARM_BLOCK, LCDBitsTall))
{
   /* Yes, use hpb values */
   LCDHeight = hpb->LCDBitsTall;
   LCDWidth = hpb->LCDBitsWide;
}
else /* No, assume the smallest display */
{
   LCDHeight = 100;
   LCDWidth = 160;
}
```

handleRclKey

Declaration: void **handleRclKey** (BOOL *ReplaceCRs*)

Category(ies): Operating System, Keyboard, Variables

Description: Manages the dialog box which appears when 2nd [RCL] is pressed.

A pop-up dialog box is displayed requesting the name of a variable. The contents of the variable are converted to text and pasted to the current application.

Inputs: *ReplaceCRs* — If TRUE, carriage returns are converted to colons after the contents of the variable have been converted to text. The variable itself is not changed.

If FALSE, carriage returns remain unchanged.

Outputs: None

Assumptions: The current application receives a CM_PASTE_HANDLE command. It is the responsibility of the application to make sure the handle contents are pasted to a text field or to call the default event handler for this message if the app has no active text field.

Nothing is pasted if the user pressed ESC to cancel the 2nd [RCL] dialog.

Side Effects: May cause heap compression. May throw errors.

Availability: All versions of the TI-89 / TI-92 Plus.

TI-89 / TI-92 Plus Differences: None

See Also: Not applicable.

Example:

```
case CM_RCL:                            /* User pressed [RCL] */
   if (! (te->flags & TE_READ_ONLY))    /* Is my text field writeable? */
      handleRclKey(FALSE);              /* Yes, allow variable recall */
   break;                               /* No, ignore [RCL] key */
```

handleVarLinkKey

Declaration: void **handleVarLinkKey** (WORD *defType*)

Category(ies): Operating System, Keyboard, Variables, Link

Description: Activate VAR-LINK.

Inputs: *defType* — Value to specify default type to display: SDT_ASM, SDT_DATA, SDT_EXPR, SDT_FIG, SDT_FUNC, SDT_GDB, SDT_LIST, SDT_MAC, SDT_MAT, SDT_MAT, SDT_OTH, SDT_PIC, SDT_PRGM, SDT_STR, SDT_TEXT. Use SDT_ALL if all variables of all types are to be displayed.

May be OR'd with XF_VARLINK_SELECT_ONLY to not allow any variables to be changed (deleted, copied, renamed, . . .).

Outputs: The name of any variable or folder selected is sent back to the currently active app as a CM_PASTE_HANDLE message.

Assumptions: None

Side Effects: May cause heap compression.

Availability: On AMS 2.00 and higher.

TI-89 / TI-92 Plus Differences: None

See Also: **SmapTypeStrings**

Example: See **SymAdd**.

LOC_formatDate

Declaration: void **LOC_formatDate** (char const * *format*, int *y*, int *m*, int *d*, char *date*[])

Category(ies): Operating System, Apps

Description: Formats date according to *format* string.

Inputs: *format* — String containing date specifiers:

- D — One- or two-digit day of month.
- DD — Two-digit day of month (leading zero if necessary).
- M — One- or two-digit month.
- MM — Two-digit month (leading zero if necessary).
- YY — Two-digit year (year without century).
- YYYY — Four-digit year.

Any other characters are copied to output.

y — Year.

m — Month.

d — Day of month.

Outputs: *date* — String containing formatted date. Caller must supply buffer long enough to contain formatted date and terminating zero byte.

Assumptions: None

Side Effects: None

Availability: On AMS 2.02 and higher.

TI-89 / TI-92 Plus Differences: None

See Also: **LOC_getLocalDateFormat, LOC_localVersionDate**

Example:

```
char formattedDate[16];
int y = 2000, m = 6, d = 9;
LOC_formatDate("M/D/YYYY", y, m, d, formattedDate);   /* 6/9/2000        */
LOC_formatDate("YYYY.MM.DD", y, m, d, formattedDate); /* 2000.06.09      */
LOC_formatDate("D-M-YY", y, m, d, formattedDate);     /* 9-6-00          */
LOC_formatDate("MM/YYYY", y, m, d, formattedDate);    /* 06/2000         */
```

LOC_getLocalDateFormat

Declaration: char const * **LOC_getLocalDateFormat** (void)

Category(ies): Operating System, Apps

Description: Gets a pointer to the date format string specified by the current language mode setting.

Inputs: None

Outputs: Returns a pointer to a date format string. See **LOC_formatDate** for a description of date format strings.

Assumptions: None

Side Effects: None

Availability: On AMS 2.02 and higher.

TI-89 / TI-92 Plus Differences: None

See Also: **LOC_formatDate, LOC_localVersionDate**

Example:

```
char formattedDate[16];
int y = 2000, m = 6, d = 9;
/* Format date according to local language */
LOC_formatDate(LOC_getLocalDateFormat(), y, m, d, formattedDate);
```

LOC_localVersionDate

Declaration: char * **LOC_localVersionDate** (char *datebuf*[])

Category(ies): Operating System, Apps

Description: Formats release date of AMS Operating System according to current language setting. The Home screen About dialog calls this routine to display the release date of the built-in calculator software.

Inputs: *datebuf* — Buffer long enough to contain formatted date and terminating zero byte.

Outputs: Fills *datebuf* with OS release date. Returns pointer to *datebuf*.

Assumptions: None

Side Effects: None

Availability: On AMS 2.02 and higher.

TI-89 / TI-92 Plus Differences: None

See Also: **LOC_formatDate, LOC_getLocalDateFormat**

Example:

```
char formattedDate[16];
/* Format OS release date according to local language */
LOC_localVersionDate(formattedDate);
```

Appendix A: System Routines — Program I/O Screen

cmd_clrio

Declaration: void **cmd_clrio** (void)

Category(ies): Program I/O Screen

Description: Clear program I/O window and set program I/O window cursor position to upper left corner.

This routine implements the TI-BASIC ClrIO command.

Inputs: None

Outputs: None

Assumptions: None

Side Effects: None

Availability: On AMS 1.05 and higher.

TI-89 / TI-92 Plus Differences: None

See Also: **cmd_clrhome**

Example:

```
cmd_clrio();
```

cmd_disp

Declaration: void **cmd_disp** (EStackIndex *e*)

Category(ies): Program I/O Screen

Description: Display estack expressions in the program I/O window. Each expression on the estack is displayed in turn until an END_TAG is encountered. The pretty print mode setting determines how the expressions are formatted for display.

This routine implements the TI-BASIC Disp command.

Inputs: *e* — estack index of first expression to display.

Outputs: None

Assumptions: None

Side Effects: May cause heap compression.

Availability: On AMS 1.05 and higher.

TI-89 / TI-92 Plus Differences: None

See Also: Not applicable.

Example:

```
/* This example displays 3.14 and 7 on the program I/O screen */
Access_AMS_Global_Variables;
EStackIndex savetop = top_estack;           /* save index to top of estack */
push_quantum(END_TAG);                      /* mark end of list */
push_long_to_integer(7);                    /* push 7 on estack */
push_Float(3.14);                           /* push 3.14 on estack */
cmd_disp(top_estack);                       /* display 3.14 and 7 */
top_estack = savetop;                       /* pop estack */
```

Appendix A: System Routines — Solver

push_csolve

Declaration: void **push_csolve** (EStackIndex *k*, EStackIndex *vi*)

Category(ies): Solver

Description: Pushes onto the estack a Boolean expression that describes the complex solution set of the Boolean expression indexed by *k*, for the variable(s) indexed by *vi*.

Uses the principal branch of any fractional powers.

If invoked via **push_internal_simplify**, *vi* and *k* are simplified to deepest variable(s).

Inputs:

k — Index of the top tag of a Boolean expression that is usually one or more equations and/or inequalities joined by “and” operators.

vi — Index of the top tag of a variable, an equation of the form variable = constant representing an initial guess, or a list of variables and/or such initial guesses.

Outputs: None

Assumptions: None

Side Effects: None

Availability: On AMS 2.02 and higher.

TI-89 / TI-92 Plus Differences: None

See Also: **push_solve, push_czeros, push_zeros, push_nSolve**

Example:

```
push_negate_quantum_as_negint(1u);
right_side = top_estack;
push_quantum_as_nonnegative_int (2u);
exponent = top_estack;
push_quantum (8u);   /* Push variable x */
replace_top2_with_pow (exponent);   /* top_estack -> x^2 */
push_equals (top_estack, right_side);
equation = top_estack;   /* equation -> x^2 = -1 */
push_quantum (8u);   /* push variable x */
push_csolve (equation, top_estack);    /* top_estack -> x=i or x=-i */
```

push_czeros

Declaration: void **push_czeros** (EStackIndex *k*, EStackIndex *vi*)

Category(ies): Solver

Description: If *vi* is one variable or guess, pushes onto the estack a list of zero or more real or unreal values that make the expression(s) indexed by *k* equal to zero when substituted for the variable.

Otherwise pushes an empty list if the function could not find any simultaneous zeros of the list of expressions indexed by *k*.

Otherwise pushes a matrix of real and/or unreal values, with each column representing the corresponding element variable of *vi*, and each row representing an alternate list of corresponding values that make all of the expressions in the list indexed by *k* equal to zero.

All fractional powers are interpreted using the principal branch.

Issues XR_USE_CSOLVE_WARN when there are additional zeros that cannot be expressed in this form.

If invoked via **push_internal_simplify**, *vi* and *k* are simplified to deepest variable(s).

Inputs:

k — Index of the top tag of an algebraic expression or a list thereof.

vi — Indexes of the top tag of a variable, an equation of the form variable = constant representing an initial guess, or a list of variables and/or such initial guesses.

Outputs: None

Assumptions: None

Side Effects: None

Availability: On AMS 2.02 and higher.

TI-89 / TI-92 Plus Differences: None

See Also: **push_solve, push_csolve, push_zeros, push_nSolve**

Example:

```
push_quantum_as_nonnegative_int (2u);
exponent = top_estack;
push_quantum (8u);   /* Push variable x */
replace_top2_with_pow (exponent);   /* top_estack -> x^2 */
add1_to_top ();
expression = top_estack;   /* top_estack -> x^2 + 1 */
push_quantum (8u);   /* push variable x */
push_czeros (expression, top_estack);   /* top_estack -> {i, -i} */
```

push_nSolve

Declaration: void **push_nSolve** (EStackIndex *i*, EStackIndex *ki*)

Category(ies): Solver

Description: If invoked via **push_internal_simplify**, *ki* and *i* are simplified to deepest variable, and the simplification of *i* is done under the temporary influence of SET_PARTIAL_SIMPLIFY to avoid costly polynomial expansions, polynomial GCDs, etc.

Pushes the input if these conditions are not met.

Otherwise pushes onto the estack TRUE_TAG, FALSE_TAG or an equation specifying one approximate solution of the form *ki* = float, found by iterative methods.

Takes into account bounds specified by **NG_such_that_index**.

Inputs:

i — Indexes the top tag of an internally-simplified equation depending only on the variable in *ki*.

ki — Indexes the top tag of a variable or an equation of the form variable = constant, representing an initial guess.

Outputs: None

Assumptions: None

Side Effects: None

Availability: On AMS 2.02 and higher.

TI-89 / TI-92 Plus Differences: None

See Also: **push_solve, push_csolve, push_zeros, push_czeros**

Example:

```
push_Float (4.0);
right_side = top_estack;
push_quantum_as_nonnegative_int (2u);
exponent = top_estack;
push_quantum (8u);   /* Push variable x */
replace_top2_with_pow (exponent);   /* top_estack = x^2 */
push_equals (top_estack, right_side);
equation = top_estack;   /* x^2 = 4 */
push_quantum (8u);   /* Push variable x */
push_nSolve (equation, top_estack);   /* Pushes 2.0 */
```

push_solve

Declaration: void **push_solve** (EStackIndex *i*, EStackIndex *vi*)

Category(ies): Solver

Description: If invoked via **push_internal_simplify**, *ki* and *i* are simplified to deepest variable(s). Pushes onto the estack another Boolean expression that describes the real solution set, using the real-branch of any fractional powers.

Inputs:

i — Indexes the top tag of an internally-simplified Boolean expression that is usually one or more equations and/or inequalities joined by “and” operators.

vi — Indexes a variable, an equation of the form variable = constant representing an initial guess, or a list of variables and/or such initial guesses.

Outputs: None

Assumptions: None

Side Effects: None

Availability: On AMS 2.02 and higher.

TI-89 / TI-92 Plus Differences: None

See Also: **push_csolve, push_nSolve, push_zeros, push_czeros**

Example:

```
push_quantum_as_nonnegative_int(4u);
right_side = top_estack;
push_quantum_as_nonnegative_int(2u);
exponent = top_estack;
push_quantum (8u);   /* Push variable x */
replace_top2_with_pow (exponent);   /* top_estack -> x^2 */
push_equals (top_estack, right_side);
equation = top_estack;   /* equation -> x^2 = 4 */
push_quantum (8u);   /* Push variable x */
push_solve (equation, top_estack);   /* Pushes x = -2 or x = 2 */
```

push_zeros

Declaration: void **push_zeros** (EStackIndex *i*, EStackIndex *vi*)

Category(ies): Solver

Description: If invoked via **push_internal_simplify**, *vi* and *i* are simplified to deepest variable(s).

If *vi* is not a list, pushes onto the estack a list of zero or more real values that make *i* or all of the expressions in list *i* zero when substituted for the variable in *vi*.

Otherwise pushes an empty list if the function could not find simultaneous real values of the variables in *vi* that make the expressions in *i* simultaneously zero.

Otherwise pushes a matrix of real values with each column representing the corresponding element variable of *vi*, and each row representing an alternate list of corresponding real values that make *i* or all of the expressions in list *i* zero.

All fractional powers are interpreted using the real branch.

Issues XR_USE_SOLVE_WARN when there are additional zeros that cannot be expressed in this form.

Inputs:

i — Indexes the top tag of an algebraic expression or a list thereof.

vi — Indexes a variable, an equation of the form variable = constant representing an initial guess, or a list of variables and/or such initial guesses.

Outputs: None

Assumptions: None

Side Effects: None

Availability: On AMS 2.02 and higher.

TI-89 / TI-92 Plus Differences: None

See Also: **push_czeros, push_solve, push_csolve, push_nSolve**

Example:

```
push_quantum_as_nonnegative_int (2u);
exponent = top_estack;
push_quantum (8u);   /* Push variable x */
replace_top2_with_pow (exponent);   /* top_estack -> x^2 */
subtract1_from_top ()  i = top_estack;   /* x^2 - 1 */
push_quantum (8u);   /* Push variable x */
push_zeros(i, top_estack);   /* Pushes {-1, 1} */
```

Appendix A: System Routines — Statistics

See Also:

cmd_showstat

Declaration: void **cmd_showstat** (void)

Category(ies): Statistics, Variables

Description: Display all of the valid stat variables and/or equations in a dialog box (this is the TI-BASIC ShowStat command).

Inputs: None

Outputs: None

Assumptions: **RM_Type** must have been set to a correct value.

Side Effects: May cause heap compression.

Availability: On AMS 1.05 and higher.
Note that **RM_Type** is only available on 2.04 and higher.

TI-89 / TI-92 Plus Differences: None

See Also: **statStart, RM_Type**

Example: See **statStart** for an example that creates and displays statistic variables.

push_randnorm

Declaration: void **push_randnorm** (EStackIndex *mean_idx*, EStackIndex *std_idx*)

Category(ies): Statistics

Description: Pushes onto the estack a random float given a specified normal distribution.

Inputs:
mean_idx — Indexes the input mean.
std_idx — Indexes the input standard deviation.

Outputs: None

Assumptions: None

Side Effects: May expand expression stack, cause heap compression, or throw an error.

Availability: On AMS 2.00 and higher.

TI-89 / TI-92 Plus Differences: None

See Also: None

Example: This example pushes a list of 10 random numbers using randnorm (3.0, 1.0).

```
EStackIndex mean_idx, std_idx;
short i;

push_Float( 3.0 );
mean_idx = top_estack;
push_Float( 1.0 );
std_idx = top_estack;

push_quantum( END_TAG );
for (i = 0; i <= 9; i++)
   push_randnorm( mean_idx, std_idx );
push_quantum( LIST_TAG );
```

QstatRcl

Declaration: BOOL **QstatRcl** (void)

Category(ies): Statistics, Variables

Description: Return TRUE if **VarRecall** can be used to read the stat variables, otherwise return FALSE.

Inputs: None

Outputs: TRUE — Stat variables are valid.

FALSE — Cannot read stat variables.

Assumptions: None

Side Effects: None

Availability: On AMS 2.00 and higher.

TI‑89 / TI‑92 Plus Differences: None

See Also: **VarRecall, statStart**

Example: The TI‑BASIC ShowStat command first checks to make sure there are any stat variables to display. If not, it gives an error and returns as shown in the example below.

```
if (!QstatRcl()) {
   DlgNotice(XR_stringPtr(XR_SHOWSTAT),XR_stringPtr(XR_rNostatvar));
   return;
}
```

statEnd

Declaration: void **statEnd** (void)

Category(ies): Statistics, Variables

Description: See **statStart**.

Inputs: None

Outputs: None

Assumptions: None

Side Effects: May cause heap compression.

Availability: On AMS 2.00 and higher.

TI-89 / TI-92 Plus Differences: None

See Also: **statFree, statStart, QstatRcl, VarStore**

Example: See **statStart**.

statFree

Declaration: void **statFree** (void)

Category(ies): Statistics, Variables

Description: Free the stat variables if they exist. This includes regeq and regcoef.

Inputs: None

Outputs: None

Assumptions: None

Side Effects: May set the graph dirty thus forcing a regraph when the grapher is activated.

Availability: On AMS 2.00 and higher.

TI-89 / TI-92 Plus Differences: None

See Also: **statEnd, statStart, QstatRcl, VarStore**

Example: See **statStart**.

statStart

Declaration: void **statStart** (void)

Category(ies): Statistics, Variables

Description: The sequence for an app to store to the stat variables is:

```
statStart()      // all stat vars initialized to invalid floats
RM_Type = . . .  // set to stat calculation type to follow
. . .            // do calculations
. . .            // store to stat vars with VarStore
statEnd()        // allow other apps to use the stat variables
```

The stat variables are now readable by the app or the user but they cannot be stored to by either. Commands like ShowStat will display any stat variables that have valid values stored in them.

> **NOTE:** All of the above code should be placed in a TRY . . . ENDTRY block.

The ONERR part should call **statEnd** first, then it can handle the error. See the example below.

Inputs: None

Outputs: May throw ER_MEMORY if not enough memory to create stat variables.

Assumptions: None

Side Effects: May cause heap compression.

Availability: On AMS 2.00 and higher.
Note that **RM_Type** is only available on 2.04 and higher.

TI-89 / TI-92 Plus Differences: None

See Also: **statEnd, statFree, QstatRcl, VarStore, RM_Type**

(continued)

statStart *(continued)*

Example: This example does no real statistical calculations but shows the steps involved in an app that writes to the system statistical variables.

```
{  Access_AMS_Global_Variables;
   BYTE tag[2];
   BCD16 fN=1.0, fMINX=2.0;

   TRY
      statStart(); /* allow writes to stat vars */
      RM_Type = RM_NONE;
      tag[1] = SYSVAR_TAG;
      tag[0] = SV_N;
      push_Float( fN );
      VarStore( tag+1, STOF_ESI, 0, top_estack );
      tag[0] = SV_MINX;
      push_Float( fMINX );
      VarStore( &tag[1], STOF_ESI, 0, top_estack );
      statEnd();
   ONERR
      statEnd(); /* signal we are not writing to stat vars */
      PASS;  /* pass error on up to previous error handler */
   ENDTRY
   cmd_showstat();
}
```

Appendix A: System Routines — Status Line

See Also:

ST_angle

Declaration: void **ST_angle** (UINT *indic*)

Category(ies): Status Line

Description: Sets the angle status line indicator.

Inputs: *indic* — ST_RAD, set status angle display to “RAD”.
ST_DEG, set status angle display to “DEG”.

Outputs: None

Assumptions: This routine does not set or change the angle mode setting — it only updates the status line display. Use **MO_digestOptions** to change the angle mode setting and the angle status indicator will be updated automatically.

Side Effects: None

Availability: All versions of the TI-89 / TI-92 Plus.

TI-89 / TI-92 Plus Differences: None

See Also: **MO_digestOptions**

Example:

```
ST_angle(ST_RAD);
```

ST_busy

Declaration: void **ST_busy** (UINT *indic*)

Category(ies): Status Line

Description: Sets the busy status line indicator.

Inputs: *indic* — ST_IDLE, turn off busy indicator.
ST_BUSY, set status busy display to “BUSY”.
ST_PAUSE, set status busy display to “PAUSE”.

Outputs: None

Assumptions: The OS event handler sets the busy indicator to “BUSY” before sending an event to your application, then turns the indicator off when the app returns from handling the event. You may want to set the busy indicator to “PAUSE” while waiting for user input.

Side Effects: None

Availability: All versions of the TI-89 / TI-92 Plus.

TI-89 / TI-92 Plus Differences: None

See Also: Not applicable

Example:

```
ST_busy(ST_PAUSE);            /* Turn on PAUSE indicator */
.
. /* get user input */
.
ST_busy(ST_BUSY);             /* Remember to turn BUSY indicator back on */
```

ST_eraseHelp

Declaration: BOOL **ST_eraseHelp** (void)

Category(ies): Status Line

Description: Erases the help message (if any) from the status line and redraws status indicators.

Inputs: None

Outputs: Returns TRUE if a help message was erased and status indicators were redrawn.

Returns FALSE if there was no help message to erase.

Assumptions: It is usually unnecessary for an app or ASM program to call this routine, as most key presses automatically erase the status help message.

Side Effects: None

Availability: All versions of the TI-89 / TI-92 Plus.

TI-89 / TI-92 Plus Differences: None

See Also: **ST_helpMsg**

Example:

```
ST_helpMsg(MY_HELP_MSG);     /* Display help to user */
 .
 . /* process user's input */
 .
ST_eraseHelp();              /* Remove help message */
```

ST_folder

Declaration: void **ST_folder** (char *name*[])

Category(ies): Status Line

Description: Sets the name of the folder which appears at the left end of the status line.

Inputs: *name* — Folder name. This parameter is copied and converted to upper case before being displayed.

Outputs: None

Assumptions: This routine does not set or change the current folder setting — it only updates the status line display. Use **push_setfold** to change the current folder setting and the folder status indicator will be updated automatically.

Side Effects: None

Availability: All versions of the TI-89 / TI-92 Plus.

TI-89 / TI-92 Plus Differences: None

See Also: **push_setfold**

Example:

```
ST_folder("MYFOLDER");
```

ST_helpMsg

Declaration: void **ST_helpMsg** (char *msg*[])

Category(ies): Status Line

Description: Displays a message in the status line. The message stays in the status line until a key is pressed on the keyboard or **ST_eraseHelp** is called.

Inputs: *msg* — Message to display.

> **Note:** The font used in the status line is very small — consider using all upper case letters in your message to improve readability.

Outputs: None

Assumptions: None

Side Effects: None

Availability: All versions of the TI-89 / TI-92 Plus.

TI-89 / TI-92 Plus Differences: None

See Also: **ST_eraseHelp**

Example:

```
ST_helpMsg("WARNING: STRING TOO LONG");
```

ST_progressBar

Declaration: void **ST_progressBar** (ST_PROGRESS_BAR * *pb*, long *low*, long *high*)

Category(ies): Status Line

Description: Creates a progress bar indicator in the status line. A bar will be displayed showing relative progress between values *low* and *high*. A time-consuming process can create a progress bar, then, through subsequent calls to **ST_progressUpdate** or **ST_progressIncrement**, indicate how much work the process has accomplished.

Inputs:

low — Low end of progress. A progress value of *low* displays an empty progress bar.

high — High end of progress. A progress value of *high* displays a solid (filled) progress bar. Progress values between *low* and *high* display a proportionally filled progress bar.

Outputs:

pb — Progress bar structure. This routine initializes the progress bar structure and clears the status line indicators.

Assumptions: None

Side Effects: None

Availability: On AMS 2.00 and higher.

TI-89 / TI-92 Plus Differences: None

See Also: **ST_progressDismiss**, **ST_progressIncrement**, **ST_progressUpdate**

Example:

```
ST_PROGRESS_BAR pb;
 .
 .
 .
ST_progressBar(&pb, 0, 100); /* show progress from 0 to 100 */
for (j = 0; j <= 100; j += 1)
{
  .
  . /* Do some work */
  .
  ST_progressUpdate(&pb, j);
}
ST_progressDismiss(&pb);
```

ST_progressDismiss

Declaration: void **ST_progressDismiss** (ST_PROGRESS_BAR * *pb*)

Category(ies): Status Line

Description: Removes progress bar from status line and redraws status indicators.

Inputs: *pb* — Progress bar structure initialized by a previous call to **ST_progressBar**.

Outputs: Status indicators are redrawn in status line.

Assumptions: None

Side Effects: None

Availability: On AMS 2.00 and higher.

TI-89 / TI-92 Plus Differences: None

See Also: **ST_progressBar, ST_progressIncrement, ST_progressUpdate**

Example:

```
ST_PROGRESS_BAR pb;
 .
 .
 .
ST_progressBar(&pb, 0, 100); /* show progress from 0 to 100 */
for (j = 0; j <= 100; j += 1)
{
  .
  . /* Do some work */
  .
  ST_progressIncrement(&pb, 1);
}
ST_progressDismiss(&pb);
```

ST_progressIncrement

Declaration: void **ST_progressIncrement** (ST_PROGRESS_BAR * *pb*, long *amount*)

Category(ies): Status Line

Description: Updates the progress bar in the status line.

Inputs:
pb — Progress bar structure initialized by a previous call to **ST_progressBar**.

amount — Amount to increment progress value.

Outputs: Progress bar is redrawn with a black bar filling the status line in proportion to the previous value plus *amount*.

Assumptions: None

Side Effects: None

Availability: On AMS 2.00 and higher.

TI-89 / TI-92 Plus Differences: None

See Also: **ST_progressBar, ST_progressDismiss, ST_progressUpdate**

Example:

```
ST_PROGRESS_BAR pb;
 .
 .
 .
ST_progressBar(&pb, 0, 100); /* show progress from 0 to 100 */
for (j = 0; j <= 100; j += 1)
{
  .
  . /* Do some work */
  .
  ST_progressIncrement(&pb, 1);
}
ST_progressDismiss(&pb);
```

ST_progressUpdate

Declaration: void **ST_progressUpdate** (ST_PROGRESS_BAR * *pb*, long *value*)

Category(ies): Status Line

Description: Updates the progress bar in the status line.

Inputs:

pb — Progress bar structure initialized by a previous call to **ST_progressBar**.

value — Progress value. *value* must be from *low* to *high* as established in the initial call to **ST_progressBar**.

Outputs: Progress bar is redrawn with a black bar filling the status line in proportion to *value*.

Assumptions: None

Side Effects: None

Availability: On AMS 2.00 and higher.

TI-89 / TI-92 Plus Differences: None

See Also: **ST_progressBar, ST_progressDismiss, ST_progressIncrement**

Example:

```
ST_PROGRESS_BAR pb;
 .
 .
 .
ST_progressBar(&pb, 0, 100); /* show progress from 0 to 100 */
for (j = 0; j <= 100; j += 1)
{
  .
  . /* Do some work */
  .
  ST_progressUpdate(&pb, j);
}
ST_progressDismiss(&pb);
```

ST_readOnly

Declaration: void **ST_readOnly** (UINT *indic*)

Category(ies): Status Line

Description: Sets the read-only status line indicator.

Inputs: *indic* — ST_READONLY_OFF, turns off the read-only indicator.
ST_READONLY_ON, turns on the read-only indicator.

Outputs: None

Assumptions: This routine does not set or change the read-only state of the text currently being edited — it only updates the status line display. Use **TE_indicateReadOnly** to test the read-only state of the current text edit record and turn on the read-only status line indicator automatically.

Side Effects: None

Availability: All versions of the TI-89 / TI-92 Plus.

TI-89 / TI-92 Plus Differences: None

See Also: **TE_indicateReadOnly**

Example:

```
TE_open(&teRec,&window,NULL,hText,0,0,TE_READ_ONLY|TE_WRAP); /* open read-only text */
TE_indicateReadOnly(&teRec);  /* turn on padlock and dim edit menus */
 .
 .
 .
ST_readOnly(ST_READONLY_OFF);  /* turn off padlock in status line */
```

Appendix A: System Routines — Strings

See Also:

cmpstri

Declaration: int **cmpstri** (unsigned char * *s*, unsigned char * *t*)

Category(ies): Strings

Description: Performs a case-insensitive string comparison.

Inputs:
s — A pointer to a null terminated string.
t — A pointer to a null terminated string.

Outputs:
<0 — Indicates string *s* "is less than" string *t*.
0 — Indicates string *s* "is identical to" string *t*.
>0 — Indicates string *s* "is greater than" string *t*.

Assumptions: None

Side Effects: None

Availability: All versions of the TI‑89 / TI‑92 Plus.

TI-89 / TI-92 Plus Differences: None

See Also: None

Example:

If s1 is a pointer to the string "aBcDe", and s2 is a pointer to the string "AbzdE", then

```
r = cmpstri (s1, s2);
```

will return a negative value in r to indicate that string s1 "is less than" string s2.

FirstNonblank

Declaration: BYTE * **FirstNonblank** (BYTE * *Buf*)

Category(ies): Strings

Description: Return a pointer to the first nonblank character in a string.

Inputs: String pointer.

Outputs: Pointer to first nonblank character in string.

Assumptions: None

Side Effects: None

Availability: On AMS 2.00 and higher.

TI-89 / TI-92 Plus Differences: None

See Also: None

Example: See **push_setfold**.

hStrAppend

Declaration: void **hStrAppend** (HANDLE *hStr1*, UCHAR * *pStr2*)

Category(ies): Strings

Description: Append a string to a handle that contains a string.

Inputs: *hStr1* — Handle to string to be lengthened.

Outputs: *pStr2* — String to append to handle.

Assumptions: None

Side Effects: May cause heap compression. Throws ER_MEMORY if not enough memory to expand handle — hStr1.

Availability: On AMS 2.00 and higher.

TI‑89 / TI‑92 Plus Differences: None

See Also: None

Example: This example creates a command string to set the current folder based on the value selected from a drop-down (OptList[0] index in hOpenFolder).

```
HANDLE hCurFolderText, hOpenFolder;
WORD OptList[3], DefIndex;

if (!(hOpenFolder = VarCreateFolderPopup(&DefIndex,0)))
   ER_throw( ER_MEMORY );
.
.
.
HeapRealloc(hCurFolderText, 20);
memset(HeapDeref(hCurFolderText), 0, 20);
hStrAppend(hCurFolderText, (UCHAR *) XR_stringPtr(XR_setFoldP));
hStrAppend(hCurFolderText, (UCHAR *) PopupText(hOpenFolder,OptList[0]));
hStrAppend(hCurFolderText, (UCHAR *) ")");
```

memchr

Declaration: void * **memchr** (const void * *buf*, int *value*, size_t *count*)

Category(ies): Strings, Utilities

Description: Searches the first *count* bytes of the block pointed to by *buf* for the byte *value.*

Inputs:
buf — Buffer to search.
value — Value to search for.
count — Number of bytes to search.

Outputs: A pointer to the byte value found or NULL if the value was not found.

Assumptions: None

Side Effects: None

Availability: All versions of the TI-89 / TI-92 Plus.

TI-89 / TI-92 Plus Differences: None

See Also: **memcmp, strchr**

Example: The following routine, strnlen, returns the length of a string but stops at maxlen characters.

```
short strnlen( char *str, short maxlen )
{
   char *match;

   if (match = memchr( str, 0, maxlen ))
      return ( match - str );
   else
      return maxlen;
}
```

memcmp

Declaration: int **memcmp** (const void * *buf1*, const void * *buf2*, size_t *length*)

Category(ies): Strings, Utilities

Description: Compares *length* bytes of *buf1* against *buf2* and returns the result of the compare.

Inputs:

buf1 — Pointer to first block to compare.

buf2 — Pointer to second block to compare.

length — Number of bytes to compare.

Outputs: An integer value is returned as follows.

< 0 — If *buf1* is less than *buf2*.

= 0 — If *buf1* is the same as *buf2*.

> 0 — If *buf1* is greater than *buf2*.

Note that the return value is the result of subtracting the first pair of values that differ in the two blocks being compared based on them being **signed** chars. Compare this to **memucmp** which computes the difference based on the values being unsigned chars.

Assumptions: None

Side Effects: None

Availability: All versions of the TI-89 / TI-92 Plus.

TI-89 / TI-92 Plus Differences: None

See Also: **memchr, memucmp, strcmp**

Example: See **memucmp** and **FSetPos**.

memucmp

Declaration: int **memucmp** (const void * *buf1*, const void * *buf2*, size_t *length*)

Category(ies): Strings, Utilities

Description: Compares *length* bytes of *buf1* against *buf2* and returns the result of the compare assuming the values are unsigned chars.

Inputs:
buf1 — Pointer to first block to compare.
buf2 — Pointer to second block to compare.
length — Number of bytes to compare.

Outputs: An integer value is returned similar to **memcmp** except the input values are assumed to be **unsigned** chars.
< 0 — If *buf1* is less than *buf2*.
= 0 — If *buf1* is the same as *buf2*.
> 0 — If *buf1* is greater than *buf2*.

Assumptions: None

Side Effects: None

Availability: On AMS 2.00 and higher.

TI-89 / TI-92 Plus Differences: None

See Also: **memcmp**

Example: This example shows the difference between **memcmp** and **memucmp**. The output is as follows:

FF80 0002 FFFE
0080 FF02 00FE

```
int v1, v2, v3;
int v1u, v2u, v3u;
BYTE b255 = 255, b127 = 127, b1 = 1;
char buf[100];

v1 = memcmp( &b255, &b127, 1 ); v1u = memucmp( &b255, &b127, 1 );
v2 = memcmp( &b1  , &b255, 1 ); v2u = memucmp( &b1  , &b255, 1 );
v3 = memcmp( &b255, &b1  , 1 ); v3u = memucmp( &b255, &b1  , 1 );

sprintf( buf, "%04X %04X %04X\n%04x %04x %04x", v1, v2, v3, v1u, v2u, v3u );
DlgNotice( "memucmp", buf );
```

push_char

Declaration: void **push_char** (EStackIndex *i*)

Category(ies): Strings

Description: Pushes a string containing the specified ASCII character.

Inputs: *i* — EStackIndex of a number between 0 and 255, inclusive.

Outputs: Pushes a tokenized string onto the estack containing the ASCII character specified by the input value. If the input is a list of numbers, the routine returns the corresponding list of strings.

Assumptions: None

Side Effects: May cause expression stack to grow or heap compression. Will throw an error if the input is not a whole number or is outside the input domain.

Availability: On AMS 2.00 and higher.

TI-89 / TI-92 Plus Differences: None

See Also: **push_format, push_instring, push_ord, push_str_to_expr, push_string, push_zstr**

Example:

If j indexes the bolded tag in the integer 65 as follows
65 1 **NONNEGATIVE_INTEGER_TAG**

then

```
push_char (j);
```

pushes the string “A” onto the stack such that **top_estack** points to the bolded tag as follows.
0 65 0 **STR_DATA_TAG**

push_instring

Declaration: void **push_instring** (EStackIndex *i*, EStackIndex *j*, EStackIndex *k*)

Category(ies): Strings

Description: Searches for a substring within a string.

Inputs:
i — The string to search.
j — The substring to search for.
k — The starting position.

Outputs: Returns a tokenized integer on the estack. The integer represents the position of the first occurrence of substring *j* within the search string *i* at or after the start position *k*. If the substring is not found, zero is pushed.

Assumptions: None

Side Effects: May expand expression stack and may cause heap compression.

Availability: On AMS 1.05 and higher.

TI-89 / TI-92 Plus Differences: None

See Also: **push_char, push_format, push_ord, push_str_to_expr, push_string, push_zstr**

Example:

If i indexes the bolded tag in the following tokenized string “hello”
0 h e l l o 0 **STR_DATA_TAG**

and j indexes the bolded tag in the following tokenized string “l”
0 l 0 **STR_DATA_TAG**

and k indexes the bolded tag in the following tokenized integer 1
1 1 **NONNEGATIVE_INTEGER_TAG**

then

```
push_instring (i, j, k);
```

pushes the tokenized integer 3.
3 1 NONNEGATIVE_INTEGER_TAG

push_ord

Declaration: void **push_ord** (EStackIndex *i*)

Category(ies): Strings

Description: Pushes the ASCII value of the first character of a string.

Inputs: *i* — EStackIndex of a tokenized string or list of tokenized strings.

Outputs: Pushes the ASCII value of the first character of the string as a tokenized integer. If the input is a list of strings, then the routine pushes a list of the appropriate ASCII values.

Assumptions: None

Side Effects: May cause expression stack to grow or heap compression.

Availability: On AMS 2.00 and higher.

TI-89 / TI-92 Plus Differences: None

See Also: **push_char, push_format, push_instring, push_str_to_expr, push_string, push_zstr**

Example:

```
push_ord (“hello”);
```

Pushes the ASCII value for ‘h’ onto the estack as a tokenized integer such that **top_estack** points to the bolded tag as follows.
104 1 **NONNEGATIVE_INTEGER_TAG**

push_str_to_expr

Declaration: void **push_str_to_expr** (EStackIndex *i*)

Category(ies): Strings

Description: Tokenizes and “executes” a string. This routine uses **push_parse_text** to convert the string to externally tokenized form. Then, it applies **push_simplify_statements** to produce a fully simplified result.

Inputs: *i* — Indexes the tokenized form of a string.

Outputs: If tokenization and execution of the string produces a result, it is pushed onto the estack. If this process does not produce a result, then no value is pushed.

Assumptions: None

Side Effects: May cause expression stack to grow or heap compression. May throw a variety of errors associated with tokenization of the string or simplification of the resulting form.

Availability: On AMS 1.05 and higher.

TI-89 / TI-92 Plus Differences: None

See Also: **push_char, push_format, push_instring, push_ord, push_string, push_zstr**

(continued)

push_str_to_expr *(continued)*

Example:

If j indexes the bolded tag in the string “ln(1)” as follows
0 l n (1) 0 **STR_DATA_TAG**

```
push_str_to_expr (j);
```

temporarily produces the tokenized form of ln(1) at the bolded tag as follows
1 1 NONNEGATIVE_INTEGER_TAG **LN_TAG**

then simplifies that expression onto the estack such that **top_estack** points to the bolded tag of the final result 0.
0 **NONNEGATIVE_INTEGER_TAG**

If j is the EStackIndex of the string “0→m : For n,1,10 : n+m→m : EndFor”, then

```
push_str_to_expr (j);
```

tokenizes and executes that sequence of statements and returns the tokenized form of the result, 55, on the estack.

If j is the EStackIndex of the string “ClrIO” then

```
push_str_to_expr (j);
```

tokenizes and executes that statement, and since it does not return a value, no value is returned on the estack.

push_string

Declaration: void **push_string** (EStackIndex *i*)

Category(ies): Strings

Description: Linearly detokenizes the expression indexed by *i*, and then pushes the tokenized string form of that result onto the estack.

Inputs: *i* — An externally tokenized expression.

Outputs: Pushes the tokenized string form of the linearly detokenized form of the input expression onto the estack.

Assumptions: None

Side Effects: May cause expression stack to grow or heap compression.

Availability: On AMS 1.05 and higher.

TI-89 / TI-92 Plus Differences: None

See Also: **push_char, push_format, push_instring, push_ord, push_str_to_expr, push_zstr**

Example:

If i indexes the bolded tag in the externally tokenized form of a + b as follows
A_VAR_TAG B_VAR_TAG **ADD_TAG**

then

```
push_string (i);
```

pushes the tokenized string form onto the estack such that **top_estack** points to the bolded tag of the expression a + b as follows.
0 a + b 0 **STR_DATA_TAG**

push_zstr

Declaration: void **push_zstr** (char * *str*)

Category(ies): Strings, EStack Utilities

Description: Pushes a null terminated string onto the estack in tokenized form.

Inputs: *str* — A pointer to a null terminated string.

Outputs: Pushes the tokenized form of the string onto the estack.

Assumptions: None

Side Effects: May cause expression stack to grow or heap compression.

Availability: On AMS 2.00 and higher.

TI-89 / TI-92 Plus Differences: None

See Also: **push_char, push_format, push_instring, push_ord, push_str_to_expr, push_string**

Example:

```
push_zstr (“hello”);
```

Pushes the following tokenized form onto the estack.
0 h e l l o 0 STR_DATA_TAG

sprintf

Declaration: int **sprintf** (char * *buf*, const char * *fmt_str*, . . .)

Category(ies): Strings, Utilities

Description: The **sprintf** function formats and prints a series of characters to *buf*. Each argument (if any) is converted and stored according to the corresponding format specification in *fmt_str*. A null-character ('\0') is appended at the end of the characters written.

The following standard conversions are supported.

%d %i — Integer argument is converted to signed decimal notation.

%u — Integer argument is converted to unsigned decimal notation.

%o — Integer argument is converted to octal notation.

%b — Integer argument is converted to binary notation.

%x %X — Integer argument is converted to hex notation (lower/upper case).

%p — Void * argument is converted to 0xhhhhhhhh form (same as %#.8x).

Note:	The above conversions may be preceded by an 'l' to signify a long integer argument.

%f — Double argument is converted to decimal notation of the form [-]ddd.ddd (where ddd represents zero or more digits).

%e %E — Double argument is converted to decimal notation of the form [-]d.ddde[+-]ddd.

%g %G — Double argument is converted according to the %f or the %e format depending on the value converted.

%c — Integer argument is converted into an unsigned char.

%s — Char * argument is a pointer to string; string characters are written up to null character or specified precision count.

%% — Literal '%'.

(continued)

sprintf *(continued)*

Description: **The following formats are unique to AMS.**

(continued)

%r %R — Double argument is converted to decimal notation of the form [-]d.ddde[+-]ddd using engineering notation.

%y %Y — Double argument is converted according to the %f or the %e format depending on the system MODE settings.

The following modifiers are scanned in the order shown.

' ' — Prefix positive values with a space.

\+ — Prefix positive values with a +.

\- — Left justify value in field.

| — Center value in field.

— Use alternate forms as follows.

- %o — Prefix with 0.
- %x %X — Prefix with 0x and 0X respectively.
- %e %E %f — Force a decimal point.
- %g %G — Force decimal point and leave trailing zeros.

0 — Pad with leading zeros.

AMS specific modifiers.

^ — Do not print leading 0 for values between -1 and 1, use LF_NEGATIVE for negation instead of minus sign '-', use LF_EXPONENT for exponent character instead of 'e' or 'E', do not print leading + on positive exponent values, force decimal point in all float strings.

z — Zap trailing blanks.

Precision supported in strings

%X.Ys — X-character wide field, print at most Y characters (even if the string is longer). If X or Y is a *, the value is taken from the next argument.

(continued)

sprintf *(continued)*

Inputs: *buf* — Output character buffer.

fmt_str — Input format string.

Outputs: The number of characters stored in *buf*, not counting the terminating null-character.

Assumptions: None

Side Effects: None

Availability: All versions of the TI-89 / TI-92 Plus.

TI-89 / TI-92 Plus Differences: None

See Also: **scanf**

Example: See **strcpy**, **strpbrk**, **strrchr**, and **strtol**.

strcat

Declaration: char * **strcat** (char * *s1*, const char * *s2*)

Category(ies): Strings

Description: Appends a copy of the string pointed to by *s2* to the end of the string pointed to by *s1*, overwriting the null character terminating the string pointed to by *s1*.

Inputs: *s1* — Character string.

s2 — Character string to append.

Outputs: Returns the new value of string *s1.*

Assumptions: *s1* points to a buffer large enough to hold the concatenated string.

Side Effects: None

Availability: All versions of the TI‑89 / TI‑92 Plus.

TI‑89 / TI‑92 Plus Differences: None

See Also: **strncat**

Example:

```
void ValToStr( char *Buf, BCD16 *Val )
{           /* Format floating point Val as a string and append to Buf */
   Access_AMS_Global_Variables;
   HANDLE hText;
   EStackIndex OldTop;

   if (!is_float_transfinite(*Val))  {
      OldTop = top_estack;
      push_Float( *Val );
      hText = Parse1DExpr(top_estack, FALSE, 0);
      top_estack = OldTop;
      if (hText != H_NULL {
         strcat( Buf, " = " );
         strcat( Buf, HeapDeref(hText) );  /* Buf contains . . . . = #### */
         HeapFree( hText );
      }
   }
}
```

strchr

Declaration: char * **strchr** (const char * *str*, int *c*)

Category(ies): Strings

Description: Locates the first occurrence of the character *c* in the string pointed to by *str*. The character *c* may be any character including the null character (\0).

Inputs: *str* — Character string to search.

c — Character to locate.

Outputs: Returns a pointer to the first occurrence of *c* in *str*. If *c* is not in *str*, **strchr** returns a null pointer.

Assumptions: None

Side Effects: None

Availability: All versions of the TI-89 / TI-92 Plus.

TI-89 / TI-92 Plus Differences: None

See Also: **strrchr, memchr, strspn, strcspn, strpbrk, strstr**

Example:

```
short KeyYesOrNo( WORD Key )
/* If Key is ENTER or is an alpha and in XR_YesStr return TRUE,
   if it is ESC or is an alpha and in XR_NoStr return FALSE,
   otherwise return -1.
*/
{
   if (Key == KB_ENTER)
      return TRUE;
   if (Key == KB_ESC)
      return FALSE;
   if (Key <= 0xFF) {
      if (strchr(XR_stringPtr(XR_YesStr), (BYTE) Key ))
         return TRUE;
      if (strchr(XR_stringPtr(XR_NoStr), (BYTE) Key ))
         return FALSE;
   }
   return -1;
}
```

strcmp

Declaration: int **strcmp** (const char * *s1*, const char * *s2*)

Category(ies): Strings

Description: Lexicographically compares two strings.

Inputs: *s1* — Character string.

s2 — Character string.

Outputs: Returns zero if the strings are identical, a positive number if the string pointed to by *s1* is greater than the string pointed to by *s2*, or a negative number if the string pointed to by *s1* is less than the string pointed to by *s2*.

If the strings differ, the value of the first nonmatching character in *s2* subtracted from the corresponding character in *s1* is returned.

Assumptions: None

Side Effects: None

Availability: All versions of the TI-89 / TI-92 Plus.

TI-89 / TI-92 Plus Differences: None

See Also: **strncmp, memcmp**

Example:

```
BOOL isSolverVar( char *str )
/* Return TRUE if string str is one of the solver variables
   else return FALSE.
*/
{
   char eqnstr[] = "eqn";
   char expstr[] = "exp";

   if( (strcmp( str, eqnstr) == 0) || (strcmp( str, expstr) == 0) )
      return TRUE;
   return FALSE;
}
```

strcpy

Declaration: char * **strcpy** (char * *s1*, const char * *s2*)

Category(ies): Strings

Description: Copies the string pointed to by *s2* to the buffer pointed to by *s1*. If the objects pointed to by *s1* and *s2* overlap in memory, the behavior is undefined.

Inputs: *s1* — Buffer to copy to.

s2 — String to be copied into *s1.*

Outputs: Returns the value of *s1.*

Assumptions: *s1* points to a buffer large enough to hold *s2.*

Side Effects: None

Availability: All versions of the TI-89 / TI-92 Plus.

TI-89 / TI-92 Plus Differences: None

See Also: **strncpy**, **memcpy**, **memmove**

Example:

```
void gr_seq_axes_lbl( SBYTE ax, char buf[] )
/* Create requested string in buf for sequence axes labels.
   input:  ax = -1 = n
              = 0 = u
              > 0 = u1 - u99
           buf = buffer for string
*/
{
   if( ax < 0 )
      strcpy( buf, "n" );                    /* n is on this axis */
   else
   {                                         /* want u or u1-u99 */
      strcpy( buf, "u" );
      if( ax )
         sprintf( buf + 1, "%d", (BYTE)ax );   /* need to add 1-99 to string */
   }
}
```

strcspn

Declaration: size_t **strcspn** (const char * *s1*, const char * *s2*)

Category(ies): Strings

Description: Calculates the index of the first character in string *s1* that matches any of the characters in string *s2*. This value is the length of a leading substring of the string pointed to by *s1* which consists entirely of characters not in the string pointed by *s2*.

Inputs: *s1* — Character string to search.

s2 — Character set.

Outputs: Return the length of the substring in *s1* that consists entirely of characters not found in string *s2*. If string *s1* contains no characters from string *s2*, **strcspn** returns the length of string *s1*.

Assumptions: None

Side Effects: None

Availability: All versions of the TI-89 / TI-92 Plus.

TI-89 / TI-92 Plus Differences: None

See Also: **strspn, strpbrk, strchr, strrchr, strstr, memchr**

Example:

```
BOOL ck_vowel( const char *str1, size_t *lenp )
/* ck_vowel - Determine the length of a leading substring of a string that
              contains no vowels.

            input:  str1 = string to search
                    lenp = ptr to location to return length of substring

            returns TRUE if str1 contains a vowel, else FALSE
*/
{

   if( ( *lenp = strcspn(str1, "aeiou")) == strlen(str1) )
      return( FALSE );   /* no vowels in str1 */
   else
      return( TRUE );    /* str1 contains a vowel after *lenp nonvowels */
}
```

stricmp

Declaration: int **stricmp** (const char * *s1*, const char * *s2*)

Category(ies): Strings

Description: This is a case insensitive version of **strcmp**.

Inputs: *s1* — Character string.

s2 — Character string.

Outputs: Ignoring the case of the letters, returns zero if the strings are identical, a positive number if the string pointed to by *s1* is greater than the string pointed to by *s2*, or a negative number if the string pointed to by *s1* is less than the string pointed to by *s2*.

If the strings differ, the value of the first nonmatching character in *s2* subtracted from the corresponding character in *s1* is returned. The subtraction casts the input strings to unsigned chars so that the characters in the range 128 . . . 255 are considered above the characters in the range 0 . . . 127.

Assumptions: None

Side Effects: None

Availability: On AMS 2.00 and higher.

TI-89 / TI-92 Plus Differences: None

See Also: **strcmp, strncmp, memcmp**

Example:

```
/* Return TRUE if strPtr points to the string “all” regardless of case */
if (stricmp((strPtr, “all”) == 0)
   return TRUE;
```

strlen

Declaration: size_t **strlen** (const char * *str*)

Category(ies): Strings

Description: Returns the length in bytes of the string pointed to by *str*, not counting the terminating null character.

Inputs: *str* — Character string.

Outputs: Length of the string pointed to by *str.*

Assumptions: None

Side Effects: None

Availability: All versions of the TI‑89 / TI‑92 Plus.

TI‑89 / TI‑92 Plus Differences: None

See Also: **strspn**, **strcspn**

Example:

```
void hStrAppend( HANDLE hStr1, UCHAR *pStr2 )
/* hStrAppend - append string to a handle.

         input:  hStr1 = handle to string to be lengthened,
                 pStr2 = string to append to handle.
*/
{
   UCHAR *pStr1;
   ULONG lStr1, lStr2, lBuf;

   pStr1 = HeapDeref(hStr1);
   lStr1 = strlen((char *)pStr1);            /* find length of string in handle */
   lStr2 = strlen((char *)pStr2);            /* length of string to append */
   lBuf = lStr1 + lStr2 + 1;                 /* calc new space requirement */
   if (HeapRealloc(hStr1, lBuf) == 0)        /* try to get new space */
      ER_throw(ER_MEMORY);
   pStr1 = (UCHAR *)HeapDeref(hStr1) + lStr1;  /* point to end of original string */
   memcpy(pStr1, pStr2, lStr2+1);            /* append new string  */
}
```

strncat

Declaration: char * **strncat** (char * *s1*, const char * *s2,* size_t *count*)

Category(ies): Strings

Description: Appends not more than *count* characters from the string pointed to by *s2* to the end of the string pointed to by *s1*. The null character terminating *s1* is overwritten by the first character in *s2*. A terminating null character is appended to the result.

Inputs:

s1 — Character string.

s2 — Character string to append.

count — Number of characters to append.

Outputs: Returns the value of *s1.*

Assumptions: *s1* points to a buffer large enough to hold the concatenated string. Since **strncat** appends a null character to the result, it may add *count* + 1 characters to the string.

Side Effects: None

Availability: All versions of the TI‑89 / TI‑92 Plus.

TI‑89 / TI‑92 Plus Differences: None

See Also: **strcat**

Example: See **strncpy**.

strncmp

Declaration: int **strncmp** (const char * *s1*, const char * *s2*, size_t *maxlen*)

Category(ies): Strings

Description: Compare at most *maxlen* characters (stopping after a null character) of *s1* and *s2* and return the same result as **strcmp**.

Inputs:

s1 — Character string.

s2 — Character string.

maxlen — Maximum length to search.

Outputs: Returns zero if the strings are identical, a positive number if the string pointed to by *s1* is greater than the string pointed to by *s2*, or a negative number if the string pointed to by *s1* is less than the string pointed to by *s2*.

If the strings differ, the value of the first nonmatching character in *s2* subtracted from the corresponding character in *s1* is returned. The subtraction casts the input strings to unsigned chars so that the characters in the range 128 . . . 255 are considered above the characters in the range 0 . . . 127.

Assumptions: None

Side Effects: None

Availability: All versions of the TI-89 / TI-92 Plus.

TI-89 / TI-92 Plus Differences: None

See Also: **strcmp, memcmp**

Example: If keyName points to a character string that begins with the string “XR_”, this example returns a pointer to the string after the “_” character.

```
if (0 == strncmp( keyName, "XR_", 3 ))
   return keyName + 3;
```

strncpy

Declaration: char * **strncpy** (char * *s1*, const char * *s2*, size_t *count*)

Category(ies): Strings

Description: Copies no more than *count* characters from the string pointed to by *s2* to the character buffer pointed to by *s1*. The result will not be null terminated if string *s2* is longer than *count* characters. If the objects pointed to by *s1* and *s2* overlap in memory, the behavior is undefined.

Inputs:
- *s1* — Buffer to copy to.
- *s2* — Character string to copy from.
- *count* — Number of characters to copy.

Outputs: Returns the value of *s1.*

Assumptions: *s1* points to a buffer large enough to hold *count* characters.

Side Effects: None

Availability: All versions of the TI‑89 / TI‑92 Plus.

TI‑89 / TI‑92 Plus Differences: None

See Also: **strcpy**, **memcpy**, **memmove**

Example:

```
short CustomError( short errCode, const BYTE *msg)
/*  Create custom error message. */
{
   BYTE buf[260];

   memset( buf, 0, sizeof(buf) );
   strncpy( (char *) buf, (char *) msg, 128 );
   strcat( (char *) buf, "\n" );
   strncat( (char *) buf, (char *) find_error_message(errCode), 128 );
   return( DlgMessage((const char *) XR_stringPtr(XR_ERROR),
      (const char *) buf, PDB_OK, 0) );
}
```

strpbrk

Declaration: char * **strpbrk** (const char * *s1*, const char * *s2*)

Category(ies): Strings

Description: Locates the first occurrence of any character from the string pointed to by *s2* in the string pointed to by *s1*.

Inputs: *s1* — Pointer to character string.

s2 — Pointer to character set.

Outputs: Returns a pointer to the located character, or a null pointer if no character from *s2* is found in *s1*.

Assumptions: None

Side Effects: None

Availability: All versions of the TI-89 / TI-92 Plus.

TI-89 / TI-92 Plus Differences: None

See Also: **strchr, strcspn, strrchr, strspn, strstr, strtok, memchr**

Example:

```
BOOL ck_vowel2( const char *str1, char *buf )
/* ck_vowel2 - Return TRUE if found a vowel in str1 and build message in buf,
               else return FALSE
*/
{
char *ptr;

   if( ( ptr = strpbrk(str1, "aeiou")) == NULL )
       return( FALSE );   /* no vowels in str1  */
   sprintf( buf, "The first vowel found in '%s' is: %c\n", str1, *ptr);
            return( TRUE );
}
```

strrchr

Declaration: char * **strrchr** (const char * *str*, int *c*);

Category(ies): Strings

Description: Locates the last occurrence of the character *c* in the string pointed to by *str*.

Inputs: *str* — Pointer to character string.

c — Character to search for.

Outputs: Returns a pointer to the located character, or a null pointer if *c* is not found in *str*.

Assumptions: None

Side Effects: None

Availability: All versions of the TI-89 / TI-92 Plus.

TI-89 / TI-92 Plus Differences: None

See Also: **strchr, strcspn, strspn, strstr, strpbrk, memchr**

Example:

```
BOOL findlastchar( const char *str1, int ch, char *buf )
/* findlastchar - Return TRUE if found ch in str1 and build message in buf
                  indicating the last occurrence, else return FALSE.
*/
{
char *ptr;

   if( ( ptr = strrchr(str1, ch)) == NULL )
      return( FALSE );   /* ch not found in str1 */
   sprintf( buf, "Last '%c' is found at position %d in string: %s\n",
            ch, (ptr - str1 + 1), str1);
      return( TRUE );
}
```

strspn

Declaration: size_t **strspn** (const char * *s1*, const char * *s2*)

Category(ies): Strings

Description: Calculates the length of a leading substring of the string pointed to by *s1*, which consists entirely of characters in the string pointed to by *s2*.

Inputs: *s1* — Character string to search.

s2 — Character set.

Outputs: Returns the length of the leading substring in *s1* that consists entirely of characters in string *s2*. If *s1* contains no characters from *s2*, zero is returned.

Assumptions: None

Side Effects: None

Availability: All versions of the TI-89 / TI-92 Plus.

TI-89 / TI-92 Plus Differences: None

See Also: **strcspn, strpbrk, strchr, strrchr, strstr, memchr**

Example:

```
BOOL ck_octal( const char *str1, size_t *lenp )
/* ck_octal - Determine the length of a leading substring of a string that
              consists of octal digits.

            input:  str1 = string to search
                    lenp = ptr to location to return length of substring

            returns TRUE if str1 contains only octal digits, else FALSE
*/
{

   if( ( *lenp = strspn(str1, "01234567")) == 0 )
      return( FALSE );  /* no octal digits in str1  */
   else if( *lenp == strlen(str1) )
      return( TRUE );   /* str1 consists entirely of octal digits  */
   else
      return( FALSE );  /* the first *lenp digits of str1 are octal  */
}
```

strstr

Declaration: char * **strstr** (const char * *s1*, const char * *s2*)

Category(ies): Strings

Description: Searches the string pointed to by *s1* for a substring matching the string pointed to by *s2*.

Inputs: *s1* — Character string to search in.

s2 — Character string to search for.

Outputs: Returns a pointer to the first character of the matching substring in *s1*, or a null pointer if no matching substring is found.

Assumptions: None

Side Effects: None

Availability: All versions of the TI-89 / TI-92 Plus.

TI-89 / TI-92 Plus Differences: None

See Also: **strchr, memchr**

Example:

```
BOOL findstr( const char *str1, const char *str2, char *buf )
/* findstr - Find the first instance of string str2 in string str1.
             Return TRUE if found str2 in str1 and build message in buf,
             else return FALSE.
*/
{
char *ptr;

   if( ( ptr = strstr(str1, str2)) == NULL )
      return( FALSE );   /* str2 not found in str1 */
   sprintf( buf, "Found '%s' at position %d in string: %s\n",
            str2, (ptr - str1 + 1), str1);
   return( TRUE );
}
```

strtok

Declaration: char * **strtok** (char * *str1*, const char * *str2*)

Category(ies): Strings

Description: The **strtok** function considers the string *str1* to consist of a sequence of zero or more text tokens, separated by spans of one or more characters from the string *str2*. The first call to **strtok** returns a pointer to the first token in *str1* and writes a null character into *str1* immediately following the returned token. Subsequent calls with NULL for the first argument will work through string *str1* in this way until no tokens remain. The separator string *str2* may be different from call to call.

Inputs: *str1* — Character string or NULL.

str2 — Character string.

Outputs: A pointer to the token found in *str1* is returned, NULL if there are no more tokens.

Assumptions: *str1* points to a character string in RAM.

Side Effects: None

Availability: All versions of the TI-89 / TI-92 Plus.

TI-89 / TI-92 Plus Differences: None

See Also: **strcspn, strspn**

Example: This example creates a string with all of the C "words" from a C program and pushes it onto the estack.

```
char cpunct[] = " %,.()!=+-#<>{}*/'[];\n\\\"";
char testIn[] = "short test( char * arg1 ) { return strlen(arg1+1) };";
char testOut[256] = "";
char *token;

token = strtok( testIn, cpunct );
while (token != NULL) {
   strcat( testOut, token );
   strcat( testOut, " " );
   token = strtok( NULL, cpunct );
}
/* Result is "short test char arg1 return strlen arg1 1" */
push_zstr( testOut );
```

XR_stringPtr

Declaration: char * **XR_stringPtr** (ULONG *n*)

Category(ies): Strings

Description: Returns a string pointer from the string cross-reference table.

Nearly all strings in the calculator are collected together in the string cross-reference table. This includes the text of menus, dialog boxes, error messages, and such. Language localizers can be installed in the calculator to override any and all strings in the cross-reference table to provide local language customization of the calculator.

String numbers in the range 0x0 to 0x7FF are reserved for system string numbers. Every system string number has an associated macro beginning with "XR_" in tiams.h. String numbers in the range 0x800 (OO_FIRST_APP_STRING) to 0xFFF are reserved for applications. See sections **7.3.1.3.19 Attribute OO_APPSTRING** and **7.3.1.4 Example** for an illustration of how to create a string cross-reference table in your app.

> **Note:** App menu IDs (hence their corresponding string numbers) are limited to the range 0x800 – 0xEFF. The OS uses menu IDs in the range 0xF00 – 0xFFF for some dynamically-built menus.

The string cross-reference table is implemented as frame attribute slots 0x800 through 0x17FF. System strings occupy slots 0x800 (OO_FIRST_STRING) through 0xFFF. App strings occupy slots 0x1000 (OO_APPSTRING) through 0x17FF. Hence, requesting string *n* fetches the pointer to string OO_FIRST_STRING + *n*. **XR_stringPtr** looks first through the running app's string cross-reference table, then through the current language localizer string table (if any), and finally at the system string table.

Inputs: *n* — Number of string to fetch.

Outputs: Returns a pointer to the requested string. This is usually a pointer to Flash memory, so copy it to RAM first if you need to modify it.

Assumptions: None

Side Effects: None

Availability: All versions of the TI-89 / TI-92 Plus.

(continued)

XR_stringPtr *(continued)*

TI-89 / TI-92 Plus Differences: None

See Also: Not applicable

Example:

```
#define VERSION_MAJOR 3
#define VERSION_MINOR 2
char buf[40];
sprintf(buf, "%s %d.%02d", XR_stringPtr(XR_Version), VERSION_MAJOR, VERSION_MINOR);
```

This puts “Version 3.02” into buf.

Appendix A: System Routines — Symbol Table Utilities

See Also:

AddSymToFolder

Declaration: HSYM **AddSymToFolder** (const BYTE * *SymName*, const BYTE * *FolderName*)

Category(ies): Symbol Table Utilities (low-level)

Description: Add a symbol to a specific folder (unlike **SymAdd** which adds to the current folder if no folder name given). Like **SymAdd**, if the symbol already exists and it has a value, that value will be deleted (unless it is a folder name, then zero is returned).

Inputs:
SymName — Pointer to tokenized symbol name to add.
FolderName — Pointer to tokenized folder name to add.

Outputs: HSYM of newly created symbol.

Assumptions: None

Side Effects: None

Availability: All versions of the TI-89 / TI-92 Plus.

TI-89 / TI-92 Plus Differences: None

See Also: **SymAdd, FolderAdd, FindSymInFolder**

Example: See **FolderAdd**.

DerefSym

Declaration: SYM_ENTRY * **DerefSym** (HSYM *hsym*)

Category(ies): Symbol Table Utilities

Description: Convert an HSYM into a direct SYM_ENTRY pointer.

Inputs: *hsym* — HSYM of a variable (from **VarRecall** or **VarStore** for example).

Outputs: SYM_ENTRY pointer represented by *hsym* or NULL if *hsym* is 0.

Assumptions:

> **NOTE:** SYM_ENTRY pointers are invalid after heap compression. HSYMs become invalid after adding or removing symbol table entries.

Side Effects: None

Availability: All versions of the TI-89 / TI-92 Plus.

TI-89 / TI-92 Plus Differences: None

See Also: **VarRecall, VarStore**

Example: See **VarStore**.

FindSymInFolder

Declaration: HSYM **FindSymInFolder** (const BYTE * *SymName*, const BYTE * *FolderName*)

Category(ies): Symbol Table Utilities (low-level)

Description: Search folder *FolderName* for the symbol *SymName* and return the HSYM if found or zero if not found. Note that most reserved symbols are not stored in the symbol table.

Inputs: *SymName* — Pointer to tokenized symbol name to search for.

FolderName — Pointer to tokenized folder name to search.

Outputs: HSYM of symbol or zero if not found.

Assumptions: None

Side Effects: None

Availability: All versions of the TI‑89 / TI‑92 Plus.

TI‑89 / TI‑92 Plus Differences: None

See Also: **SymAdd, FolderAdd, AddSymToFolder, VarRecall**

Example: See **FolderAdd**.

FolderAdd

Declaration: HANDLE **FolderAdd** (const BYTE * *FolderName*)

Category(ies): Symbol Table Utilities (low-level)

Description: Add the given *FolderName* to the home folder. Return the HANDLE of the new folder if OK, H_NULL if error (not enough memory or folder already exists). Note that reserved names are not valid folder names and that this routine does not check for reserved names. It is up to the caller to validate the folder name before calling this routine. In general, **cmd_newfold** should be used to create folders. This routine can be used to create temporary folders which begin with a number and are not displayed in VAR-LINK. The following temporary folder numbers are reserved for the system (they are all four digit numbers):

0001 . . . 8192	—	TI-BASIC local symbols.
9998	—	Data/Matrix Editor.
9999	—	Reserved.

Inputs: *FolderName* — Pointer to tokenized folder name to create.

Outputs: HANDLE of new folder or H_NULL if an error.

Assumptions: None

Side Effects: May cause heap compression.

Availability: All versions of the TI-89 / TI-92 Plus.

TI-89 / TI-92 Plus Differences: None

See Also: **cmd_newfold, AddSymToFolder, FindSymInFolder**

(continued)

FolderAdd *(continued)*

Example: This example creates a temporary folder, adds a locked symbol to it, dumps the symbol table to the link port (see **HeapWalk**), looks up the same symbol just added and finally deletes the temporary folder which deletes everything in the folder including the locked symbol.

```
BYTE foldName[] =  {0,'9','0','0','0',0};
BYTE symName[] =  {0,'s','y','m','1',0};
HSYM hsym1, hsym2;

if (FolderFind(foldName+5) == FL_NOTFOUND)
   if( !FolderAdd( foldName+5 ))
      ER_throw( ER_MEMORY );
hsym1 = AddSymToFolder( symName+5, foldName+5 );
DerefSym(hsym1)->Flags |= SF_LOCK;   /* to show FolderDel will still delete it */
HeapWalk( H_WALK_SYM );   /* dump symbol table to link port */
/* lookup same symbol we just added */
hsym2 = FindSymInFolder( symName+5, foldName+5 );
/* HSYMs better match! */
if (hsym1 != hsym2)
   ER_THROW( FIRST_INTERNAL_ERR );
FolderDel( foldName+5, FALSE );
HeapWalk( H_WALK_SYM );   /* our temporary folder should now be gone */
```

FolderCount

Declaration: WORD **FolderCount** (const SYM_ENTRY * *SymPtr*)

Category(ies): Symbol Table Utilities (low-level)

Description: Return number of used entries in a folder.

Inputs: *SymPtr* — SYM_ENTRY pointer to a folder.

Outputs: Number of used symbol table entries in that folder.

Assumptions: None

Side Effects: None

Availability: All versions of the TI-89 / TI-92 Plus.

TI-89 / TI-92 Plus Differences: None

See Also: **DerefSym, FolderFind, SymFindHome**

Example:

```
void DISP( char *str, WORD v ) {
   char buf[255];
   sprintf(buf, str, v );
   DlgNotice( NULL, buf );
}

BYTE mainFold[] = {0,'m','a','i','n',0};
HSYM hsym;

/* All folders, including main, are in the HOME folder. MAIN can never be deleted so
   hsym will never be NULL in this example.
*/
if (hsym = SymFindHome( mainFold+5 ))
   DISP( "%d symbols in main", FolderCount( DerefSym( hsym ) ) );
```

FolderCur

Declaration: BOOL **FolderCur** (const BYTE **FolderName*, BOOL *CheckGraphDirty*)

Category(ies): Symbol Table Utilities

Description: Make the given folder the default folder. Return TRUE if folder is valid, FALSE otherwise. *CheckGraphDirty* should always be set to TRUE.

Inputs:

FolderName	—	EStackIndex of folder name.
CheckGraphDirty	—	Set to TRUE (if FALSE, the current graph can become invalid).

Outputs:

TRUE	—	If successful.
FALSE	—	Invalid folder name.

Assumptions: **push_setfold** is similar except that **push_setfold** throws any errors and returns the current previous default folder on the estack as well as updating the status line with **ST_folder**.

> **NOTE:** Caller must call **ST_folder** to update the status line unless this is a temporary folder.

Side Effects: Status line is NOT updated to reflect current folder.

Availability: All versions of the TI-89 / TI-92 Plus.

TI-89 / TI-92 Plus Differences: None

See Also: **push_setfold, push_getfold, FolderGetCur**

Example:

```
WORD TokenizeFoldName( const char *strFileName, BYTE *TokFName );
DWORD NoCallBack( WORD DlgId, DWORD Value ) { return TRUE; }

/* Prompt the user for a folder name and set that to the current default folder.
   If the user enters nothing or an invalid name, reprompt this time with the
   current folder. Return TRUE if valid name entered, FALSE otherwise
*/
BOOL tFolderGetSet( void )
{  HANDLE dH = H_NULL;
   BYTE szBuf[SYM_LEN+1], TokFName[MAX_SYM_LEN], *StrPtr;
```

(continued)

FolderCur *(continued)*

Example:
(continued)

```
   if ((dH = DialogNew(0, 0, NoCallBack)) != H_NULL) {
      if (DialogAdd(dH,0,8,16,D_EDIT_FIELD,"Folder:",0,SYM_LEN,SYM_LEN) &&
          DialogAdd(dH,0,0,0,D_HEADER,"CHANGE CUR FOLDER",PDB_OK,PDB_CANCEL)) {
         memset( szBuf, 0, SYM_LEN );
redo:
         if (KB_ENTER == DialogDo( dH,-1,-1, (char *) szBuf, NULL)) {
            StrPtr = FirstNonblank((BYTE *) szBuf);
            if (*StrPtr == '\0')  {
redo2:
               FolderGetCur( szBuf );
               goto redo;
            }
            if (FS_OK == TokenizeFoldName((char *) szBuf, TokFName))
               if (FolderCur(TokNameRight(TokFName),TRUE)) {
                  ST_folder( (char *) StrPtr );
                  DlgNotice( "FOLDER CHANGED TO", (char *) szBuf );
                  DialogFree( dH );
                  return( TRUE );
               }
            DlgNotice( "BAD NAME OR FOLDER NOT FOUND", (char *) szBuf );
            goto redo2;
         }
      }
   }
   if (dH)
      DialogFree( dH );
   return(FALSE);
}

/* This routine is the same as TokenizeName in the file system except it does not
   fully-qualify names (add folder name if not there) so it can be used to tokenize
   folder names. */
WORD TokenizeFoldName( const char *strFileName, BYTE *TokFName )
{ EStackIndex oldTop;

   if (oldTop = TokenizeSymName( (BYTE *) strFileName,0))  {
      /* copy name from estack to buffer (may include trash at front) */
      memcpy( TokFName, TokNameLeft(top_estack), MAX_SYM_LEN );
      top_estack = oldTop; /* restore estack top */
      return FS_OK;
   }  else
      return FS_BAD_NAME;
}
```

FolderDel

Declaration: BOOL **FolderDel** (const BYTE * *FolderName*, BOOL *SymsOnly*)

Category(ies): Symbol Table Utilities (low-level)

Description: Delete the given *FolderName* from the home folder. If a symbol in the folder has a value (handle != NULL) then the memory for the value is released. Return TRUE if it was deleted, FALSE if it was not found or was not a folder. If *FolderName* was the current folder then the HOME folder becomes the current folder. This routine can be used to delete all of the symbols in MAIN but not MAIN itself.

If *SymsOnly* is TRUE then only the symbols within the folder (and their values) are deleted, the folder name is kept in the symbol table.

> **NOTE:** This routine will delete all symbols in the folder even if they are locked, in-use, or archived!

Inputs: *FolderName* — Pointer to tokenized folder name to delete.

SymsOnly — TRUE to just delete symbols but leave folder.

Outputs: TRUE if folder deleted, FALSE if the folder was not found.

Assumptions: None

Side Effects: None

Availability: All versions of the TI‑89 / TI‑92 Plus.

TI‑89 / TI‑92 Plus Differences: None

See Also: **FolderAdd**

Example: See **FolderAdd**.

FolderFind

Declaration:	WORD **FolderFind** (const BYTE * *FolderName*)		
Category(ies):	Symbol Table Utilities (low-level)		
Description:	Find the given folder and return the folder type.		
Inputs:	*FolderName*	—	Pointer to tokenized folder name to find.
Outputs:	FL_MAIN	—	Home folder.
	FL_OTHER	—	Other folder.
	FL_NOTFOUND	—	Not found.
	FL_NOTFOLDER	—	FolderName is already used as nonfolder.
Assumptions:	None		
Side Effects:	None		
Availability:	All versions of the TI‑89 / TI‑92 Plus.		
TI‑89 / TI‑92 Plus Differences:	None		
See Also:	**FolderAdd, FolderCount, FolderRename**		
Example:	See **FolderAdd**.		

FolderGetCur

Declaration: void **FolderGetCur** (BYTE * *FolderName*)

Category(ies): Symbol Table Utilities

Description: Return the current default folder name (SYM_LEN + 1 chars) as a zero byte terminated ASCII string.

Inputs: *FolderName* — Address of BYTE buffer of at least SYM_LEN + 1 chars.

Outputs: *FolderName* — Current default folder name returned here.

Assumptions: **push_getfold** returns the same name only it returns it as a STR_DATA_TAG value on the estack.

Side Effects: None

Availability: All versions of the TI-89 / TI-92 Plus.

TI-89 / TI-92 Plus Differences: None

See Also: **push_setfold, push_getfold, FolderCur**

Example: See **FolderCur**.

FolderOp

Declaration: BOOL FolderOp (const BYTE * *FolderName*, WORD *Operation*)

Category(ies): Symbol Table Utilities (low level)

Description: Lock or unlock folders.

Inputs: *FolderName* — Tokenized name of folder to operate on.

Operation — FL_UNLOCK — Unlock the given folder.

FL_LOCK — Lock the given folder.

If *Operation* is OR'd with FL_ALL then all folders are either locked or unlocked with *FolderName* being ignored in that case.

Outputs: TRUE if operation successful, FALSE if *FolderName* not found.

Assumptions: None

Side Effects: None

Availability: All versions of the TI‑89 / TI‑92 Plus.

TI‑89 / TI‑92 Plus Differences: None

See Also: None

Example: See **SymFindFirst**.

That example locks the HOME folder (the folder that contains all of the folders) since it uses **SymFindFirst** that returns direct pointers to the symbol table. It also allocates memory from the heap which could possibly move the symbol table. Therefore, it locks the HOME folder, walks through all of the folders in the HOME folder, and then unlocks the HOME folder.

FolderRename

Declaration: BOOL **FolderRename** (const BYTE * *OldName*, const BYTE * *NewName*)

Category(ies): Symbol Table Utilities (low-level)

Description: Rename a folder. This is the low-level folder rename routine. In general, the routine **cmd_rename** should be used since this routine does not check for reserved names.

Inputs: *OldName* — Pointer to tokenized original folder name.

NewName — Pointer to tokenized new folder name.

Outputs: Returns TRUE if successful; *OldName* was found and was a folder; *NewName* was not found, otherwise FALSE.

Will throw an ER_RESERVED error if renaming to or from MAIN.

Will throw a ER_VAR_IN_USE if any variable in the given folder is in-use.

Assumptions: None

Side Effects: None

Availability: All versions of the TI‑89 / TI‑92 Plus.

TI‑89 / TI‑92 Plus Differences: None

See Also: **cmd_rename**

Example:

```
BYTE foldName[] =  {0,'t','m','p','1',0};
BYTE newFoldName[] =  {0,'t','e','m','p','1',0};
FolderAdd( foldName+5 );
.
.
.
FolderRename( foldName+5, newFoldName+6 );
```

HSymDel

Declaration: void **HSymDel** (HSYM *hsym*)

Category(ies): Symbol Table Utilities

Description: Delete a variable using an HSYM. If the symbol has a value, its memory is also released.

Inputs: *hsym* — HSYM of a variable to delete.

Outputs: Nothing is returned since it is assumed *hsym* is a valid HSYM; may throw the following errors:

ER_RESERVED — Can not delete MAIN folder.

ER_VAR_IN_USE — Variable is in use.

ER_LOCKED — Variable is archived (available in 2.04 and above).

Assumptions: The caller must be sure the symbol table has not been changed since the HSYM was obtained.

> **NOTE:** If called to delete a folder than that folder MUST be empty! Also do not use to delete twin or archived variables.

Side Effects: As with all routines that modify the symbol table, this routine invalidates any other existing HSYMs.

Availability: All versions of the TI-89 / TI-92 Plus.

TI-89 / TI-92 Plus Differences: None

See Also: **cmd_delvar, cmd_delfold, SymDel, FolderDel**

(continued)

HSymDel *(continued)*

Example:

```
/* Clear the single letter variables  (A . . . Z) in the current folder and
   return the number that could not be deleted.
*/
short clearAtoZ( void )
{  HSYM hSym;
   SYM_ENTRY *pSym;
   short nLeft;
   BYTE symbol[] = {0, 0, 0};

   for (nLeft=26, symbol[1] = 'a'; symbol[1] <= 'z'; symbol[1]++) {
      if (H_NULL == (hSym = SymFind(symbol+2)))  /* lookup var in current folder */
         nLeft--; /* not found if HSYM is NULL */
      else {
         pSym = DerefSym(hSym);  /* HSymDel does not check LOCKED, IN-USE flags */
         if ((pSym->Flags & (SF_LOCK|SF_INUSE)) == 0) {
            TRY
               HSymDel(hSym); /* delete it */
               nLeft--;
            ONERR
            ENDTRY
        }
      }
   }
   return nLeft;
}
```

MakeHsym

Declaration: HSYM **MakeHsym** (HANDLE *fHandle*, SYM_ENTRY * *SymEntry*)

Category(ies): Symbol Table (low-level)

Description: Given a handle to a folder and a SYM_ENTRY pointer to a symbol in that folder create an HSYM from both values. An HSYM is basically a combination of the folder's handle and the offset of a symbol into that folder. HSYMs are valid until a symbol is added or removed from the folder they belong to. Dereferencing an HSYM with **DerefSym** produces a direct pointer to the symbol table which is valid until heap compression is done.

Inputs: *fHandle* — Handle of folder.

SymEntry — SYM_ENTRY pointer to symbol.

Outputs: HSYM representing the given symbol.

Assumptions: None

Side Effects: HSYMs are valid until a symbol is added or removed from the folder they belong to.

Availability: All versions of the TI-89 / TI-92 Plus.

TI-89 / TI-92 Plus Differences: None

See Also: **DerefSym**

Example: In this example, SymPtr (a SYM_ENTRY pointer to a specific symbol) must be maintained. Heap compression would cause the pointer to be invalid since it is a direct pointer into memory. So the SymPtr is converted to an HSYM with **MakeHsym** (along with the HANDLE of the folder that the symbol belongs to). After the code that may cause heap compression is executed, the HSYM is converted back into a SYM_ENTRY pointer with **DerefSym**.

```
HANDLE folderHandle;
SYM_ENTRY *SymPtr;
HSYM hsym;

hsym = MakeHsym( folderHandle, SymPtr );
. . . Do something that may cause heap compression . . .
SymPtr = DerefSym( hsym );
```

push_getfold

Declaration: void **push_getfold** (void)

Category(ies): Symbol Table Utilities

Description: Return the current default folder on the estack as a TI-BASIC string (STR_DATA_TAG). This is the TI-BASIC function getFold.

Inputs: None

Outputs: Current default folder on estack.

Assumptions: None

Side Effects: None

Availability: On AMS 1.05 and higher.

TI-89 / TI-92 Plus Differences: None

See Also: **push_setfold, FolderGetCur**

Example: See **push_setfold**.

push_setfold

Declaration: void **push_setfold** (EStackIndex *foldName*)

Category(ies): Symbol Table Utilities

Description: Make the given folder the default folder. This is the TI-BASIC function setFold. Returns the current previous default folder on the estack as a string (STR_DATA_TAG). Updates the status line with the new folder name.

Inputs: *foldName* — EStackIndex of folder name.

Outputs: May throw these errors:

ER_ILLEGAL_IN_FUNC	—	The default folder cannot be changed inside a function.
ER_INVALID_FOLDER_NAME, INVALID_PATHNAME_ERROR	—	Invalid folder name.

Assumptions: None

Side Effects: Unlike **FolderCur**, this routine updates the status line.

Availability: On AMS 1.05 and higher.

TI-89 / TI-92 Plus Differences: None

See Also: **push_getfold, FolderCur**

Example: Compare this example to the example for **FolderCur**. This example (1) uses a static dialog box, listed at the end, (2) uses XR_FolderC which is a system string reference and will be automatically translated by the current localizer, and (3) uses XFLAGS with XF_ALLOW_VARLINK set so the user can pop-up VAR-LINK within the dialog box and paste folder names.

(continued)

push_setfold *(continued)*

Example:
(continued)

```
/* Prompt the user for a folder name and set that to the current default folder.
   If the user enters nothing or an invalid name, reprompt this time with the
   current folder. Return TRUE if valid name entered, FALSE otherwise.
*/
{
   BYTE szBuf[SYM_LEN+2], *ptr;
   EStackIndex oldTop = top_estack;
   memset( szBuf, 0, SYM_LEN+1 );
redo:
   TRY
      if (KB_ENTER == Dialog( &dGetFold,-1,-1, (char *) szBuf, NULL)) {
         ptr = FirstNonblank((BYTE *) szBuf);
         if (*ptr == '\0')  {
redo2:
            push_getfold();
            /* Pushed on estack: 0, cur_fold, 0, STR_DATA_TAG. So we start at the
               last char of cur_fold and find the starting char (going from high to
               low memory) */
            ptr = top_estack - 2;
            while (*ptr)
               ptr--;
            strcpy( (char *) szBuf, (char *) ptr+1 );
            goto redo;
         }
         if (TokenizeSymName( (BYTE *) szBuf,0))  {
            TRY
               push_setfold( top_estack  );
            ONERR
               goto redo3;
            ENDTRY
            DlgNotice( "FOLDER CHANGED TO", (char *) szBuf );
            top_estack = oldTop;
            return(TRUE);
         }
redo3:
         DlgNotice( "BAD NAME OR FOLDER NOT FOUND", (char *) szBuf );
         goto redo2;
      }
   ONERR
   ENDTRY
   top_estack = oldTop;
   return(FALSE);
}
DIALOG dGetFold, 0, 0, NoCallBack {
   EDIT,   {0, 8, 15}, XR_FolderC, 0, 8, 9
   HEADER, {0, 0, 0}, "SET FOLDER", PDB_OK, PDB_CANCEL
   XFLAGS, {0, 0, 0}, XF_ALLOW_VARLINK | XF_VARLINK_SELECT_ONLY, 0, 0, 0
}
```

ResetSymFlags

Declaration: void **ResetSymFlags** (WORD *mask*)

Category(ies): Symbol Table (low-level)

Description: Reset flags on all variables in the symbol table using *mask*.

Inputs: *mask* — All flags matching mask will be reset.

See section **13.3 Managing Variables** for a list of the symbol flags.

Outputs: None

Assumptions: None

Side Effects: None

Availability: All versions of the TI-89 / TI-92 Plus.

TI-89 / TI-92 Plus Differences: None

See Also: **SymFindFirst, SymFindNext**

Example: This is the core code in **ResetSymFlags**. It walks through every symbol in the symbol table resetting the flags specified. The real routine also resets some internal variables that are not in the symbol table.

```
SYM_ENTRY *symp;

mask = ~mask;
for(symp = SymFindFirst(NULL,FO_RECURSE); symp != NULL; symp = SymFindNext())
   symp->Flags &= mask;
```

SetOK

Declaration: void **SetOK** (BOOL *Result*)

Category(ies): Symbol Table Utilities

Description: Set the global user variable, OK, to the numeric value one if *Result* is TRUE (nonzero) or to zero if *Result* is FALSE (zero).

Inputs: *Result* — TRUE (nonzero) or FALSE (zero).

Outputs: None

Assumptions: None

Side Effects: None

Availability: All versions of the TI‑89 / TI‑92 Plus.

TI‑89 / TI‑92 Plus Differences: None

See Also: **VarStore**

(continued)

SetOK *(continued)*

Example: This example sets the global OK variable to 1 if the graph is blank or 0 if it is not blank (axes not included).

```
#include "tiams.h"

void main(void)
{  Access_AMS_Global_Variables;
   WORD SaveSize;
   HANDLE hBitmap;
   BYTE *Ptr;
   WIN_RECT wr;
   BOOL flag;
   WINDOW *winPtr;

   wr = *MakeWinRect(0,0,gr_active->xmaxpix,gr_active->ymaxpix);
   winPtr = gr_active->grwinp;
   if ((SaveSize = WinBitmapSize( winPtr, &wr )) > 0) {
      if (hBitmap = HeapAlloc(SaveSize)) {
         Ptr = HeapDeref(hBitmap);
         flag = WinDupStat( winPtr, TRUE );
         WinBitmapGet( winPtr, &wr, (BITMAP *) Ptr );
         WinDupStat( winPtr, flag );
         SaveSize -= BITMAP_HDR_SIZE;
         Ptr += BITMAP_HDR_SIZE;
         flag = TRUE;
         while (SaveSize--) {
            if (*Ptr++) {
               flag = FALSE;
               break;
            }
         }
         HeapFree( hBitmap );
         SetOK( flag );
      }  else
         ER_THROW( ER_MEMORY );
   }  else
      ER_THROW( ER_RESERVED );
}
```

SymAdd

Declaration: HSYM **SymAdd** (const BYTE * *SymName*)

Category(ies): Symbol Table (low-level)

Description: Add the given symbol name to the symbol table and return an HSYM to the new symbol table entry. If the symbol already exists and it has a value, that value will be deleted (unless it is a folder name, then NULL is returned).

Inputs: *SymName* — Pointer to tokenized symbol name.

Outputs: HSYM of newly created symbol or zero if not enough memory.

May throw an ER_LOCKED error if the symbol already exists and is locked.

Assumptions: **VarStore** or the file system are normally used to create symbols.

Side Effects: May cause heap compression.

> **NOTE:** This routine does not check for reserved symbols and so caution must be used when using this routine.

Availability: All versions of the TI-89 / TI-92 Plus.

TI-89 / TI-92 Plus Differences: None

See Also: **SymDel, VarStore, FOpen**

Example:

```
Access_AMS_Global_Variables;
BYTE foldName[] =  {0,'t','m','p','1',0};
BYTE symName[] =  {0,'s','y','m','1',0};
BCD16 fNum = 1.234;
HSYM hsym1;

if (FolderFind(foldName+5) == FL_NOTFOUND)  {
   if( !FolderAdd( foldName+5 ))
      ER_throw( ER_MEMORY );
}
FolderCur( foldName+ 5, TRUE );
if (hsym1 = SymAdd( symName+5 ))  {
   push_Float( fNum );
   VarStore( symName+5, STOF_ESI, 0, top_estack );  /* give it a value */
   /* HSYMs better match! */
   if (hsym1 != SymFind(symName+5))
      ER_THROW( FIRST_INTERNAL_ERR );
   handleVarLinkKey(SDT_ALL); /* show new symbol */
   /* hsym1 is now invalid since user can add/delete variables in VAR-LINK */
   if (!SymDel( symName+5 ))
      DlgNotice( NULL, "You deleted tmp1\\sym1 in VAR-LINK" );
}
```

SymDel

Declaration: BOOL **SymDel** (const BYTE * *SymName*)

Category(ies): Symbol Table (low-level)

Description: Delete the given symbol, return TRUE if deleted or FALSE if not found. If the symbol has a value, its memory is also released. Note that this routine will delete variables even if they are locked or in-use!

Inputs: *SymName* — Pointer to tokenized symbol name.

Outputs: TRUE if deleted or FALSE if not found.

Assumptions:

> **NOTE:** Do not call to delete a folder, twin, or archived variable.

Side Effects: None

Availability: All versions of the TI‑89 / TI‑92 Plus.

TI‑89 / TI‑92 Plus Differences: None

See Also: **SymAdd, cmd_delvar**

Example: See **SymAdd**.

SymFind

Declaration: HSYM **SymFind** (const BYTE * *SymName*)

Category(ies): Symbol Table Utilities (low-level)

Description: Search for the symbol *SymName* and return the HSYM if found or zero if not found. Note that most reserved symbols are not stored in the symbol table. If *SymName* does not contain a folder then the current folder is searched.

Inputs: *SymName* — Tokenized symbol to search for.

Outputs: HSYM of symbol or zero if not found.

Assumptions: None

Side Effects: None

Availability: All versions of the TI-89 / TI-92 Plus.

TI-89 / TI-92 Plus Differences: None

See Also: **SymAdd, SymDel**

Example: See **SymAdd**.

SymFindFirst

Declaration: SYM_ENTRY * **SymFindFirst** (const BYTE * *FolderName*, WORD *Options*)

Category(ies): Symbol Table Utilities (low level)

Description: Find the first symbol in the given folder and setup internal pointers so that **SymFindNext** and **SymFindPrev** may be called to traverse the folder.

> **NOTE:** Since this routine and subsequent calls to **SymFindNext** and **SymFindPrev** return direct pointers to the symbol table, anything that would cause heap compression will cause the results to be invalid or may lock-up the system.

Inputs:

FolderName — Tokenized name of folder to traverse.

Options — The following flags may be set.

FO_RECURSE
: The symbol table is searched recursively (i.e., through the HOME symbol table and down each folder), therefore, *FolderName* is ignored. If and only if FO_RECURSE is set then **SymFindFoldername** will return the name of the folder for the symbol just returned.

FO_NOTEMPS
: Used with FO_RECURSE to skip over any temporary folders.

FO_CKTWINS
: **SymFindNext** and **SymFindPrev** normally will skip Flash entries corresponding to a twin entry. Setting this will cause twin entries to be returned.

FO_SKIP_COLLAPSE
: Used with FO_RECURSE to skip variables in a folder with the SF_COLLAPSE bit set.

Outputs: SYM_ENTRY pointer of first symbol found or NULL if none found.

Assumptions: None

Side Effects: Heap compression will invalidate the pointers returned necessitating another call to **SymFindFirst**.

Availability: All versions of the TI-89 / TI-92 Plus.

(continued)

SymFindFirst *(continued)*

TI-89 / TI-92 Plus Differences: None

See Also: **SymFindNext, SymFindPrev, SymFindFoldername, FolderOp**

Example: This example walks through the HOME folder (not the folder 'main') which is the folder that contains all of the folders. It uses **SymFindFirst** and **SymFindNext** to build a pop-up of all of the folders in the system. This is basically the code for the **VarCreateFolderPopup** routine. See **FolderOp** for a description of why the HOME folder is locked during this operation.

```
HANDLE hPopup;
SYM_ENTRY *se;
BYTE HomeFolder[] =  {0,127,0};

if ((hPopup = PopupNew( NULL, 0 )) == H_NULL)
   return;
FolderOp( HomeFolder+2, FL_LOCK );
se = SymFindFirst( HomeFolder+2, FO_NONE );
while (se != NULL) {
   if ((se->Flags & SF_FOLDER) && !IsTempNameChar(se->Name[0])) {
      if (!PopupAddText( hPopup, EOF, (char *) se->Name, 0 )) {
         FolderOp( HomeFolder+2, FL_UNLOCK );
         PopupFree( hPopup );
         return;
      }
   }
   se = SymFindNext();
}
FolderOp( HomeFolder+2, FL_UNLOCK );
PopupDo( hPopup, -1,-1, 0 );
PopupFree( hPopup );
```

SymFindFoldername

Declaration: BYTE * **SymFindFoldername** (void)

Category(ies): Symbol Table Utilities (low-level)

Description: If calling **SymFindFirst** / **SymFindNext** / **SymFindPrev** to recurse through the symbol table, then calling this routine will get the name of the folder for the symbol just returned. Do not call this routine if not using FO_RECURSE!

Inputs: None

Outputs: Pointer to folder name.

Assumptions: None

Side Effects: None

Availability: All versions of the TI-89 / TI-92 Plus.

TI-89 / TI-92 Plus Differences: None

See Also: **SymFindFirst, SymFindNext, SymFindPrev**

Example: This is the code that is executed when a **HeapWalk**(H_WALK_SYM) is done. It first saves the symbol table private global structure (SymPG_S), since it is a static global, and executing the **SymWalk** function, would modify it, possibly corrupting data used by a calling app. It then walks the entire symbol table, printing every symbol and folder using **SymFindFoldername** to print the folder of each symbol found. When done it restores SymPG_S. The purpose of the FINALLY, ENDFINAL section is to insure that SymPG_S is restored whether an error is generated by **LIO_SendData** or not. The PRINTF macro redirects the output of **SymWalk** to the link port. A PC may capture that output with a gray-link cable and a terminal emulation program (like HyperTerminal).

(continued)

SymFindFoldername *(continued)*

Example:
(continued)

```
#define PRINTF(s) LIO_SendData((BYTE *)s, strlen((char *)s))

void SymWalk( void )
{  Access_AMS_Global_Variables;
   SYM_ENTRY *SymPtr;
   BYTE buf[256];
   SymPG_S saveSPGS;

TRY
   memcpy( &saveSPGS, pSymPG, sizeof(SymPG_S) );
   if ((SymPtr = SymFindFirst(NULL,FO_RECURSE)) != NULL) {
      PRINTF("\r\nName/Flags/hVal (dec)\r\n");
      do {
         if (SymPtr->Flags & SF_FOLDER)
            sprintf((char *)buf, "FOLDER: %-8s   %04X  %d\r\n", SymPtr->Name,
                    SymPtr->Flags, SymPtr->hVal );
         else
            sprintf((char *) buf, "%8s\\%-8s  %04X  %d\r\n", SymFindFoldername(),
                    SymPtr->Name, SymPtr->Flags, SymPtr->hVal );
         PRINTF( buf );
         SymPtr = SymFindNext();
      } while(SymPtr != NULL);
   }
FINALLY
   memcpy( pSymPG, &saveSPGS, sizeof(SymPG_S) );
ENDFINAL
}
```

SymFindHome

Declaration: HSYM **SymFindHome** (const BYTE * *SymName*)

Category(ies): Symbol Table (low-level)

Description: Search for the symbol *SymName* in the HOME folder and return the HSYM if found or zero if not found. Note that the HOME folder contains all of the folders in the system and normally does not contain anything else.

Inputs: *SymName* — Pointer to tokenized name to look up (should be a folder name).

Outputs: HSYM of symbol if found, zero if not found or invalid name.

Assumptions: None

Side Effects: None

Availability: All versions of the TI‑89 / TI‑92 Plus.

TI‑89 / TI‑92 Plus Differences: None

See Also: **SymFind, SymFindMain**

Example: See **FolderCount**.

SymFindMain

Declaration: HSYM SymFindMain (const BYTE * *SymName*)

Category(ies): Symbol Table (low-level)

Description: Search for the symbol *SymName* in the MAIN folder and return the HSYM if found or zero if not found. Note that some variables such as the system reserved equations (Y1 . . . Y99, RegEq, . . .) are all stored in the MAIN folder and cannot be moved to a different folder.

Inputs: *SymName* — Pointer to tokenized name to look up.

Outputs: HSYM of symbol if found, zero if not found or invalid name.

Assumptions: None

Side Effects: None

Availability: All versions of the TI‑89 / TI‑92 Plus.

TI‑89 / TI‑92 Plus Differences: None

See Also: **SymFind, SymFindHome**

(continued)

SymFindMain *(continued)*

Example: This example walks through all of the Yn functions and displays them as text in a dialog box.

```
int i, Skip;
BYTE buf[6];
HANDLE hVal, hText;
HSYM hsym;
SYM_ENTRY *SymPtr;
EStackIndex esi;

for (i=1; i<=99; i++) {
   Skip = sprintf( (char *) buf, "%cy%d", 0, i );
   if (hsym = SymFindMain( (BYTE *) (buf + Skip) )) {
      SymPtr = DerefSym( hsym );
      if (hVal = SymPtr->hVal) {
         esi = HToESI( hVal );
         /* Normally would check the tag *esi but Yn are always functions and are
            always tokenized so do not need to check the FF_PARSE flag *(esi-1)
            either. */
         HeapLock( hVal );
         TRY
            hText = display_statements( GetFuncPrgmBodyPtr( esi ), FALSE, TRUE );
         FINALLY
            HeapUnlock( hVal ); /* This always gets executed even if an error is
                                   thrown. Any errors will be passed on up to our
                                   caller. */
         ENDFINAL
         HeapLock( hText );
         DlgNotice( (char *) buf+1, (char *) (HeapDeref(hText)) );
         HeapFree( hText );
      }
   }
}
```

SymFindNext

Declaration: SYM_ENTRY * **SymFindNext** (void)

Category(ies): Symbol Table Utilities (low-level)

Description: Return the next symbol as specified by **SymFindFirst**.

Inputs: None

Outputs: SYM_ENTRY pointer to next symbol.

Assumptions: This is a direct pointer to the symbol table, heap compression will invalidate this pointer.

Side Effects: None

Availability: All versions of the TI‑89 / TI‑92 Plus.

TI‑89 / TI‑92 Plus Differences: None

See Also: **SymFindFirst, SymFindPrev**

Example: See **SymFindFirst**.

SymFindPrev

Declaration: SYM_ENTRY * **SymFindPrev** (void)

Category(ies): Symbol Table Utilities (low-level)

Description: Return the previous symbol as specified by **SymFindFirst**.

Inputs: None

Outputs: SYM_ENTRY pointer to previous symbol.

Assumptions: This is a direct pointer to the symbol table, heap compression will invalidate this pointer.

Side Effects: None

Availability: All versions of the TI‑89 / TI‑92 Plus.

TI‑89 / TI‑92 Plus Differences: None

See Also: **SymFindFirst, SymFindNext**

Example: Normally the only routine used with **SymFindFirst** is **SymFindNext**. This routine allows an app to traverse the symbol table in both directions. VAR-LINK uses this routine to present a scrollable list of all symbols without having to maintain a separate in-memory list.

VarCreateFolderPopup

Declaration: HANDLE **VarCreateFolderPopup** (WORD * *DefIndex*, WORD *Flags*)

Category(ies): Symbol Table Utilities, Menus

Description: Create a dynamic pop-up with a list of all of the current folders in the system and return a handle to it.

Inputs: *Flags* — VCFP_ALL
Include "All" as the first option in the list. See VAR-LINK F2 (View) for an example which is included below.

VCFP_SKIP_CURDIR
Do not include the current directory.

Outputs: HANDLE of dynamic pop-up of all folders in the system, H_NULL if not enough memory.

DefIndex — Index of current default folder.

Assumptions: Caller must eventually free the handle returned.

Side Effects: May cause heap compression.

Availability: All versions of the TI-89 / TI-92 Plus.

TI-89 / TI-92 Plus Differences: None

See Also: See **SymFindFirst** for the basic source to this routine (not including *Flags*).

Example: This example is basically the code from the VAR-LINK VIEW (F2) key. It creates a dialog with two dynamic pop-ups (DYNPOPUPs in the resource file). One (hFolderPopup) of which uses **VarCreateFolderPopup** to list all of the folders in the system including "All" so that the user can view all of the folders in the system at one time.

(continued)

VarCreateFolderPopup *(continued)*

```
#define SDT_APP 16
enum ViewPopup { VL_VIEW_VARS=1, VL_VIEW_APPS, VL_VIEW_SYS };
enum ViewOpts { VLO_VIEW, VLO_FOLDER, VLO_VARTYPE };
WORD VL_ViewOpts[3];
HANDLE hFolderPopup, hViewPopup;
WORD SymFindType = SDT_ALL;
WORD SymFindFlags = 0;
BYTE szVLfolder[SYM_LEN+2];

/* Private routine for VL_View to return dynamic pop-up handle */
HANDLE VL_GetFolder(WORD dii)
{
   if (1 == dii)
      return hFolderPopup;
   else
      return hViewPopup;
}

/* VL_View callback */
WORD VL_ViewCB( WORD DlgId, DWORD Value )
{
   if (DB_QACTIVE == DlgId) {
      if (VLO_FOLDER == Value || VLO_VARTYPE == Value)
         return (1 == VL_ViewOpts[VLO_VIEW]);
      else
         return TRUE;
   }  else if (VLO_VIEW == DlgId)
      return DB_REDRAW_AND_CONTINUE;
   return TRUE;
}

/* VarLink VIEW (F2) dialog box code */
void VL_View( void )
{  WORD wVal, RestoreCurFolder;

VL_ViewOpts[VLO_VARTYPE] = SDT_ALL + 1;
if (SDT_APP == SymFindType)
   VL_ViewOpts[VLO_VIEW] = VL_VIEW_APPS;
else if (SDT_SYS == SymFindType)
   VL_ViewOpts[VLO_VIEW] = VL_VIEW_SYS;
else {
   VL_ViewOpts[VLO_VIEW] = VL_VIEW_VARS;
   VL_ViewOpts[VLO_VARTYPE] = SymFindType + 1;
}
```

VarCreateFolderPopup *(continued)*

```
VL_ViewOpts[VLO_FOLDER] = 1;
if ((hFolderPopup = VarCreateFolderPopup(&VL_ViewOpts[VLO_FOLDER], VCFP_ALL))
!= H_NULL)
{
   if (hViewPopup = PopupNew(NULL, 0)) {
      PopupAddText(hViewPopup,-1,XR_stringPtr(XR_Vars), VL_VIEW_VARS );
      PopupAddText(hViewPopup,-1,XR_stringPtr(XR_vtFlashApp), VL_VIEW_APPS );
      PopupAddText(hViewPopup,-1,XR_stringPtr(XR_vtSystem), VL_VIEW_SYS );
      if (MenuFlags(hViewPopup) & MF_ERROR)
         goto VM1;
   }  else
      goto VM1;
   wVal = Dialog(&VarLinkView, -1, -1, NULL, VL_ViewOpts);
   if (wVal == KB_ENTER) {
      if (VL_VIEW_SYS == VL_ViewOpts[VLO_VIEW])
         SymFindType = SDT_SYS;
      else if (VL_VIEW_APPS == VL_ViewOpts[VLO_VIEW])
         SymFindType = SDT_APP;
      else {
         SymFindType = VL_ViewOpts[VLO_VARTYPE] - 1;
         strcpy( (char *) szVLfolder+1, (char *) PopupText( hFolderPopup,
         VL_ViewOpts[VLO_FOLDER]) );
         if (strcmp((char *) szVLfolder+1, XR_stringPtr(XR_All)) == 0)
            SymFindFlags = FO_RECURSE | FO_CKTWINS;
         else
            SymFindFlags = FO_NONE | FO_CKTWINS;
         szVLfolder[0] = '\0';
      }
   }  else if (wVal == DB_MEMFULL)
      goto VM1;
} else
VM1:
      ERD_dialog( ER_MEMORY, FALSE );
   if (hFolderPopup)
      PopupFree(hFolderPopup);
   if (hViewPopup)
      PopupFree(hViewPopup);
}
```

VarCreateFolderPopup *(continued)*

```
// PopupVarType
POPUP PopupVarType, RC_NO_IDS, 0 {
   XR_All,                                  SDT_ALL+1
   XR_vtExpr,                               SDT_EXPR+1
   XR_vtList,                               SDT_LIST+1
   XR_vtMatrix,                             SDT_MAT+1
   XR_vtFunction,                           SDT_FUNC+1
   XR_vtProgram,                            SDT_PRGM+1
   XR_vtPicture,                            SDT_PIC+1
   XR_vtString,                             SDT_STR+1
   XR_vtText,                               SDT_TEXT+1
   XR_vtGDB,                                SDT_GDB+1
   XR_vtData,                               SDT_DATA+1
   XR_vtFigure,                             SDT_FIG+1
   XR_vtMacro,                              SDT_MAC+1
   XR_vtAsm,                                SDT_ASM+1
   XR_vtOther,                              SDT_OTH+1
}

// VarLinkView
DIALOG VarLinkView, 0, 0, VL_ViewCB {
   DYNPOPUP,  {DF_TAB_ELLIPSES, 8, 15}, XR_View, VL_GetFolder, 0
   DYNPOPUP,  {DF_SCREEN_SAVE|DF_TAB_ELLIPSES, 8, 28}, XR_Folder, VL_GetFolder, 1
   POPUP,   {DF_TAB_ELLIPSES, 8, 41}, XR_VarType, PopupVarType, 2
   HEADER, {0,0,0}, XR_VARLINKVIEW, PDB_OK, PDB_CANCEL
}
```

VarRecall

Declaration: HSYM **VarRecall** (BYTE * *Var*, RECALL_FLAGS *Flag*)

Category(ies): Symbol Table Utilities, Variables

Description: Recalls a variable, returning its HSYM or NULL if not found. Note that this routine handles system variables even if they are not in the symbol table.

> **NOTE:** See the "Storing and Retrieving variable data" section in the Memory Management chapter for further details on **VarRecall**.

Inputs:

Var — EStackIndex of variable name to look-up.

Flags — *VR_NO_SYS_VARS* Throw an error if the variable to look-up is a system variable.

Outputs: Returns the HSYM of the variable to look-up, NULL if the variable is not found.

Assumptions: None

Side Effects: Some system variables are not in the symbol table and so **VarRecall** returns a dummy HSYM which is shared by all such variables. Thus the next call to **VarRecall** for such a variable will return the same HSYM but with a different value (and the previous HSYM will be invalid). So if **VarRecall** is used for system variables it is best to keep a copy of the value pointed to by the HSYM if it is needed.

Availability: All versions of the TI‑89 / TI‑92 Plus.

TI‑89 / TI‑92 Plus Differences: None

See Also: **VarStore, TokenizeSymName**

(continued)

VarRecall *(continued)*

Example: See **GetDataType** for another example.

```
/* This is the FILE FDelete function. It uses VarRecall to look-up the file to be
   deleted. If it is not found or is locked, in use, a folder, or in the archive an
   error is returned. Otherwise HSymDel is used to delete the variable.
*/
WORD FDelete( const char *fileName ) {
   HSYM hSym;
   BYTE TokFName[MAX_SYM_LEN];

   if (FS_OK != TokenizeName(fileName, TokFName))
      return( FS_BAD_NAME );
    if (hSym = VarRecall(TokNameRight(TokFName), VR_NO_SYS_VARS ))
      if (!(DerefSym(hSym)->Flags & (SF_LOCK | SF_INUSE | SF_FOLDER | SF_EXTMEM |
          SF_EM_TWIN))) {
         HSymDel( hSym );
         return( FS_OK );
      }
      return( FS_ERROR );
}
```

VarStore

Declaration: HSYM **VarStore** (BYTE * *DestVar*, WORD *Flags*, WORD *SourceSize*, [*parm1*] [, *parm2*])

Category(ies): Symbol Table Utilities, Variables

Description: This is the general system routine for storing to variables. It handles system variables (some of which are not in the symbol table) as well as all of the special cases for all variables throughout the system.

> **NOTE:** See the "Storing and Retrieving variable data" section in the Memory Management chapter for further details on **VarStore**.

Inputs:

DestVar — EStackIndex of variable name to store to.

Flags — STOF_ESI, STOF_HESI, STOF_ELEMENT, STOF_NONE, USER_FUN_TAG, TEXT_VAR_TAG, GDB_VAR_TAG, PIC_VAR_TAG, DATA_VAR_TAG, GEN_DATA_TAG

SourceSize — Size of source.

Outputs: Returns the HSYM of the newly created variable. Otherwise returns NULL for system variables not in the symbol table.

Assumptions: None

Side Effects: May cause heap compression.

Availability: All versions of the TI‑89 / TI‑92 Plus.

TI‑89 / TI‑92 Plus Differences: None

See Also: **VarRecall, TokenizeSymName**

(continued)

VarStore *(continued)*

Example:

```
/* This example routine stores to the variable 'var' the region defined by 'wr'
   for the WINDOW pointed to by 'winPtr'.
*/
HSYM StoPic( WINDOW *winPtr, EStackIndex var, WIN_RECT *wr )
{  WORD SaveSize;
   HSYM hsym;
   void *Ptr;
   EStackIndex varI;
   BOOL oldFlag;
   SYM_ENTRY *symPtr;

   if ((SaveSize = WinBitmapSize( winPtr, wr )) > 0) {
      if (hsym = VarStore(var, PIC_VAR_TAG, SaveSize+1)) {
         symPtr = DerefSym( hsym );
         symPtr->Version = TV_TI_92;   /* PICs are same across all versions */
         Ptr = HeapDeref(symPtr->hVal);
         (*(WORD *) Ptr) = SaveSize + 1;   /* store size of picture */
         Ptr = (BYTE *) Ptr + 2;   /* skip size */
         (*((BYTE *) Ptr + SaveSize)) = PIC_VAR_TAG;   /* store tag */
         oldFlag = WinDupStat( winPtr, FALSE );   /* get backup image if have one */
         WinBitmapGet( winPtr, wr, (BITMAP *) Ptr );   /* get image data */
         WinDupStat( winPtr, oldFlag );   /* restore DupStat */
      }  else
         ER_THROW( ER_RESERVED );
   }  else
      ER_THROW( ER_DIMENSION );
   return( hsym );
}
```

Appendix A: System Routines — Text Editing

CB_fetchTEXT

Declaration: BOOL **CB_fetchTEXT** (HANDLE * *h*, size_t * *size*)

Category(ies): Text Editing

Description: Fetch contents of clipboard if it contains text.

Inputs: None

Output: *h* — Handle to text in clipboard.

size — Length of text.

Returns TRUE if clipboard contains any text, otherwise, returns FALSE.

Assumptions: None

Side Effects: None

Availability: All versions of the TI-89 / TI-92 Plus.

TI-89 / TI-92 Plus Differences: None

See Also: **CB_replaceTEXT**

Example:

```
HANDLE hText;
size_t size;
   .
   .
   .
if (CB_fetchTEXT(&hText, &size))
{
   /* do something with text in hText */
}
```

CB_replaceTEXT

Declaration: BOOL **CB_replaceTEXT** (void * *pText*, size_t *size*, BOOL *bStripCmd*)

Category(ies): Text Editing

Description: Copy text to clipboard.

Inputs:

pText — Pointer to text to copy to clipboard.

size — Byte length of text.

bStripCmd — Remove command byte after each carriage return in copied text. The built-in text editor application passes TRUE in this parameter to remove the command byte at the beginning of each line when copying text to the clipboard.

Outputs: Return TRUE if text was copied to clipboard. Return FALSE if memory could not be allocated for clipboard copy of text.

Assumptions: None

Side Effects: May cause heap compression.

Availability: All versions of the TI‑89 / TI‑92 Plus.

TI‑89 / TI‑92 Plus Differences: None

See Also: **CB_fetchTEXT**

Example:

```
CB_replaceTEXT(&textbuf[offset], textlen, FALSE);
```

TE_close

Declaration: void **TE_close** (TERecord * *teRec*)

Category(ies): Text Editing

Description: Releases edit buffer memory from text edit record.

Inputs: *teRec* — Text edit record previously initialized with a call to **TE_open**, **TE_reopen**, or **TE_openFixed**.

Outputs: None

Assumptions: This routine releases its edit buffer memory. You should store or process the edit buffer's contents before calling this routine. Call **TE_shrinkWrap** to get the handle to the edit buffer.

Side Effects: None

Availability: All versions of the TI-89 / TI-92 Plus.

TI-89 / TI-92 Plus Differences: None

See Also: **TE_open, TE_reopen, TE_openFixed, TE_shrinkWrap**

Example: See **TE_open**.

TE_empty

Declaration: void **TE_empty** (TERecord * *teRec*)

Category(ies): Text Editing

Description: Deletes all text from edit buffer.

Inputs: *teRec* — Contains the text edit state.

Outputs: None

Assumptions: This routine turns off cursor blink before emptying the edit buffer. This routine does not repaint the edit region. It makes its parent window dirty so the edit region will be updated when the next paint message arrives.

Side Effects: May cause heap compression. Sets WF_DIRTY flag in parent window.

Availability: All versions of the TI‑89 / TI‑92 Plus.

TI‑89 / TI‑92 Plus Differences: None

See Also: None

Example:

```
case CM_CLEAR_ALL:      /* User chose Clear Editor from Tools menu */
   /* Confirm Clear Editor */
   if (Dialog(&DataClearDlg, -1, -1, NULL, NULL) == KB_ENTER)
   {
      TE_empty(&teText);           /* OK - empty edit buffer */
      TE_focus(&teText);           /* Turn cursor blink back on */
   }
   break;
```

TE_focus

Declaration: BOOL **TE_focus** (TERecord * *teRec*)

Category(ies): Text Editing

Description: Makes *teRec* the current input field. Highlights selected text or turns on cursor blink in the input field.

Inputs: *teRec* — Contains the text edit state.

Outputs: Returns TRUE if text edit was already focused on this field before calling this routine. Otherwise returns FALSE.

Assumptions: This routine is used in conjunction with **TE_unfocus** to move the focus of text editing from one edit field to another. Call **TE_unfocus** to unhighlight text and stop cursor blink in the current edit field. Then call **TE_focus** to highlight text and start cursor blink in another edit field.

Side Effects: Cursor blink is turned on.

Availability: All versions of the TI-89 / TI-92 Plus.

TI-89 / TI-92 Plus Differences: None

See Also: **TE_unfocus**

Example:

```
case KB_ENTER:   /* Move edit focus to next field */
   TE_unfocus(&teField1);
   TE_focus(&teField2);
   break;
```

TE_handleEvent

Declaration: BOOL **TE_handleEvent** (TERecord * *teRec*, Event * *e*)

Category(ies): Text Editing

Description: This routine is the text editor's event handler. Keypresses, cursor blink, cut, copy and paste operations, displaying and scrolling text, and other details of interpreting the interaction between the user and text edit fields are handled by this routine.

Applications using the text edit manager typically process events in three phases.

In phase one, the application examines the event for action it needs to take. Either the application handles the event and returns to the event manager or it proceeds to phase two.

In phase two, the application calls **TE_handleEvent** to allow the text edit manager to process the event. Either **TE_handleEvent** handles the event and returns function value TRUE, or it does not understand the event and returns FALSE. If **TE_handleEvent** does not handle the event, the application proceeds to phase three.

In phase three, the application calls **EV_defaultHandler** to let the event manager have one last try at handling the event. System-wide default behavior is implemented in **EV_defaultHandler**.

Inputs:

teRec — The state record of a text edit field.

e — An event message received from the event manager. Applications may drive the text editor by calling **TE_handleEvent** with their own created event messages, but in practice, the application just forwards events it received from the event manager.

Outputs: Returns TRUE if the text edit manager handled the event.

Returns FALSE if the text edit manager did not understand the event and consequently did not process the event.

Assumptions: *teRec* must have been initialized by a prior call to **TE_open**, **TE_openFixed**, or **TE_reopen**.

Side Effects: The heap may be compressed.

Availability: All versions of the TI-89 / TI-92 Plus.

(continued)

TE_handleEvent *(continued)*

TI-89 / TI-92 Plus Differences: None

See Also: **TE_open, TE_openFixed, TE_reopen**

Example:

```
void appHandleEvent(Event *e)   /* receive message from event manager */
{
   switch (e->command)
   {
      case CM_WPAINT:   /* paint my window */
        .
        .
        .
      default:
         if (! TE_handleEvent(&appteRec, e))  /* did text edit handle event? */
            EV_defaultHandler(e);             /* no, defer to default behavior */
         break;
   }
}
```

TE_indicateReadOnly

Declaration: void **TE_indicateReadOnly** (TERecord * *teRec*)

Category(ies): Text Editing

Description: If text edit record *teRec* was opened with the TE_READ_ONLY flag set, turns on the read-only indicator (padlock symbol) in the status line and disables menu commands “Cut,” “Paste,” “Delete,” and “Clear All” if they are defined in the application’s registered menu.

Inputs: *teRec* — The state record of a text edit field.

Outputs: None

Assumptions: The application should call this routine after it opens a read-only text edit field or if it is focused on a text edit field when it is activated (receives CM_ACTIVATE message).

The application should call **ST_readOnly**(ST_READONLY_OFF) to turn off the status line read-only indicator when it is deactivated (receives CM_DEACTIVATE message).

Side Effects: None

Availability: All versions of the TI-89 / TI-92 Plus.

TI-89 / TI-92 Plus Differences: None

See Also: **TE_open, ST_readOnly**

Example:

```
TE_open(&teRec, &window, NULL, hText, 0, 0, TE_READ_ONLY | TE_WRAP);
                                          /* open read-only text */
TE_indicateReadOnly(&teRec);              /* turn on padlock and dim edit menus */
```

TE_isBlank

Declaration: BOOL **TE_isBlank** (TERecord * *teRec*)

Category(ies): Text Editing

Description: Query if edit buffer is empty or filled only with blanks.

Inputs: *teRec* — The state record of a text edit field.

Outputs: Returns TRUE if edit buffer is empty or contains only blanks. Returns FALSE otherwise.

Assumptions: None

Side Effects: None

Availability: All versions of the TI‑89 / TI‑92 Plus.

TI‑89 / TI‑92 Plus Differences: None

See Also: **None**

Example:

```
if (! TE_isBlank(&teRec)) {
   /* evaluate expression in edit buffer */
   .
   .
   .
}
```

TE_open

Declaration: BOOL **TE_open** (TERecord * *teRec*, WINDOW * *w*, WIN_RECT * *editRect*, HANDLE *hText*, USHORT *cursor*, USHORT *promptlen*, TE_FLAGS *flags*)

Category(ies): Text Editing

Description: Open a text edit record. Initialize fields of *teRec*.

Inputs:

w — Window containing edit field.

editRect — Rectangle defining the window-relative extents of the edit field or NULL to use entire client rectangle of window *w* for the edit region.

hText — Handle to the text to be edited. The text must be terminated with a zero (0x00) byte. If *hText* is H_NULL, **TE_open** will allocate a new handle and initialize it with no text.

cursor — Offset from the beginning of the text to the position to display the edit cursor. Position 0 is to the left of the first character. If the contents of the text edit buffer are too long to display entirely in the edit region, the text is scrolled to make sure the cursor is visible.

Set *cursor* to TE_FAR_RIGHT to place the edit cursor after the last character in the edit buffer.

promptlen — Indicates how many characters at the beginning of the text make up a prompt. The user cannot change the text of the prompt characters nor move the edit cursor into the prompt.

flags — Each bit specifies optional features of the text editor.

0x0001 TE_WRAP

Set this flag to 1 for multiline edit regions. Set to 0 for single line edit regions. Text wraps around end of line to the beginning of the next line in multiline edit regions.

The program editor is an example of a multiline edit region. The Home screen author line is an example of a single line edit region.

(continued)

TE_open *(continued)*

Inputs: ***(continued)***

0x0002 TE_COLON

Place colon (:) character at the beginning of each line. Program editor uses this flag to mark the beginning of each line of the program.

0x0006 TE_COMMANDS

Leave room for one character at the beginning of each line for a command character. The Text Editor application uses this flag.

NOTE: This flag is combined with the TE_COLON flag.

0x0008 TE_MORE_ARROWS

Set this flag to display arrows at the left and right ends of a single line edit region to indicate when more text is to the left or right of the edit region.

0x0018 TE_MORE_ELLIPSES

Set this flag to display ellipses (. . .) at the left and right ends of a single line edit region to indicate when more text is to the left or right of the edit region.

0x0100 TE_CHANGED

Do not set this flag. It is a status flag which if 1 indicates the contents of the edit buffer have changed.

0x0400 TE_AUTO_ANS

Set this flag to 1 to cause “ans(1)” to be inserted automatically when the edit buffer is empty and an arithmetic operation is typed.

0x0800 TE_READ_ONLY

Display text and allow arrow keys to navigate through edit buffer, but do not allow changing the text.

The remaining bits are internal status flags or reserved for future use.

(continued)

TE_open *(continued)*

Outputs: Returns TRUE if edit buffer could be allocated, or FALSE if insufficient memory to allocate edit buffer. This routine always returns TRUE if *hText* is passed in with the handle to a text buffer.

teRec is initialized with text edit state. *teRec* must be allocated statically to maintain state between calls to the text edit routines.

Assumptions: Window *w* must already be open. Handle *hText* must not be locked.

Side Effects: May cause heap compression.

Availability: All versions of the TI-89 / TI-92 Plus.

TI-89 / TI-92 Plus Differences: None

See Also: **TE_close, TE_openFixed, TE_reopen, TE_shrinkWrap**

Example:

```
WIN_RECT rect;
static TERecord teRec;
   .
   .
   .
/* Create an edit region at the top of window */
rect.x0 = 0;
rect.y0 = 0;
rect.x1 = WinWidth(&window);
rect.y1 = LF_HEIGHT + 1;
bOK = TE_open(&teRec, &window, &rect, H_NULL, 0, 0, TE_MORE_ELLIPSES);
   .
   .
   .
TE_close (&teRec);
```

TE_openFixed

Declaration: void **TE_openFixed** (TERecord * *teRec*, WINDOW * *w*, WIN_RECT * *editRect*, UCHAR * *buf*, USHORT *length*, TE_FLAGS *flags*)

Category(ies): Text Editing

Description: Open a text edit record with a fixed length buffer. Initialize fields of *teRec*.

Inputs:

w — Window containing edit field.

editRect — Rectangle defining the window-relative extents of the edit field or NULL to use entire client rectangle of window *w* for the edit region.

buf — Fixed-length byte buffer of text. Initial text in the buffer must be zero terminated. The buffer must be big enough for the longest string to be edited with one additional byte for the zero string terminator.

length — Buffer length in bytes.

flags — Each bit specifies optional features of the text editor.

0x0001 TE_WRAP
Set this flag to 1 for multiline edit regions. Set to 0 for single line edit regions. Text wraps around end of line to the beginning of the next line in multiline edit regions.

0x0002 TE_COLON
Place colon (:) character at the beginning of each line. Program editor uses this flag to mark the beginning of each line of the program.

0x0006 TE_COMMANDS
Leave room for one character at the beginning of each line for a command character. The Text Editor application uses this flag.

> **NOTE:** This flag is combined with the TE_COLON flag.

0x0008 TE_MORE_ARROWS
Set this flag to display arrows at the left and right ends of a single line edit region to indicate when more text is to the left or right of the edit region.

(continued)

TE_openFixed *(continued)*

Inputs: ***(continued)***

0x0018 TE_MORE_ELLIPSES
Set this flag to display ellipses (. . .) at the left and right ends of a single line edit region to indicate when more text is to the left or right of the edit region.

0x0100 TE_CHANGED
Do not set this flag. It is a status flag which if 1 indicates the contents of the edit buffer have changed.

0x0400 TE_AUTO_ANS
Set this flag to 1 to cause “ans(1)” to be inserted automatically when the edit buffer is empty and an arithmetic operation is typed.

0x0800 TE_READ_ONLY
Display text and allow arrow keys to navigate through edit buffer, but do not allow changing the text.

The remaining bits are internal status flags or reserved for future use.

Outputs: *teRec* is initialized with text edit state. *teRec* must be allocated statically to maintain state between calls to the text edit routines.

Assumptions: Window *w* must already be open.

Side Effects: None

Availability: All versions of the TI‑89 / TI‑92 Plus.

TI‑89 / TI‑92 Plus Differences: None

See Also: **TE_close, TE_open, TE_reopen, TE_shrinkWrap**

Example:

```
WIN_RECT rect;
static TERecord teRec;
UCHAR text[20];
   .
   .
   .
/* Create an edit region at the top of window */
rect.x0 = 0;
rect.y0 = 0;
rect.x1 = WinWidth(&window);
rect.y1 = LF_HEIGHT + 1;
TE_openFixed(&teRec, &window, &rect, text, sizeof(text), TE_MORE_ARROWS);
```

TE_pasteText

Declaration: void **TE_pasteText** (TERecord * *teRec*, UCHAR * *text*, size_t *size*)

Category(ies): Text Editing

Description: Insert text into edit buffer at the cursor blink location or replace highlighted text in the edit buffer.

The cursor is normally moved to the end of the newly inserted text. A different cursor location may be indicated by an embedded character '\002' (TE_SELECTION_MARKER) in the text where the cursor should be located. Furthermore, two embedded selection marker characters delimit a range of characters to highlight.

For example, inserting the text "ABC\002DEF" leaves the cursor between C and D instead of after F. Inserting the text "ABCD\002EFGH\002IJKL" leaves the characters EFGH highlighted, thus: ABCDEFGHIJKL.

If the TE_WRAP flag is not set in the text edit record *teRec*, carriage return characters are converted into colons to show where lines are separated. See **TE_open** for a description of the TE_WRAP flag.

Inputs:

teRec — Pointer to text edit record previously initialized with a call to **TE_open** or **TE_openFixed**.

text — Pointer to a character array of text to insert into edit buffer.

size — Byte length of character array to insert into edit buffer.

Outputs: The edit buffer in *teRec* is updated with inserted text.

Assumptions: The character array pointed to by text must not move during the paste operation. If the text being inserted resides in the memory heap, you should lock its handle before passing the address of the text array to **TE_pasteText**. You can use **Hlock** to lock a handle and look up its address in one operation.

Side Effects: Inserting text can cause a memory full error in which case the new text is not inserted and the edit buffer remains unchanged.

The heap may be compressed.

Availability: All versions of the TI-89 / TI-92 Plus.

TI-89 / TI-92 Plus Differences: None

See Also: **TE_open**

(continued)

TE_pasteText *(continued)*

Example:

```
size_t size;
UCHAR *pText;
HANDLE hText;
.
.
.
if (CB_fetchTEXT(&hText, &size))            /* get handle to clipboard text */
{
   pText = HLock(hText);                    /* lock handle and convert to pointer */
   TE_pasteText(&textEdit, pText, size);    /* paste text into edit buffer */
   HeapUnlock(hText);                       /* unlock clipboard handle */
}
```

TE_reopen

Declaration: void **TE_reopen** (TERecord * *teRec*, BOOL *bFocus*)

Category(ies): Text Editing

Description: Reopen text edit record closed by **TE_shrinkWrap**. Use this routine when you have a text edit field which you want to reopen containing previous input from the user. Unlike **TE_reopenPlain**, this routine changes the edit selection to cover all the text.

Inputs:

teRec — Pointer to text edit record previously closed with a call to **TE_shrinkWrap**.

bFocus — TRUE means automatically set the focus to this field after reopening the text edit record.

Outputs: None

Assumptions: Flags, edit buffer, and most state information in *teRec* are used as they were set when the edit record was closed by **TE_shrinkWrap**. The entire contents of the edit buffer are selected, i.e., the selection highlight covers all the text.

Side Effects: May cause heap compression. Turns cursor blink on if *bFocus* is TRUE.

Availability: All versions of the TI‑89 / TI‑92 Plus.

TI‑89 / TI‑92 Plus Differences: None

See Also: **TE_reopenPlain, TE_shrinkWrap**

Example: See **TE_shrinkWrap**.

TE_reopenPlain

Declaration: void **TE_reopenPlain** (TERecord * *teRec*, BOOL *bFocus*)

Category(ies): Text Editing

Description: Reopen text edit record closed by **TE_shrinkWrap**. Use this routine when you have a text edit field which you want to reopen containing previous input from the user. The cursor selection highlight starts in the same place as when the field was closed.

Inputs:

teRec — Pointer to text edit record previously closed with a call to **TE_shrinkWrap**.

bFocus — TRUE means automatically set the focus to this field after reopening the text edit record.

Outputs: None

Assumptions: Flags, edit buffer, and state information in *teRec* are used as they were set when the edit record was closed by **TE_shrinkWrap**. The cursor position or edit selection highlight remains unchanged from when the field was closed.

Side Effects: May cause heap compression. Turns cursor on if *bFocus* is TRUE.

Availability: All versions of the TI‑89 / TI‑92 Plus.

TI‑89 / TI‑92 Plus Plus Differences: None

See Also: **TE_reopen, TE_shrinkWrap**

Example: See **TE_shrinkWrap**.

TE_select

Declaration: void **TE_select** (TERecord * *teRec*, USHORT *left*, USHORT *right*)

Category(ies): Text Editing

Description: Set left and right ends of selected text in an edit buffer. The selected text is visible on screen as highlighted text in an edit field. The selection may extend over more than one line of text.

Inputs:

teRec — Pointer to text edit record previously initialized with a call to **TE_open** or **TE_openFixed**.

left — Edit buffer offset to beginning of selected text — zero (0) may be used to set the left end of the selection to the beginning of the edit buffer.

right — Edit buffer offset to end of selected text. Symbol TE_FAR_RIGHT may be used to set the right end of the selection to the end of the edit buffer.

Outputs: None

Assumptions: None

Side Effects: None

Availability: All versions of the TI-89 / TI-92 Plus.

TI-89 / TI-92 Plus Differences: None

See Also: Not applicable.

Example:

```
TE_select(&textEdit, 0, TE_FAR_RIGHT);   /* select entire contents of edit buffer */
```

TE_shrinkWrap

Declaration: HANDLE **TE_shrinkWrap** (TERecord * *teRec*)

Category(ies): Text Editing

Description: Returns handle to edit buffer. This routine turns off the cursor and frees any slack space in the edit buffer. Unlike **TE_close**, this routine does not release edit buffer memory.

During normal edit processing, the text editor keeps some slack space in the edit buffer. This allows new text to be inserted quickly without annoying pauses while the edit buffer is expanded to accommodate another character. **TE_shrinkWrap** removes the slack space before returning the edit buffer to you.

Inputs: *teRec* — Pointer to text edit record previously initialized with a call to **TE_open** or **TE_openFixed**.

Outputs: Returns a handle to the edit buffer. A null byte marks the end of the text in the edit buffer.

Assumptions: This routine does not free edit buffer memory. You must either free the edit buffer handle or call **TE_close** to do it when you are finished with the text.

Side Effects: Cursor blink is turned off.

Availability: All versions of the TI-89 / TI-92 Plus.

TI-89 / TI-92 Plus Differences: None

See Also: **TE_open, TE_openFixed, TE_reopen**, **TE_reopenPlain**

Example:

```
hBuf = TE_shrinkWrap(&teRec);   /* get handle to edit buffer */
   .
   .   /* process contents of edit buffer */
   .
if (bSelectAll)
   TE_reopen(&teRec, TRUE);   /* select all text */
else
   TE_reopenPlain(&teRec, TRUE);  /* start cursor/select where TE_shrinkWrap left it */
```

TE_unfocus

Declaration: BOOL **TE_unfocus** (TERecord * *teRec*)

Category(ies): Text Editing

Description: Removes focus from *teRec*. Unhighlights selected text or turns off cursor blink in the input field.

Inputs: *teRec* — Contains the text edit state.

Outputs: Returns TRUE if text edit was focused on this field before calling this routine. Otherwise returns FALSE.

Assumptions: This routine is used in conjunction with **TE_focus** to move the focus of text editing from one edit field to another. Calls **TE_unfocus** to unhighlight text and stop cursor blink in the current edit field. Then calls **TE_focus** to highlight text and start cursor blink in another edit field.

Side Effects: Cursor blink is turned off.

Availability: All versions of the TI‑89 / TI‑92 Plus.

TI‑89 / TI‑92 Plus Differences: None

See Also: **TE_focus**

Example: See **TE_focus**.

Appendix A: System Routines — Timer

OSFreeTimer

Declaration: WORD **OSFreeTimer** (WORD *TimerNode*)

Category(ies): Timer

Description: Free a system timer so it can be used by another application.

Inputs: *TimerNode* — Name of the timer node being requested. Currently only the USER timer is available for apps.

Outputs: False (zero) if not successful, true (not zero) otherwise.

Assumptions: None

Side Effects: Freeing system timers other than the USER timer is not recommended.

Availability: All versions of the TI-89 / TI-92 Plus.

TI-89 / TI-92 Plus Differences: None

See Also: **OSRegisterTimer, OSTimerCurVal, OSTimerExpired, OSTimerRestart**

Example: See **idle**.

OSRegisterTimer

Declaration: WORD **OSRegisterTimer** (WORD *TimerNode*, DWORD *InitVal*)

Category(ies): Timer

Description: Request a system timer. Initialize it to start counting down from *InitVal*.

Inputs:

TimerNode — Name of the timer node being requested. Currently only the USER timer is available for apps.

InitVal — Number of system ticks before rollover. See tiams.h for equates that relate system ticks to approximate time.

Outputs: False (zero) if not successful, true (not zero) otherwise.

Assumptions: Some useful time equates in tiams.h:

MS50	—	50 milliseconds (smallest increment).
MS100	—	100 milliseconds.
MS500	—	500 milliseconds.
ONE_SECOND	—	One second.
ONE_MINUTE	—	One minute.

Side Effects: None

Availability: All versions of the TI-89 / TI-92 Plus.

TI-89 / TI-92 Plus Differences: None

See Also: **OSFreeTimer, OSTimerCurVal, OSTimerExpired, OSTimerRestart**

Example: See **idle**.

OSTimerCurVal

Declaration: DWORD **OSTimerCurVal** (WORD *TimerNode*)

Category(ies): Timer

Description: Returns the current value of the queried timer.

Inputs: *TimerNode* — Name of the timer node being queried.

Outputs: Number of system ticks until rollover. –1 indicates not running.

Assumptions: None

Side Effects: None

Availability: All versions of the TI-89 / TI-92 Plus.

TI-89 / TI-92 Plus Differences: None

See Also: **OSRegisterTimer, OSFreeTimer, OSTimerExpired, OSTimerRestart**

Example:

```
TickCount = OSTimerCurVal(USER)*3;
```

OSTimerExpired

Declaration: WORD **OSTimerExpired** (WORD *TimerNode*)

Category(ies): Timer

Description: Test if the timer has rolled over.

Inputs: *TimerNode* — Name of the timer node being queried.

Outputs: Returns true if the timer has expired, false otherwise.

Assumptions: None

Side Effects: Resets the timer's expired flag.

Availability: All versions of the TI-89 / TI-92 Plus.

TI-89 / TI-92 Plus Differences: None

See Also: **OSRegisterTimer, OSFreeTimer, OSTimerCurVal, OSTimerRestart**

Example: See **idle**.

OSTimerRestart

Declaration: DWORD **OSTimerRestart** (WORD *TimerNode*)

Category(ies): Timer

Description: Reset an already allocated timer.

Inputs: *TimerNode* — Name of the timer node being queried.

Outputs: Value of the timer at the time of the call.

Assumptions: Sets the value of the timer to that passed in by the **OSRegisterTimer** call, and resets the timer expired flag.

Side Effects: None

Availability: All versions of the TI-89 / TI-92 Plus.

TI-89 / TI-92 Plus Differences: None

See Also: **OSRegisterTimer, OSFreeTimer, OSTimerCurVal, OSTimerExpired**

Example:

```
OldValue = OSTimerRestart(USER);
```

Appendix A: System Routines — Token Operations

See Also:

get_key_ptr

Declaration: const unsigned char * **get_key_ptr** (Quantum *q1*, Quantum *q2*)

Category(ies): Token Operations, Strings

Description: Used by the detokenization routine to access the catalog string associated with a particular token.

Inputs: *q1* — The first token value.

q2 — The second token value or zero if the first token is a primary tag.

Outputs: A pointer to the catalog string associated with the token specified by the input values. The return value may be a NULL pointer if no string is associated with the input values.

Assumptions: The input token is NOT any type of user variable, any type of number, or any type of arbitrary constant (@1, @n1, etc.). These types will cause the routine to return a pointer to the string "_ERROR_".

Side Effects: None

Availability: On AMS 1.05 and higher.

TI-89 / TI-92 Plus Differences: None

See Also: None

(continued)

get_key_ptr *(continued)*

Example:

```
/* The system only uses get_key_ptr after user variables, arbitrary constants, and
   numbers have been processed. If i is the EStackIndex of an expression, and kp is
   an unsigned char pointer, then the following is an appropriate approach. */

Quantum q, r;

q = ESTACK (i);   /* get the topmost tag */
r = 0;            /* assume the topmost tag is a primary tag */
if (IS_USER_VAR (q))  /* check for user variables */
{
   . . . /* code to handle user variables */
}
else if (IS_ARB_TAG (q))   /* check for arbitrary constants */
{
   . . . /* code to handle arbitrary constants, @1, @n1, etc. */
}
else if (IS_NUMBER_TAG (q))  /* check for numbers */
{
   . . . /* code to handle numbers */
}
else if (IS_DOUBLE_TAG (q))  /* check for tokens that require two values */
{                            /* SYSVAR_TAG, SECONDARY_TAG, COMMAND_TAG */
   r = ESTACK (i - 1);
}
kp = get_key_ptr (q, r);    /* get pointer to catalog string */
.
.
.

/* ------------------------------------------------------------------- */
/* if q = SIN_TAG and r = 0, then get_key_ptr (q, r) returns a pointer  */
/* to the string “sin(“.                                                */
/*                                                                      */
/* if q = SYSVAR_TAG and r = SV_XMIN, then get_key_ptr (q, r) returns a */
/* pointer to the string “xmin”.                                        */
/*                                                                      */
/* if q = SECONDARY_TAG and r = GETKEY_TAG, then get_key_ptr (q, r)     */
/* returns a pointer to the string “getkey()”.                          */
/* ------------------------------------------------------------------- */
```

GetTagStr

Declaration: char * **GetTagStr** (EStackIndex *tagPtr*, char * *Buf*)

Category(ies): Token Operations, Strings

Description: Return a pointer to the string which defines a tag.

Inputs: *tagPtr* — Pointer to tag to decode.

Buf — Buffer of at least 11 bytes long.

Outputs: Returns *Buf*.

Assumptions: *Buf* must be at least 11 bytes long.

Side Effects: None

Availability: On AMS 2.00 and higher.

TI-89 / TI-92 Plus Differences: None

See Also: None

Example: The **ShowStat** command prints the string name for each statistical variable it finds using **GetTagStr** to print the name of the variable as shown in this example.

```
BYTE Tag[2];
char Buf[24];
char kwBuf[SYM_LEN+2];

Tag[0] = Index; /* SV_XBAR .. SV_MEDY */
Tag[1] = SYSVAR_TAG;
sprintf( Buf, "%-7s", GetTagStr( Tag+1, kwBuf ) );
```

NG_RPNToText

Declaration: HANDLE **NG_RPNToText** (HANDLE *hExpr*, BOOL *bRetToColon*, BOOL *bFullPrec*)

Category(ies): Token Operations

Description: Converts the tokenized form of an expression, statement, or group of statements as contained in the data referenced by the handle *hExpr* to linear ASCII text form. The first word of the data must be the length of the data which is used to find the first tag of the tokenized data.

Inputs:

hExpr — Handle to tokenized object (first word is length of object).

bRetToColon — If TRUE, convert carriage returns to colons; otherwise leave them alone.

bFullPrec — If TRUE convert floats to full precision; otherwise convert floats according to the current mode settings.

Outputs: Returns the HANDLE to a heap packet which contains the ASCII text result. Throws an ER_MEMORY error if not enough memory.

Assumptions: None

Side Effects: May cause heap compression.

Availability: All versions of the TI-89 / TI-92 Plus.

TI-89 / TI-92 Plus Differences: None

See Also: **display_statements**

Example: This example is a small viewer for variables of type EXPR, LIST, MAT, and STR. These types are always tokenized (programs and functions may be in text format).

```
/* Resource for vSym */
DIALOG dGetName, 0, 0, NoCallBack
{
   EDIT,   {0, 8, 15}, "", 0, 17, 18
   HEADER, {0, 0, 0}, "Enter variable name, ESC to exit", PDB_OK, PDB_CANCEL
   XFLAGS, {0, 0, 0}, XF_ALLOW_VARLINK | XF_VARLINK_SELECT_ONLY, 0, 0, 0
}

void vSym( void )
{  char inpBuf[MAX_SYM_LEN];
   BYTE nameBuf[MAX_SYM_LEN];
   HSYM hsym;
   HANDLE hVal;
   volatile HANDLE hText;
   SYM_ENTRY *SymPtr;
```

(continued)

NG_RPNToText *(continued)*

Example:
(continued)

```
   while (KB_ENTER == Dialog( &dGetName,-1,-1, inpBuf, NULL)) {
      if (FS_OK == TokenizeName( inpBuf, nameBuf ))
         if (hsym = SymFindMain( TokNameRight(nameBuf) )) {
            SymPtr = DerefSym( hsym );
            if (hVal = SymPtr->hVal) {
               switch ((BYTE) *HToESI( hVal ))  {
                  /* NG_RPNToText cannot handle these types */
                  case GEO_FILE_TAG:
                  case GEO_MACRO_TAG:
                  case DATA_VAR_TAG:
                  case GDB_VAR_TAG:
                  case ASM_PRGM_TAG:
                  case PIC_VAR_TAG:
                  case GEN_DATA_TAG:
                  case TEXT_VAR_TAG:
                     DlgNotice( "NOTE", "Can not display value" );
                     break;
                  /* programs/functions are special, would need
                     to use GetFuncPrgmBodyPtr */
                  case USER_DEF_TAG:
                     DlgNotice( "NOTE", "Program or function" );
                     break;
                  default:
                     HeapLock( hVal ); /* tokenized */
                     TRY
                        hText = NG_RPNToText( hVal, TRUE, TRUE );
                     ONERR
                        hText = H_NULL;
                        ERD_dialog( errCode, FALSE );
                     ENDTRY
                     HeapUnlock( hVal );
                     if (hText) {
                        HeapLock( hText );
                        DlgNotice( (char *) inpBuf, (char *) (HeapDeref(hText)) );
                        HeapFree( hText );
                     }
               }  /* end switch */
            }  else
               DlgNotice( "ERROR", "Symbol has no value" );
         }  else
            DlgNotice( "ERROR", "Symbol not found" );
   } /* end while */
} /* end vSym */
```

NG_tokenize

Declaration: BOOL **NG_tokenize** (HANDLE *hText,* SINT * *errNo,* USHORT * *offset*)

Category(ies): Token Operations

Description: Creates external-tokenized form from text input.

Inputs:

hText — HANDLE of memory block that contains null terminated text string.

errNo — Pointer to a signed integer in which an error code can be returned.

offset — Pointer to an unsigned integer in which the location of the error can be returned as an offset from the beginning of the text.

Outputs: Pushes the external-tokenized form of the input onto the expression stack.

When no error occurs, returns TRUE and

— * *errNo* = TRUE if input represented a single algebraic expression.

— * *errNo* = FALSE if input represented a STO operation, a command, or more than one statement.

— * *offset* is not accessed.

When an error occurs, returns FALSE and

— * *errNo* = error code.

— * *offset* = location of error as an offset from the beginning of the text.

Assumptions: None

Side Effects: May expand expression stack, cause heap compression, or throw an error.

Availability: All versions of the TI‑89 / TI‑92 Plus.

TI‑89 / TI‑92 Plus Differences: None

See Also: **push_parse_text**

Example:

```
/* If h is the handle of a memory block containing the text string "x + y", then */

SINT errNo;
USHORT offset;
BOOL err;
err = NG_tokenize (h, &errNo, &offset);

/* pushes the external-tokenized form X_VAR_TAG Y_VAR_TAG ADD_TAG onto the
   expression stack */
```

push_parse_text

Declaration: Boolean **push_parse_text** (unsigned char * *cp*)

Category(ies): Token Operations

Description: The tokenization routine that creates external tokenized form from text input.

Inputs: *cp* — Points to a null terminated text string.

Outputs: Pushes the external tokenized form of the input onto the expression stack.

Returns TRUE when the input represents a single algebraic expression.

Returns FALSE when the input contains a STO▸ operation, any command, or any statement separator indicating more than one statement is represented.

Assumptions: None

Side Effects: May expand expression stack and may cause heap compression.

Availability: On AMS 1.05 and higher.

TI-89 / TI-92 Plus Differences: None

See Also: **NG_tokenize**

Example:

```
/* Given the text string "x + y", push the external tokenized form of the expression
   (X_VAR_TAG Y_VAR_TAG ADD_TAG) onto the expression stack. */
unsigned char expression[] = "x + y";
(void) push_parse_text (expression);
```

Appendix A: System Routines — Utilities

See Also:

AB_getGateArrayVersion

Declaration: ULONG **AB_getGateArrayVersion** (void)

Category(ies): Utilities

Description: Returns the version number of the gate array. This really amounts to a hardware revision number.

Currently there are two gate array versions. Gate array version 1 was introduced in the original TI‑92 calculator. Gate array version 2 increased the speed of the calculator and instituted changes in LCD memory addressing, RAM execution protection, Flash memory protection, low battery detection, and timer control. Gate array version 2 is used in the manufacture of all new TI‑92 Plus and TI‑89 calculators.

Inputs: None

Outputs: Returns either the number 1 or 2.

Assumptions: None

Side Effects: None

Availability: On AMS 2.00 and higher.

TI‑89 / TI‑92 Plus Differences: None

See Also: None

Example:

```
ULONG hwVersion = AB_getGateArrayVersion();
```

AB_prodid

Declaration: void **AB_prodid** (char *prodid*[])

Category(ies): Utilities

Description: Returns a string containing the product ID of the Operating System software running in the calculator. The ID string is in the form “p-h-r-b”.

p — Product number: 01 for TI‑92 Plus, 03 for TI‑89.
h — Hardware revision level.
r — Software revision level.
b — Build number.

All the above fields consist of hexadecimal digits.

Inputs: None

Output: *prodid* — A character buffer large enough (≥ 12 bytes) to hold the product ID string.

Assumptions: None

Side Effects: None

Availability: All versions of the TI‑89 / TI‑92 Plus.

TI‑89 / TI‑92 Plus Differences: None

See Also: **AB_prodname, AB_serno**

Example:

```
char prodid[12];
AB_prodid(prodid);   /* get product ID */
```

AB_prodname

Declaration: void **AB_prodname** (char *name*[])

Category(ies): Utilities

Description: Get the name of the Operating System software running in the calculator. This is the same name that appears on the second line of the About window.

Inputs: None

Outputs: *name* — A buffer large enough (≥ 40 bytes) to hold the Operating System name.

Assumptions: None

Side Effects: None

Availability: All versions of the TI-89 / TI-92 Plus.

TI-89 / TI-92 Plus Differences: None

See Also: **AB_prodid, AB_serno**

Example:

```
char prodname[40];
AB_prodname(prodname);   /* get name of operating system software */
```

AB_serno

Declaration: BOOL **AB_serno** (char *serno*[])

Category(ies): Utilities

Description: Get the calculator's serial number. The serial number is returned in a string of the form "pphnn nnnnn vvvv".

pp	—	platform ID: 01 for TI-92 Plus, 03 for TI-89
h	—	hardware revision level
nn nnnnn	—	unique ID number
vvvv	—	verification number

All the above fields consist of hexadecimal digits.

Inputs: None

Outputs: *serno* — A buffer large enough (≥ 17 bytes) to hold the calculator's serial number.

Returns TRUE if the calculator has a serial number, FALSE if the serial number cannot be found in the calculator's certificate memory.

Assumptions: None

Side Effects: None

Availability: All versions of the TI-89 / TI-92 Plus.

TI-89 / TI-92 Plus Differences: None

See Also: **AB_prodid, AB_prodname**

Example:

```
char serno[17];
if (AB_serno(serno))
{
   /* Do something with serial number */
}
```

abs

Declaration: int **abs** (int *val*)

Category(ies): Utilities

Description: Returns the absolute value of an integer.

Inputs: *val* — Input value.

Outputs: Absolute value of input.

Assumptions: None

Side Effects: None

Availability: In the jump table on AMS 2.04 and higher.

Always available in the AMS library.

TI-89 / TI-92 Plus Differences: None

See Also: **labs**

Example: See **WinScrollH** for an example of calling **abs**.

The following code fragment will run on any version of AMS since it forces the **abs** routine from the AMS library to be linked in.

```
include "tiams.h"
#undef abs
int abs( int );
.
.
.
int val;
.
.
.
return( abs( val ) );
```

cmd_newprob

Declaration: void **cmd_newprob** (void)

Category(ies): Utilities

Description: Delete single letter variables a through z in the current folder, reset CAS arbitrary real and integer variable number counters to zero, turn off graphing of functions and data plots, then clear the graph screen, program I/O screen, table screen, and Home screen, and reset the error state.

This routine implements the TI-BASIC NewProb command.

Inputs: None

Outputs: None

Assumptions: None

Side Effects: None

Availability: On AMS 1.05 and higher.

TI-89 / TI-92 Plus Differences:

See Also: **cmd_clrdraw, cmd_clrgraph, cmd_clrhome, cmd_clrio**

Example:

```
cmd_newprob();
```

div

Declaration: div_t **div** (int *numerator,* int *denominator*)

Category(ies): Utilities

Description: Computes the quotient and remainder of the division of the numerator by the denominator. If the division is inexact, the resulting quotient is the integer of lesser magnitude that is the nearest to the algebraic quotient. If the result can be represented, then:

numerator = quot * *denominator* + rem

Inputs: *numerator* — Numerator.

denominator — Denominator.

Outputs: A structure (div_t) whose elements are quot (quotient) and rem (remainder) as defined above.

Assumptions: None

Side Effects: None

Availability: In the jump table on AMS 2.04 and higher.

Always available in the AMS library.

TI-89 / TI-92 Plus Differences: None

See Also: **ldiv**

Example: The following code fragment will run on any version of AMS since it forces the **div** routine from the AMS library to be linked in.

```
include "tiams.h"
#undef div
div_t div( int, int );
.
.
.
int num, denom;
div_t dt;
.
.
.
dt = div( num, denom );
```

EX_getArg

Declaration: EStackIndex **EX_getArg** (USHORT *j*)

Category(ies): Utilities

Description: Retrieves a pointer to the *jth* argument of an ASM program.

ASM programs can be called with arguments from TI-BASIC programs and from the Home screen author line. The arguments are evaluated and pushed on the estack from left to right. The left-most argument is deepest on the estack when the ASM program is called.

Inputs: *j* — Argument number to retrieve. 0 retrieves the right-most argument.

Outputs: Returns pointer to *jth* argument on the estack or NULL if *j* is past end of argument list.

Assumptions: None

Side Effects: None

Availability: All versions of the TI-89 / TI-92 Plus.

TI-89 / TI-92 Plus Differences: None

See Also: **next_expression_index**

Example:

```
EStackIndex e;
USHORT j;
 .
 .
 .
/* Process each arg until NULL arg encountered */
for (j = 0; (e = EX_getArg(j)) != NULL; j += 1)
{
   processArg(e);
}
```

EX_getBCD

Declaration: BOOL **EX_getBCD** (USHORT *n*, BCD16 * *bcd*)

Category(ies): Utilities, Direct Floating Point Operations

Description: Retrieves *nth* argument of an ASM program as a BCD value.

ASM programs can be called with arguments from TI-BASIC programs and from the Home screen author line. The arguments are evaluated and pushed on the estack from left to right. The left-most argument is deepest on the estack when the ASM program is called.

Inputs: *n* — Argument number to retrieve. 0 retrieves the right-most argument.

Outputs: *bcd* — Pointer to place where BCD value is returned.

Returns TRUE if *nth* argument exists and is a BCD value, otherwise FALSE.

Assumptions: None

Side Effects: None

Availability: All versions of the TI-89 / TI-92 Plus.

TI-89 / TI-92 Plus Differences: None

See Also: **EX_stoBCD**

Example:

```
BCD16 a, b;
   .
   .
   .
/* Get arguments into variables a and b */
if (EX_getBCD(1, &a) && EX_getBCD(0, &b))
{
   /* Perform calculation on a and b */
   .
   .
   .
}
else /* Could not retrieve a or b */
```

HToESI

Declaration: EStackIndex HToESI (HANDLE *h*)

Category(ies): Utilities

Description: Convert HANDLE to EStackIndex pointer. If the handle is not locked, **HToESI** must be done again after heap compression since the block of memory associated with the handle may have moved.

Inputs: *h* — Handle to a memory block containing the contents of a variable.

Outputs: EStackIndex pointer to the data type tag of the variable, which is the highest address of the variable data.

Assumptions: The handle is valid.

Side Effects: None

Availability: All versions of the TI‑89 / TI‑92 Plus.

TI‑89 / TI‑92 Plus Differences: None

See Also: **HeapDeref**

Example:

```
ptr = HToESI( h ); /* ptr=high address of memory block associated with handle h */
if( ESTACK( ptr ) == FLOAT_TAG )    /* Does block contain a tokenized
                                       floating-point number? */
   num = ESTACK_TO_FLOAT( ptr );    /* Yes, convert to BCD16 float in num. */
```

KB_89

Declaration: USHORT **KB_89** (USHORT *ch*)

Category(ies): Utilities

Description: Translates TI-92 Plus keys to TI-89 keys.

If this routine is running on a TI-89, then *ch* is returned unchanged. If this routine is running on a TI-92 Plus, the cursor arrow key codes are translated into TI-89 cursor arrow key codes.

This routine is useful for writing one source to run on both platforms.

Inputs: *ch* — Character from the CM_KEY_PRESS event or the **ngetchx** routine.

Outputs: Returns translated character.

Assumptions: None

Side Effects: None

Availability: All versions of the TI-89 / TI-92 Plus.

TI-89 / TI-92 Plus Differences: None

See Also: Not applicable.

Example:

```
switch (event->command)
{
  USHORT key;
  .
  .
  .
  case CM_KEY_PRESS:
    key = KB_89(event->info.keyInfo.keyCode);
    /* process translated key press */
  .
  .
  .
}
```

labs

Declaration: long **labs** (long *val*)

Category(ies): Utilities

Description: Returns the absolute value of a long integer.

Inputs: *val* — Input value.

Outputs: Absolute value of input.

Assumptions: None

Side Effects: None

Availability: In the jump table on AMS 2.04 and higher.

Always available in the AMS library.

TI-89 / TI-92 Plus Differences: None

See Also: **abs**

Example: The following code fragment will run on any version of AMS since it forces the **labs** routine from the AMS library to be linked in.

```
include "tiams.h"
#undef labs
long labs( long );
.
.
.
long lVal;
.
.
.
return( labs( lVal ) );
```

ldiv

Declaration: ldiv_t **ldiv** (long *numerator,* long *denominator*)

Category(ies): Utilities

Description: Computes the quotient and remainder of the division of the numerator by the denominator. If the division is inexact, the resulting quotient is the integer of lesser magnitude that is the nearest to the algebraic quotient. If the result can be represented, then:

numerator = quot * *denominator* + rem

Inputs: *numerator* — Numerator.

denominator — Denominator.

Outputs: A structure (ldiv_t) whose elements are quot (quotient) and rem (remainder) as defined above.

Assumptions: None

Side Effects: None

Availability: In the jump table on AMS 2.04 and higher.

Always available in the AMS library.

TI-89 / TI-92 Plus Differences: None

See Also: **div**

Example: The following code fragment will run on any version of AMS since it forces the **ldiv** routine from the AMS library to be linked in.

```
include "tiams.h"
#undef ldiv
ldiv_t ldiv( long, long );
.
.
.
long num, denom;
ldiv_t ldt;
.
.
.
ldt = div( num, denom );
```

NeedStack

Declaration: void **NeedStack** (short *StackNeeded*)

Category(ies): Utilities

Description: Throw an ER_MEMORY error if there is not at least *StackNeeded* bytes available on the hardware stack. The hardware stack is 16K in size. When a function calls another function the system will throw an ER_MEM_VIO error if there is not enough hardware stack to make the call.

A function may have a complex set of operations that may not be easily undone in a ONERR block. The function may also require that all of the operations do not fail due to a lack of hardware stack. In this case, the function can be started with a call to **NeedStack** to at least guarantee that the hardware stack will not overflow during the critical section of the function.

Inputs: *StackNeeded* — Minimum bytes of hardware stack required.

Outputs: May throw an ER_MEMORY error.

Assumptions: None

Side Effects: None

Availability: All versions of the TI-89 / TI-92 Plus.

TI-89 / TI-92 Plus Differences: None

See Also: None

Example: The **cmd_copyvar** function has a TRY/ENDTRY block in case some of the operations it executes fail due to lack of heap memory (or because of locked variables and the like). But since it also directly modifies elements of the symbol table, it calls **NeedStack** first. The call to **NeedStack** insures that none of the critical operations are partially completed due to a lack of hardware stack thus leaving the symbol table in an undefined state. The TI-BASIC interpreter uses the hardware stack to make recursive calls and so all TI-BASIC commands and functions cannot rely on the hardware stack being at any particular level.

```
NeedStack( 500 );
/* . . . prepare to do copy . . . */
TRY
   /* . . . do copy . . . */
ONERR
   /* . . . recover from any known error conditions . . . */
ENDTRY
```

strtod

Declaration: double **strtod** (const char * *nptr*, char ** *endptr*)

Category(ies): Utilities, Direct Floating Point Operations, Strings

Description: Converts the initial portion of the string pointed to by *nptr* into a double (BCD floating-point number), and if *endptr* is not a null pointer, sets the character pointer pointed to by *endptr* to the address of the first character following the set of characters converted.

The function skips leading whitespace, then accepts a sequence of characters matching a floating-point constant. The expected format of the floating-point constant consists of an optional sign character (+ or LF_NEGATIVE), followed by a sequence of one or more decimal digits optionally containing a decimal point, followed by an optional exponent. The optional exponent consists of LF_EXPONENT, followed by an optional sign, followed by one or more decimal digits.

If the first nonwhitespace character is not a sign, decimal digit or a decimal point, the conversion fails. If the string contains an initial subset of the optional exponent, and **strtod** reaches an unrecognized character before scanning at least one decimal digit, the conversion fails.

Inputs:

nptr — String to convert.

endptr — Points to a character pointer that will contain the address of the first character following the set of characters converted when the function returns (or null).

Outputs: Returns the converted double value (BCD floating-point number). If *endptr* is not a null pointer, the object pointed to by it is assigned the address of the first character of the string following the converted floating-point number. If the conversion fails, zero is returned and * *endptr* is assigned the value *nptr*.

If the result of the conversion would cause overflow, **errno** is set to ERANGE and returns plus or minus DBL_MAX. If the result of the conversion would cause underflow, **errno** is set to ERANGE and returns 0.

Assumptions: None

Side Effects: None

Availability: On AMS 2.00 and higher.

TI-89 / TI-92 Plus Differences: None

(continued)

strtod *(continued)*

See Also: **strtol**

Example:

```
void StrToList ( const char *str )
/* Create list of floats on estack from the string of floats */
{
   Access_AMS_Global_Variables;
   BCD16 num;
   unsigned char * start_ptr, * end_ptr;
   EStackIndex old_top, k;

   end_ptr = (unsigned char *)str;
   old_top = top_estack;
   push_quantum (END_TAG);
   while( *end_ptr )
   {  /* stop when end of string or fail */
      start_ptr = end_ptr;                    /* pt to next value to convert */
      num = strtod((const char *)start_ptr, (char **)&end_ptr);
      if( start_ptr == end_ptr )
         break;                               /* conversion failed */
      push_Float( num );
   }
   k = top_estack;                            /* point to last val */
   push_reversed_tail (k);                    /* reverse the order */
   delete_between (old_top, k);               /* delete the old copy */
   push_quantum (LIST_TAG);                   /* make it a list */
   return;
}
```

strtol

Declaration: long **strtol** (const char * *str*, char ** *end_ptr*, int *base*)

Category(ies): Utilities, Strings

Description: Converts the initial portion of the string pointed to by *str* into a long int, and if *end_ptr* is not a null pointer, sets the character pointer pointed to by *end_ptr* to the address of the first character following the set of characters converted. The function skips leading whitespace, accepts an optional sign character (+ or -), then scans a sequence of alphanumeric characters matching some integer represented in a base determined from the value of *base*.

If the value of *base* is zero, the conversion base is determined by the initial characters in the alphanumeric sequence. If the first character is 0 and the second character is x or X, hexadecimal conversion is performed. If the first character is 0 and the second character is not x or X, octal conversion is performed. Otherwise, decimal conversion is performed.

If the value of *base* is between 2 and 36, a sequence of letters and digits representing an integer of the indicated base is accepted. The letters a through z (or A through Z) represent the values 10 through 35. Only letters and digits with values less than the value of *base* are converted. If the value of *base* is 16 the sequence 0 followed by x or X may optionally precede the alphanumeric sequence.

Conversion of the string continues until a character is encountered that is not a digit in the indicated base. If *end_ptr* is not a null pointer, the object it points to is assigned a pointer to this first unconverted character.

Inputs:

str — String to convert.

end_ptr — Points to a character pointer that will contain the address of the first character following the set of characters converted when the function returns (or null).

base — Conversion base.

Outputs: Returns the converted long int value. If no conversion can be performed, zero is returned. If the conversion results in an overflow, **errno** is set to ERANGE. In this case, **strtol** returns LONG_MAX or LONG_MIN depending on the sign of the result. If the value of *base* is neither 0 nor in the range 2 to 36, the object pointed to by *end_ptr* is set to *str*, 0 is returned, and **errno** is set to EDOM.

(continued)

strtol *(continued)*

Assumptions: None

Side Effects: None

Availability: On AMS 2.00 and higher.

TI-89 / TI-92 Plus Differences: None

See Also: **strtod**

Example:

```
/*   Extract a long value from a given string using strtol().   */

char string = "-0x2FF is an odd number";
char *rest_of_string;
long val;
char buf[50];

val = strtol(string, &rest_of_string, 0);

sprintf(buf, "%d%s\n", val, rest_of_string);  /* buf= "-767 is an odd number" */
```

WordInList

Declaration: BOOL **WordInList** (WORD *Match*, WORD * *List*)

Category(ies): Utilities

Description: Return TRUE if *Match* is in *List*.

Inputs: *Match* — WORD to search for.

Outputs: *List* — Zero terminated list of WORDs to search.

Assumptions: None

Side Effects: None

Availability: All versions of the TI-89 / TI-92 Plus.

TI-89 / TI-92 Plus Differences: None

See Also: None

Example: The **QSysKey** routine is implemented using **WordInList** as shown below.

```
static const WORD SysKeyList[] = {KB_MATH, KB_CATLG, KB_CHAR, KB_CUSTOM, 0};

BOOL  QSysKey( WORD Key )
{
   return (WordInList(Key,(WORD *) SysKeyList));
}
```

Appendix A: System Routines — Variable Name Utilities

CheckReservedName

Declaration: WORD **CheckReservedName** (BYTE * *TokName*)

Category(ies): Variable Name Utilities

Description: Check for graph functions and other special names and return the type of the variable passed or zero otherwise. Note that this routine works the same as CheckSysFunc only a tokenized name is passed and it may also return R_SYSVAR.

Inputs: *TokName* — Pointer to tokenized variable name.

Outputs: Same return value as **CheckSysFunc** with the addition of R_SYSVAR for system variables.

Assumptions: None

Side Effects: None

Availability: All versions of the TI‑89 / TI‑92 Plus.

TI‑89 / TI‑92 Plus Differences: None

See Also: **CheckSysFunc**

Example: The rename command, once it determines the variable to rename is a folder uses **CheckReservedName** to insure the new name is not a reserved name as shown in the following code fragment.

```
EStackIndex origName, EStackIndex newName

.
.
.
if (FolderFind(origName) <= FL_OTHER) {
   if (CheckReservedName(newName))
      ER_THROW( ER_RESERVED );
   .
   .
   .
```

CheckSysFunc

Declaration: WORD **CheckSysFunc** (BYTE * *SymName*, short * *RetFuncNum*)

Category(ies): Variable Name Utilities

Description: Check for graph functions and other special names and return the type of the variable passed or zero otherwise.

Inputs: *SymName* — C pointer to the first letter of an untokenized symbol name (does not point to the zero byte terminator).

Outputs: *RetFuncNum* — If not NULL and *SymName* was a valid graph function name or C_COL then return the function/column number (1 . . . 99) at the WORD pointed to by *RetFuncNum*.

Returns —

zero	—	Not a system function or special name.
GR_FUNC	—	y1 . . . y99
GR_PAR	—	xt1 . . . xt99, yt1 . . . yt99
GR_POL	—	r1 . . . r99
GR_SEQ	—	u1 . . . r99
GR_3D	—	z1 . . . z99
GR_DE	—	y’1 . . . y’99
SEQ_INITC	—	ui1 . . . ui99
DE_INITC	—	yi1 . . . yi99
DE_FLDPIC	—	FldPic
UNIT_VAR	—	Name begins with a leading underscore.
C_COL	—	c1 . . . c99
R_REGEQ	—	RegEq
SOLVER_SYS_VARS	—	Exp, Eqn

Assumptions: None

Side Effects: None

Availability: All versions of the TI-89 / TI-92 Plus.

(continued)

CheckSysFunc *(continued)*

TI-89 / TI-92 Plus Differences: None

See Also: **SymSysVar, CheckReservedName**

Example: The **VarOpen** function uses **CheckSysFunc** to make sure that graph functions are not added to the list of functions for the user to select from since they can only be modified by the Y= editor or directly through the Define command. The following code fragment is from **VarOpen**.

```
SYM_ENTRY *SymPtr;
HANDLE hOpenVar;
.
.
.
if (0 == CheckSysFunc( SymPtr->Name, NULL ))
   if (!PopupAddText( hOpenVar, EOF, SymPtr->Name, 0 ))
      / * memory full error */
.
.
.
```

HSYMtoName

Declaration: BOOL **HSYMtoName** (HSYM *hSym*, BYTE * *SymName*)

Category(ies): Variable Name Utilities

Description: Given the HSYM of a symbol, convert that to a qualified symbol name and store it in *SymName,* which must be at least MAX_SYM_LEN bytes long.

Inputs: *hSym* — HSYM of symbol to convert.

Outputs: Return TRUE if *Handle* found, FALSE if not.

SymName — Fully qualified name (includes folder name, unless reserved name).

Assumptions: None

Side Effects: None

Availability: All versions of the TI‑89 / TI‑92 Plus.

TI‑89 / TI‑92 Plus Differences: None

See Also: **MakeHsym, VarRecall**

Example: See **FOpen**.

is_variable

Declaration: Boolean **is_variable** (EStackIndex *i*)

Category(ies): Variable Name Utilities, Algebra Utilities

Description: Determines whether *i* indexes a subscripted or unsubscripted variable.

Inputs: *i* — Index of the top tag of an internally-simplified expression.

Outputs: Returns TRUE if *i* indexes a subscripted or unsubscripted variable.

Assumptions: None

Side Effects: None

Availability: On AMS 2.00 and higher.

TI-89 / TI-92 Plus Differences: None

See Also: **is_complex_number, is_constant**

Example:

```
Boolean is_rational_expression (EStackIndex i)
/* i indexes an expression.
   Returns TRUE if it is a rational expression in all of its variables.
   Otherwise returns FALSE.
*/
{  EStackIndex j;
   for (;;)
      if (IS_NUMBER_OR_VARIABLE_TAG (ESTACK (i)) || is_variable (i))
         return TRUE;
      else if (ADD_TAG == ESTACK (i) || MULTIPLY_TAG == ESTACK (i))
            if (is_rational_expression (--i))
               i = next_expression_index (i);
            else
               return FALSE;
      else if (EXPONENTIATION_TAG == ESTACK (i))
            {  j = next_expression_index (--i);
               if (! IS_INTEGER_TAG (ESTACK (j)))
                  return is_constant (i);
            }
      else
         return is_constant (i);
}
```

StrToTokN

Declaration: BYTE * **StrToTokN** (BYTE * *StrSymName*, BYTE * *TokName*)

Category(ies): Variable Name Utilities

Description: Convert an ASCIIZ symbol name into tokenized format and store in *TokName*. Note that this routine does NOT handle reserved names. *TokName* must point to a buffer of MAX_SYM_LEN bytes. The tokenized named is stored there starting at the end of the buffer.

Inputs: *StrSymName* — ASCIIZ name to convert into tokenized format.

Outputs: *TokName* — Pointer to ASCIIZ symbol name in tokenized format.

Assumptions: This routine merely converts a name into tokenized format, it does not handle reserved names or check for the validity of the name passed to it and so in general **TokenizeSymName** should be used to tokenize symbol names.

Side Effects: None

Availability: All versions of the TI-89 / TI-92 Plus.

TI-89 / TI-92 Plus Differences: None

See Also: **TokenizeSymName**

Example: See **PopupClear**.

SymSysVar

Declaration: WORD **SymSysVar** (BYTE * *StrName*)

Category(ies): Variable Name Utilities

Description: Check if given variable is a system reserved variable stored in the symbol table (in the MAIN folder).

Inputs: *StrName* — Pointer to an ASCIIZ symbol name.

Outputs: Same return value as **CheckSysFunc**. But returns R_SYSVAR for SysData and RegCoef.

Assumptions: None

Side Effects: None

Availability: All versions of the TI‑89 / TI‑92 Plus.

TI‑89 / TI‑92 Plus Differences: None

See Also: **CheckSysFunc, TokenizeSymName**

Example: System reserved names always reside in the MAIN folder. In the **TokenizeSymName** function, if the TSF_FULLY_QUALIFIED flag is set and the variable does not already have a folder and it is not a system reserved name (which needs no folder name) then one is added. The check for the previous conditions is shown in the code fragment below.

```
if ((Flags & TSF_FULLY_QUALIFIED) && (strchr((char *)StrSymName, SYM_SEP) == NULL)
    && !SymSysVar(StrSymName)) {
   /* . . . add folder name . . . */
}
```

TokenizeSymName

Declaration: EStackIndex **TokenizeSymName** (BYTE * *StrSymName*, BOOL *Flags*)

Category(ies): Variable Name Utilities

Description: Convert a symbol name in standard C string format to a tokenized name and push it on the ESTACK, returning a pointer to the old **top_estack**. The caller must restore the ESTACK. Returns NULL if the symbol name was invalid.

Inputs:

StrSymName — Input symbol name in standard C format (points to first character of the string).

Flags —

TSF_ALLOW_RESERVED
Allow reserved names to be tokenized (otherwise they cause an error).

TSF_FULLY_QUALIFIED
Add current default folder to name if no folder specified in name (do not use if tokenizing stand alone folder names — to use for **cmd_newfold** and **cmd_delfold** for example).

TSF_PASS_ERRORS
Use ER_THROW on any errors instead of returning NULL.

Outputs: Original ESTACK pointer, NULL if error in name.

If Flags, TSF_PASS_ERRORS is set then may throw INVALID_PATHNAME_ERROR or ER_RESERVED errors.

Assumptions: This is the only routine that can be used to tokenize folder names. **TokenizeName** (in the FILE system) can only be used to tokenize variable names.

Side Effects: May cause heap compression.

Availability: All versions of the TI-89 / TI-92 Plus.

TI-89 / TI-92 Plus Differences: None

See Also: **TokenizeName**

Example: See **FolderCur** or **cmd_delvar** for an example. **FolderCur** contains code for a utility **TokenizeFoldName**, a routine like **TokenizeName** only it can handle folder names.

TokToStrN

Declaration: BOOL **TokToStrN** (BYTE * *StrSymName*, const BYTE * *TokPtr*)

Category(ies): Variable Name Utilities

Description: Convert a tokenized symbol name into an ASCIIZ string. This routine does not handle reserved names.

Inputs:

StrSymName — Pointer to buffer of MAX_SYM_LEN chars.

TokPtr — Pointer to a tokenized symbol name that is not a reserved variable.

Outputs: Return TRUE if symbol converted, FALSE if *TokPtr* does not point to a tag that represents a variable or is a reserved variable name.

StrSymName — ASCIIZ version of name.

Assumptions: None

Side Effects: None

Availability: All versions of the TI-89 / TI-92 Plus.

TI-89 / TI-92 Plus Differences: None

See Also: **StrToTokN, TokenizeSymName, TokenizeName**

Example: This is an example TI-BASIC callable routine that is passed the name of a file name. Since the FILE system uses ASCIIZ terminated names, **TokToStrN** is used to convert the input name into ASCIIZ format.

```
void _FCount(void)
{  Access_AMS_Global_Variables;
   char fName[MAX_SYM_LEN];

   if (!TokToStrN( (BYTE *) fName, top_estack ))
      ER_throw( EXPECTED_VAR_ERROR );
   .
   .
   .
   /* fName will be passed to the FILE routines like FOpen */

}
```

Appendix A: System Routines — Variables

See Also:

checkCurrent

Declaration: HSYM checkCurrent (EStackIndex *symName*, BYTE *tag*)

Category(ies): Variables, Data Utilities

Description: Make sure the given symbol exists and matches the requested tag type.

Inputs:
symName — Indexes the symbol name to check (points to the zero byte terminator).
tag — Requested tag type.

Outputs: HSYM of the symbol table entry if it exists and matches the requested tag type, otherwise returns H_NULL.

Assumptions: None

Side Effects: None

Availability: All versions of the TI-89 / TI-92 Plus.

TI-89 / TI-92 Plus Differences: None

See Also: **VarRecall, TokenizeSymName**

Example: The text editor when it is told to edit the current text variable uses **checkCurrent** to make sure the previous name entered by the user still exists and is a text variable. If it is not then it falls through to the code to prompt the user for a new text variable to edit.

```
static EStackIndex curTextNameSym;
HSYM hSym;
.
.
.
case AP_START_CURRENT:
   // Does current text exist?
   hSym = checkCurrent(curTextNameSym, TEXT_VAR_TAG);
   if (hSym != H_NULL)        /* text exist? */
      break;                  /* yes, open existing text ---> */

// Fall through to open new text

case AP_START_NEW:
   // Prompt for new text name
   hSym = VarNew((BYTE *)types);
```

cmd_archive

Declaration: void **cmd_archive** (EStackIndex *ePtr*)

Category(ies): Variables

Description: Archive one or more variables. This is the TI-BASIC archive command.

Inputs: *ePtr* — EStackIndex of variable name(s) to archive (terminated by an END_TAG).

Outputs: May throw these errors:

ER_RESERVED	—	Reserved names may not be archived.
ER_UNDEFINED_VAR	—	Variable not found.
ER_VAR_IN_USE	—	Variable in use.
ER_INVALID_VAR_REF	—	Cannot archive local variables.
ER_MEMORY	—	Not enough memory (Flash).

Assumptions: None

Side Effects: May cause heap compression.

Availability: On AMS 2.00 and higher.

TI-89 / TI-92 Plus Differences: None

See Also: **cmd_unarchiv**

Example: This function archives or unarchives a variable passed as a string name.

```
BOOL ArchiveVar( char *szBuf, BOOL archive )
{ Access_AMS_Global_Variables;
   EStackIndex oldTop;

   oldTop = top_estack;
   TRY
      push_quantum( END_TAG );
      if (TokenizeSymName( (BYTE *) szBuf, 0 ) == NULL)
         ER_THROW( INVALID_PATHNAME_ERROR );
      archive ? cmd_archive( top_estack ) : cmd_unarchiv( top_estack );
   ONERR
      top_estack = oldTop;
      ERD_dialog( errCode, FALSE );
      return FALSE;
   ENDTRY
   top_estack = oldTop;
   return(TRUE);
}
```

cmd_copyvar

Declaration: void **cmd_copyvar** (EStackIndex *OldSym*, EStackIndex *NewSym*)

Category(ies): Variables

Description: Copy a variable. This is the TI-BASIC command CopyVar. As specified in section **13.3. Managing Variables**, these routines should be used instead of the low-level Symbol Table routines.

Inputs:
OldSym — EStackIndex of source variable.
NewSym — EStackIndex of destination variable.

Outputs: May throw these errors (plus any that **VarRecall** or **VarStore** might throw):

ER_VAR_IN_USE	—	Source/destination is in-use.
ER_DUPLICATE_VAR_NAME	—	Destination same as source.
ER_UNDEFINED_VAR	—	Source variable not found.
ER_PROTECTED	—	Invalid copy to a system variable.
ER_MEMORY	—	Not enough memory for copying.

Assumptions: None

Side Effects: May cause heap compression.

Availability: On AMS 1.05 and higher.

TI-89 / TI-92 Plus Differences: None

See Also: **VarRecall, VarStore, TokenizeSym, TokenizeName**

Example: This example copies the variable *sourceName* to *destName*.

```
EStackIndex esSource, oldTop = top_estack;
BYTE sourceName[MAX_SYM_LEN], destName[MAX_SYM_LEN];

TRY
   if (TokenizeSymName( sourceName, 0 ) == NULL)
      ER_THROW( INVALID_PATHNAME_ERROR );
   esSource = top_estack;
   if (TokenizeSymName( destName, 0 ) == NULL)
      ER_THROW( INVALID_PATHNAME_ERROR );
   cmd_copyvar( esSource, top_estack );
   top_estack = oldTop; /* restore top of ESTACK */
ONERR
   top_estack = oldTop; /* restore top of ESTACK */
   PASS; /* pass error on up to caller */
ENDTRY
```

cmd_delfold

Declaration: void **cmd_delfold** (EStackIndex *foldName*)

Category(ies): Variables

Description: Delete one or more empty folders. This is the TI-BASIC command DelFold.

Inputs: *foldName* — EStackIndex of folder name(s) to delete (terminated by an END_TAG).

Outputs: May throw these errors:

ER_LOCKED	—	Folder is locked or in-use.
ER_ARG_MUST_BE_EMPTY_FOLDER	—	Folder must be empty.

Assumptions: None

Side Effects: None

Availability: On AMS 1.05 and higher.

TI-89 / TI-92 Plus Differences: None

See Also: **TokenizeSymName, cmd_newfold**

Example: Note that this example uses **TokenizeSymName** and not **TokenizeName** which does not work for folder names.

```
void DelFold( BYTE *foldName ) {
EStackIndex oldTop = top_estack; /* save top of ESTACK */

TRY
   push_quantum (END_TAG);  /* mark end of parameter list */
   if (TokenizeSymName( foldName, 0 ) == NULL) /* push tokenized name on ESTACK */
      ER_THROW( INVALID_PATHNAME_ERROR );
   cmd_delfold(top_estack);
   top_estack = oldTop; /* restore top of ESTACK */
ONERR
   top_estack = oldTop; /* restore top of ESTACK */
   PASS; /* pass error on up to caller */
ENDTRY
}
```

cmd_delvar

Declaration: void **cmd_delvar** (EStackIndex *sym*)

Category(ies): Variables

Description: Delete one or more variables. This is the TI-BASIC command DelVar. As specified in the Managing Variables section of the Memory Management chapter, these routines should be used instead of the low-level symbol table routines.

Inputs: *sym* — EStackIndex of variable name(s) to delete (terminated by an END_TAG).

Outputs: May throw these errors:

ER_LOCKED	—	Variable is locked or in-use.
ER_ARG_CANNOT_BE_FOLDER	—	Use **cmd_delfold** to delete folders.
ER_RESERVED	—	Reserved names cannot be deleted.

Assumptions: None

Side Effects: None

Availability: On AMS 1.05 and higher.

TI-89 / TI-92 Plus Differences: None

See Also: **TokenizeSymName**

Example: This example deletes the variable name in *varName.*

```
void DelVar( BYTE *varName ) {
EStackIndex oldTop = top_estack; /* save top of ESTACK */

TRY
   push_quantum (END_TAG);  /* mark end of parameter list */
   if (TokenizeSymName( varName, 0 ) == NULL) /* push tokenized name on ESTACK */
      ER_THROW( INVALID_PATHNAME_ERROR );
   cmd_delvar(top_estack);
   top_estack = oldTop; /* restore top of ESTACK */
ONERR
   top_estack = oldTop; /* restore top of ESTACK */
   PASS; /* pass error on up to caller */
ENDTRY
}
```

cmd_lock

Declaration: void **cmd_lock** (EStackIndex *sym*)

Category(ies): Variables

Description: Lock one or more variables. This is the TI-BASIC command Lock.

Inputs: *sym* — EStackIndex of variable name(s) to lock (terminated by an END_TAG).

Outputs: May throw these errors:

ER_VAR_IN_USE	—	Variable is in-use.
ER_UNDEFINED_VAR	—	Variable not found.
ER_RESERVED	—	Reserved or system names cannot be locked or unlocked.

Assumptions: None

Side Effects: None

Availability: On AMS 1.05 and higher.

TI-89 / TI-92 Plus Differences: None

See Also: **TokenizeSymName, cmd_unlock**

Example: This example locks or unlocks the variable name in *varName.*

```
void LockOp( BYTE *varName, BOOL LockIt ) {
EStackIndex volatile oldTop = top_estack; /* save top of ESTACK */

TRY
   push_quantum (END_TAG);  /* mark end of parameter list */
   if (TokenizeSymName( varName, 0 ) == NULL) /* push tokenized name on ESTACK */
      ER_THROW( INVALID_PATHNAME_ERROR );
   LockIt ? cmd_lock(top_estack) : cmd_unlock(top_estack);
   top_estack = oldTop; /* restore top of ESTACK */
ONERR
   top_estack = oldTop; /* restore top of ESTACK */
   PASS; /* pass error on up to caller */
ENDTRY
}
```

cmd_movevar

Declaration: void **cmd_movevar** (EStackIndex *varName*, EStackIndex *oldFolder*, EStackIndex *newFolder*)

Category(ies): Variables

Description: Move a variable from one folder to another. This is the TI-BASIC command MoveVar. If the destination folder (*newFolder*) does not exist it will be created.

Inputs:

varName — ESTACK index of variable to move.

oldFolder — Original folder.

newFolder — New folder.

Outputs: May throw these errors:

ER_FOLDER	—	Destination folder name is invalid.
ER_LOCKED	—	Destination variable is locked or in-use.
ER_RESERVED	—	Cannot move reserved names.
ER_MEMORY	—	Not enough memory to do the move.
ER_UNDEFINED_VAR	—	Source variable not found.

Assumptions: None

Side Effects: May cause heap compression.

Availability: On AMS 1.05 and higher.

TI-89 / TI-92 Plus Differences: None

See Also: **TokenizeSymName, TokenizeName, cmd_newfold**

Example: This example passes direct pointers to 'tokenized' names instead of using **TokenizeSymName**. Compare this to the **cmd_delvar** example which uses **TokenizeSymName** or the **VarRecall** example which uses **TokenizeName**.

```
const BYTE MainFolderName[6] = {0, 'm','a','i','n', 0};
const BYTE NewFolderName[10] =  {0,'z','f','o','l','d','e','r','1',0};
const BYTE fName[4] =  {0,'f','1',0};

if (FS_OK != FCreate("main\\f1", "DAT"))
   return;
/* "main" is always reserved but since this code by-passes TokenizeSymName we must
   make sure our name "zfolder1" is not reserved. Adding a digit to the end will
   insure it does not conflict with any new reserved names added by a localizer. Since
   a tokenized name consists of a zero byte followed by the name followed by a zero
   byte, we can just pass a pointer to the second zero byte as the tokenized name. */
cmd_movevar( (BYTE *)fName+3, (BYTE *)MainFolderName+5, (BYTE *)NewFolderName+9 );
```

cmd_newfold

Declaration: void **cmd_newfold** (EStackIndex *foldName*)

Category(ies): Variables

Description: Create a new folder. This is the TI-BASIC command NewFold.

Inputs: *foldName* — EStackIndex of folder name to create.

Outputs: May throw these errors:

ER_RESERVED	—	Reserved name.
INVALID_PATHNAME_ERROR	—	Name contains another folder name (nested folders are not allowed).
ER_INVALID_FOLDER_NAME	—	Invalid name.
ER_DUPLICATE_VAR_NAME	—	Folder already exists.
ER_MEMORY	—	Memory full.

Assumptions: None

Side Effects: May cause heap compression.

Availability: On AMS 1.05 and higher.

TI-89 / TI-92 Plus Differences: None

See Also: **TokenizeSymName, cmd_delfold**

Example: This example creates a temporary folder name, which may already exist and then creates a file in the folder, uses it, and then deletes the file.

```
EStackIndex oldTop;
FILES f1;

if (oldTop = TokenizeSymName((BYTE *) "ztemp1", 0)) {
   TRY
      cmd_newfold( top_estack ); /* create temporary folder */
   ONERR
      top_estack = oldTop;
      /* Ignore error if duplicate name (folder already exists) */
      if (errCode != ER_DUPLICATE_VAR_NAME)
         PASS;
   ENDTRY
   top_estack = oldTop;
   if (FS_OK == FOpen("FILE1", &f1, FM_WRITE, "DAT" )) {
      /* . . . use file . . . */
      FClose( &f1 );
   }
   FDelete( "FILE1" );  /* remove file */
}
```

cmd_rename

Declaration: void **cmd_rename** (EStackIndex *OldSym*, EStackIndex N*ewSym*)

Category(ies): Variables

Description: Rename a variable. This is the TI-BASIC command Rename. As specified in the Managing Variables section of the Memory Management chapter, these routines should be used instead of the low-level symbol table routines.

Inputs:

OldSym — EStackIndex of source variable.

NewSym — EStackIndex of destination variable.

Outputs: May throw these errors (plus any that **VarRecall** might throw):

ER_RESERVED	—	Illegal rename of a reserved name.
ER_LOCKED	—	Cannot rename to or from locked or in-use variables.
ER_DUPLICATE_VAR_NAME	—	*OldSym* and *NewSym* refer to same variable.
ER_DATATYPE	—	Conflict between source and destination data type.
ER_UNDEFINED_VAR	—	*OldSym* not found.
ER_MEMORY	—	Not enough memory to do rename.
ER_FOLDER	—	System variables can only reside in MAIN.

Assumptions: None

Side Effects: May cause heap compression.

Availability: On AMS 1.05 and higher.

TI-89 / TI-92 Plus Differences: None

See Also: **VarRecall, VarStore, TokenizeSymName**

(continued)

cmd_rename *(continued)*

Example: This example renames the variable *sourceName* to *destName*.

```
EStackIndex esSource, oldTop = top_estack;
BYTE sourceName[MAX_SYM_LEN], destName[MAX_SYM_LEN];

TRY
   if (TokenizeSymName( sourceName, 0 ) == NULL)
      ER_THROW( INVALID_PATHNAME_ERROR );
   esSource = top_estack;
   if (TokenizeSymName( destName, 0 ) == NULL)
      ER_THROW( INVALID_PATHNAME_ERROR );
   cmd_rename( esSource, top_estack );
   top_estack = oldTop; /* restore top of ESTACK */
ONERR
   top_estack = oldTop; /* restore top of ESTACK */
   PASS; /* pass error on up to caller */
ENDTRY
```

cmd_unarchiv

Declaration: void **cmd_unarchiv** (EStackIndex *ePtr*)

Category(ies): Variables

Description: Unarchive one or more variables. This is the TI-BASIC unarchiv command.

Inputs: *ePtr* — EStackIndex of variable name(s) to unarchive (terminated by an END_TAG).

Outputs: May throw these errors:

ER_RESERVED	—	Reserved names may not be archived.
ER_UNDEFINED_VAR	—	Variable not found.
ER_VAR_IN_USE	—	Variable in use.
ER_INVALID_VAR_REF	—	Cannot archive local variables.
ER_MEMORY	—	Not enough memory (RAM).

Assumptions: None

Side Effects: May cause heap compression.

Availability: On AMS 2.00 and higher.

TI-89 / TI-92 Plus Differences: None

See Also: **cmd_archive**

Example: See **cmd_archive**.

cmd_unlock

Declaration: void **cmd_unlock** (EStackIndex *sym*)

Category(ies): Variables

Description: Unlock one or more variables. This is the TI-BASIC command UnLock.

Inputs: *sym* — EStackIndex of variable names to unlock (terminated by an END_TAG).

Outputs: May throw these errors:

ER_VAR_IN_USE	—	Variable is in-use.
ER_UNDEFINED_VAR	—	Variable not found.
ER_RESERVED	—	Reserved or system name(s) cannot be locked or unlocked.
ER_MEMORY	—	Variables in Flash cannot be unlocked.

Assumptions: None

Side Effects: None

Availability: On AMS 1.05 and higher.

TI-89 / TI-92 Plus Differences: None

See Also: **TokenizeSymName, cmd_lock**

Example: See **cmd_lock**.

EX_stoBCD

Declaration: void **EX_stoBCD** (UCHAR * *name*, BCD16 * *bcd*)

Category(ies): Variables

Description: Stores BCD value into a variable in the calculator's symbol table.

Inputs:

name — Pointer to the name of a calculator variable where the BCD value should be stored. The name is a standard C zero-terminated string except it consists of unsigned characters since the names of variables in this calculator may contain international letters in the range '\xC0' to '\xFF'.

The name may be an eight-letter variable name or eight-letter folder name and variable name separated by a back-slash.

bcd — Pointer to BCD value to store.

Outputs: None

Assumptions: Variable named in *name* will be created if it does not already exist. Value in *name* will be overwritten unless it is a protected type.

Side Effects: May cause heap compression. Throws an error if variable *name* cannot be created or overwritten.

Availability: All versions of the TI-89 / TI-92 Plus.

TI-89 / TI-92 Plus Differences: None

See Also: **EX_getBCD**

Example:

```
BCD16 csum;
   .
   . /* calculate csum */
   .
EX_stoBCD((UCHAR *)"myfold" SYM_SEP_STR "csum", &csum);
```

push_assignment

Declaration: void **push_assignment** (EStackIndex *i*)

Category(ies): Variables

Description: Assigns a value to a symbol in the symbol table. It is the primary processing routine for the store operator and the Define command.

Inputs: *i* — EStackIndex of an assignment pair. The topmost expression is the symbol. The next expression below the symbol is the value to be stored. (The value can be specified in internal or external tokenized form.)

Outputs: The external tokenized form of the value is stored as the value of the symbol and is pushed on the top of the estack.

Assumptions: None

Side Effects: May expand expression stack or cause heap compression. May throw an error associated with simplification of the value or the specified symbol name.

Availability: On AMS 2.00 and higher.

TI-89 / TI-92 Plus Differences: None

See Also: None

Example:

```
/* The QR command performs the QR decomposition of a matrix. The command is given a
   matrix, a symbol name in which to store the Q matrix, and a symbol name to store
   the R matrix. Toward the end of the process the QR command has built the Q and R
   matrices on the expression stack with the R matrix top-most and the Q matrix below
   it. The estack indexes q and r reference these matrices. The estack indexes qsym
   and rsym reference the symbol names. The estack index old_top references the top of
   the stack when the QR command received control. The QR command takes the following
   steps to store the matrices to the appropriate symbols.
*/
.
.
.
push_expression (rsym);  /* push the symbol name above the R matrix */
push_assignment (top_estack);  /* store the R matrix into the symbol */
top_estack = q;  /* throw away the stack copy of the R matrix and symbol name */
push_expression (qsym);  /* push the symbol name above the Q matrix */
push_assignment (top_estack);  /* store the Q matrix into the symbol */
top_estack = old_top;  /* restore original stack position before return */
.
.
.
```

Appendix A: System Routines — Windows

CalcBitmapSize

Declaration: WORD **CalcBitmapSize** (BITMAP * *bitmapPtr*)

Category(ies): Windows

Description: Calculate a BITMAP size given a pointer to a BITMAP structure.

Inputs: *bitmapPtr* — Pointer to BITMAP structure.

Outputs: Size in bytes of a BITMAP pointed to by *bitmapPtr.*

```
typedef struct {
   WORD NumRows;
   WORD NumCols;
   BYTE Data[1];
}  BITMAP;
```

Assumptions: Compare this routine with **WinBitmapSize** and **WinBitmapSizeExt** which compute the size of a BITMAP if it were pulled out of a given window.

Side Effects: None

Availability: On AMS 2.00 and higher.

TI-89 / TI-92 Plus Differences: None

See Also: **WinBitmapSize, WinBitmapSizeExt, WinGetBitmap, WinPutBitmap**

Example:

```
/* Return HANDLE of the copy of a BITMAP or H_NULL if not enough memory */
HANDLE BitmapCopy( BITMAP *bPtr )
{  HANDLE hCopy;
   WORD size = CalcBitmapSize( bPtr );

   if (hCopy = HeapAlloc( size ))
      memcpy( HeapDeref(hCopy), bPtr, size );
   return hCopy;
}
```

DrawWinBorder

Declaration: void **DrawWinBorder** (WINDOW * *w*, const SCR_RECT * *Clip*)

Category(ies): Windows

Description: Draw the border for a window including the title if one exists. If the WF_ACTIVE flag is set in the WINDOW structure then draw a double border, otherwise draw a single border. Borders are not drawn on full screen windows.

Inputs:
w — Address of previously opened WINDOW structure.
Clip — Clipping rectangle (if none needed use &w.Window).

Outputs: None

Assumptions: None

Side Effects: None

Availability: All versions of the TI-89 / TI-92 Plus.

TI-89 / TI-92 Plus Differences: None

See Also: **WinActivate, WinDeactivate**

Example:

```
/* One of the things WinActivate does is set the given window to be active (and sets
   the currently active window to inactive) then calls DrawWinBorder to redraw the
   windows border.
*/
.
.
.
w->Flags |= WF_ACTIVE;
DrawWinBorder( w, &w->Window );
.
.
.
/* If an application must completely redraw its window (such as the example below
   where it is using the WF_DUP_SCR feature of WinOpen), it must also redraw its
   border
*/
case CM_WPAINT:
   DrawWinBorder( &appW, &appW.Window );
   WinBackupToScr( &appW );
   break;
```

MakeScrRect

Declaration: SCR_RECT * **MakeScrRect** (SWORD *x0*, SWORD *y0*, SWORD *x1*, SWORD *y1*, SCR_RECT * *sr*)

Category: Windows

Description: Given the upper left and lower right coordinates of a screen region and an address of a SCR_RECT, assign them to that SCR_RECT and return that address

Inputs:

x0, *y0* — Upper left coordinates of SCR_RECT.

x1, *y1* — Lower right coordinates of SCR_RECT.

sr — Address of a SCR_RECT structure.

Outputs: *sr*

Assumptions: Unlike **MakeWinRect**, the caller passes the address of the SCR_RECT to assign the coordinates to.

Note that WIN_RECTs and SCR_RECTs are not the same. WIN_RECTs are base on unsigned short values (WINDOW coordinates) whereas SCR_RECTs are base on BYTE values (screen coordinates)

Side Effects: None

Availability: All versions of the TI-89 and TI-92 Plus.

TI-89/TI-92 Plus Differences: None

See Also: MakeWinRect

Example: See **SetWinClip**.

MakeWinRect

Declaration: WIN_RECT * **MakeWinRect** (SWORD *x0*, SWORD *y0*, SWORD *x1*, SWORD *y1*)

Category(ies): Windows

Description: Given the upper left and lower right coordinates of a window return a pointer to a static WIN_RECT structure with those coordinates.

Inputs: *x0*, *y0* — Upper left coordinates of WIN_RECT.

x1, *y1* — Lower right coordinates of WIN_RECT.

Outputs: Pointer to a statically allocated WIN_RECT (make a copy if a permanent version is needed).

Assumptions: The address of the WIN_RECT returned is statically allocated and so cannot be shared.

Side Effects: None

Availability: All versions of the TI-89 / TI-92 Plus.

TI-89 / TI-92 Plus Differences: None

See Also: **MakeScrRect**

Example: See **SetWinClip** for an example where a copy is not made.

```
/* Create a WIN_RECT with MakeWinRect and save a copy in 'wr'. Without the memcpy
   we would be using a statically allocated WIN_RECT that would be overwritten on
   the next call to MakeWinRect.
*/
WIN_RECT wr;
memcpy(&wr, MakeWinRect(W_MIN_X,W_MIN_Y,W_MAX_X,W_MAX_Y), sizeof(WIN_RECT) );
```

ScrToWin

Declaration: WIN_RECT * **ScrToWin** (const SCR_RECT * *sr*)

Category(ies): Windows

Description: Convert a SCR_RECT structure to a WIN_RECT (return a pointer to statically allocated data!).

Inputs: *sr* — SCR_RECT to convert to WIN_RECT.

Outputs: Pointer to a statically allocated WIN_RECT (make a copy if a permanent version is needed — see example below).

Assumptions: The address of the WIN_RECT returned is statically allocated and so cannot be shared.

Side Effects: Note that WIN_RECTs and SCR_RECTs are not the same. WIN_RECTs are based on unsigned short values (WINDOW coordinates) whereas SCR_RECTs are based on BYTE values (screen coordinates).

Availability: All versions of the TI‑89 / TI‑92 Plus.

TI‑89 / TI‑92 Plus Differences: None

See Also: None

Example:

```
WIN_RECT wr;
WINDOW Win;
memcpy(&wr, ScrToWin(&Win->Client), sizeof(WIN_RECT) );
WinScrollV( &Win, &wr, -V_SCROLL );
```

SetWinClip

Declaration: void **SetWinClip** (WINDOW * *w*, SCR_RECT * *sr*)

Category(ies): Windows

Description: Set the clip region of a window from a Client based SCR_RECT.

Inputs:
w — WINDOW struct of a previously opened window.
sr — Client based coordinates of new clipping region. So (0, 0) is upper left coordinates of window, not screen.

Outputs: *sr* — The new clipping region of the window (screen based coordinates).

Assumptions: None

Side Effects: None

Availability: On AMS 2.00 and higher.

TI-89 / TI-92 Plus Differences: None

See Also: **MakeScrRect, WinOpen**

Example:

```
WINDOW w;
SCR_RECT clip1, oldClip;

if (WinOpen(&w, MakeWinRect(10,10,90,50), WF_TTY )) {
   WinActivate( &w );
   oldClip = w.Clip;
   SetWinClip( &w, MakeScrRect(2, 4,50,24,&clip1) );
   WinStrXY( &w, 0,0,"This is\nClipped\nAs you can see" );
   GKeyIn( NULL, 0 );
   w.Clip = oldClip;  /* clip region is SCREEN based, so to restore original clip
                         use the value saved before the call to SetWinClip. */
   WinStrXY( &w, 0,0,"This is\nClipped\nAs you can see" );
   GKeyIn( NULL, 0 );
   WinClose(&w);
}
```

WinActivate

Declaration: void **WinActivate** (WINDOW ** w*)

Category(ies): Windows

Description: Make the window the current active window. This will cause the following events:

1. The currently active window will be deactivated (its border will be changed to a single-line border).
2. The border for the window will switch to a double-line border. (Unless it is a full screen window.)
3. The current screen state will be reset to the current windows defaults (draw attributes, current X, Y location, . . .).
4. The window will be marked as visible (see **WinHide**).

Inputs: *w* — WINDOW struct of a previously opened window.

Outputs: None

Assumptions: None

Side Effects: Deactivates the currently active window.

Availability: All versions of the TI‑89 / TI‑92 Plus.

TI‑89 / TI‑92 Plus Differences: None

See Also: **WinBeginPaint, WinDeactivate, WinHide**

Example:

```
/* When a app gets the CM_ACTIVATE message it usually activates its window. */
case CM_ACTIVATE:
   EV_defaultHandler(e);
   WinBeginPaint( &appW );
   WinActivate( &appW );
   break;
```

WinAttr

Declaration: BYTE **WinAttr** (WINDOW ** w*, BYTE *Attr*)

Category(ies): Windows

Description: Set the attribute for the next write to a window (characters or lines).

Inputs: *w* — WINDOW struct of a previously opened window.

Attr — For characters (**WinChar[XY]**, **WinStr[XY]**) each attribute affects the background and foreground pixels defining the character differently:

Attribute	Background	Foreground
A_NORMAL	Unchanged	ON
A_REPLACE	OFF	ON
A_REVERSE	ON	OFF
A_SHADED	OFF	Every other pixel on
A_XOR	Unchanged	XOR'd with destination

For lines, ellipses and pixels (**WinLine[Ext, Rel, To]**, **WinEllipse**, **WinPixSet**) there are three supported attributes:

A_NORMAL	Destination pixels turned ON
A_REVERSE	Destination pixels turned OFF
A_XOR	Source pixels XOR'd with destination pixels

WinLine, **WinLineRel**, and **WinLineTo** also support A_THICK1 which draws a double thick NORMAL line.

Outputs: Original attribute for window.

Assumptions: Note the application of attributes as listed above and that some window drawing routines (**WinBitmapPut**, **WinFill**, **WinRect**, and **WinFillTriangle**) have a separate attribute that is passed each time.

Side Effects: Sets the default attribute for the window until changed.

Availability: All versions of the TI-89 / TI-92 Plus.

TI-89 / TI-92 Plus Differences None

See Also: **WinChar[XY], WinEllipse, WinLine[Ext, To, Rel], WinPixSet, WinStr[XY]**

Example: See **WinStr** for an example of using all of the character attributes.

```
WinAttr( &w, A_NORMAL );  /* normal draw */
WinStrXY( &w, 0, 0, "TEST" );  /* draw string */
GKeyIn( NULL, 0 ); /* wait on user */
WinAttr( &w, A_XOR );  /* XOR draw */
WinStrXY( &w, 0, 0, "TEST" );  /* erase string just drawn */
```

WinBackground

Declaration: void **WinBackground** (WINDOW ** w*, BYTE *Attr*)

Category(ies): Windows

Description: Change the current default attribute for the background of a window. The background is used when the window is cleared or scrolled (explicitly with **WinScrollH** or **WinScrollV** or implicitly when WF_TTY is set in the **WinOpen** and the screen is scrolled).

Inputs:

w — WINDOW struct of a previously opened window.

Attr — The valid values are and the result of all of the pixels are:

A_NORMAL	Turned on.
A_REVERSE	Turned off (cleared).
A_XOR	Pixels already in the window are flipped.

Outputs: None

Assumptions: The default background is A_REVERSE (clear).

Side Effects: Sets the background attribute for the window until changed.

Availability: All versions of the TI‑89 / TI‑92 Plus.

TI‑89 / TI‑92 Plus Differences: None

See Also: **WinClr, WinScrollH, WinScrollV**

Example:

```
WinBackground( &Win, A_NORMAL );  /* Window background is now solid */
WinClr( &Win );
WinBackground( &win, A_REVERSE ); /* back to normal */
```

WinBackupToScr

Declaration: void **WinBackupToScr** (WINDOW * *w*)

Category(ies): Windows

Description: If the given window was opened with the WF_DUP_SCR flag, copy the current backup screen (duplicate screen) to the real screen. When a window is opened with the WF_DUP_SCR flag, all output to that window is saved in a backup screen image (allocated in the heap). This routine copies the contents of that image to the real screen.

Inputs: *w* — WINDOW struct of a previously opened window with the WF_DUP_SCR flag.

Outputs: None

Assumptions: The WINDOW *w* was opened with the WF_DUP_SCR.

Side Effects: None

Availability: All versions of the TI-89 / TI-92 Plus.

TI-89 / TI-92 Plus Differences: None

See Also: **WinOpen**

Example: See **DrawWinBorder**.

WinBeginPaint

Declaration: void **WinBeginPaint** (WINDOW * *w*)

Category(ies): Windows

Description: Save the current screen state and sets up the screen to draw for the current window. **WinBeginPaint** is always paired with **WinEndPaint** which restores the screen state.

Inputs: *w* — WINDOW struct of a previously opened window.

Outputs: None

Assumptions: If an app has only one window then it can do a **WinBeginPaint** on its CM_ACTIVATE message and a **WinEndPaint** on its CM_DEACTIVATE message. If an app has multiple windows then routines that draw to separate windows should have **WinBeginPaint** and **WinEndPaint** pairs around them.

Side Effects: The screen is setup to draw for this window.

Availability: All versions of the TI-89 / TI-92 Plus.

TI-89 / TI-92 Plus Differences: None

See Also: **WinEndPaint**

Example:

```
CM_ACTIVATE:
   EV_defaultHandler(e);
   WinBeginPaint( &appW );
   WinActivate( &appW );
   WinStr( &appW, "Just activated\n" );
   break;
case CM_DEACTIVATE:
   WinEndPaint( &appW );
   break;
```

WinBitmapGet

Declaration: BOOL **WinBitmapGet** (WINDOW * *w*, const WIN_RECT * *WinRect*, BITMAP * *Bitmap*)

Category(ies): Windows

Description: Store a series of bytes (the size of which is defined by **WinBitmapSize**) defining a bitmap for a window into *Bitmap*. Return FALSE if the region defined by *WinRect* is outside of the window, TRUE if it is partially or entirely inside the window.

Inputs:

w — WINDOW struct of a previously opened window.

WinRect — The region of the WINDOW *w* to get as a bitmap.

Outputs: *Bitmap* — Output BITMAP (the first WORD is the number of rows, the second WORD is the number of columns and then the actual data follows as a series of bytes). Note that a BITMAP must always have one or more rows and one or more columns so its size is always at least five bytes long.

```
typedef struct {
WORD NumRows;
WORD NumCols;
BYTE Data[1];
} BITMAP;
```

Assumptions: *Bitmap* has enough room to store the retrieved BITMAP.

Side Effects: Note that there is a special case if the upper left x or y coordinate is less than zero. The region from the negative area to the lower right clip area is returned (use **WinBitmapSizeExt** in that case to determine the correct size of *Bitmap*). Otherwise the region returned is based from the Clip region.

Availability: All versions of the TI-89 / TI-92 Plus.

TI-89 / TI-92 Plus Differences: None

See Also: **WinBitmapPut, WinBitmapSize, WinBitmapSizeExt**

(continued)

WinBitmapGet *(continued)*

Example:

```
short x, y, i;
WINDOW w1;
WIN_RECT r1 = {119,0,239,119}, r2 = {72,16,89,39};
BITMAP *Bitmap;
HANDLE h;
char Str[20];

if (WinOpen( &w1, &r1, WF_DUP_SCR )) {
   WinActivate( &w1 );
   WinClr( &w1 );
   Str[18] = '\0';
   for( i = 0; i <= 9; i++) {
      memset( Str, '0' + i, 18 );
      WinStrXY( &w1, 6, i*8+8, Str );
   }
   if ((h = HeapAlloc(WinBitmapSize(&w1,&r2))) != H_NULL) {
      Bitmap = (BITMAP *) HeapDeref(h);
      WinBitmapGet( &w1, &r2, Bitmap );
      for (x = 12, y = 72; x <= 138; x += 18, y += 8)
         WinBitmapPut( &w1, x, y, Bitmap, A_REPLACE );
      HeapFree( h );
      GKeyIn(NULL,0);
   }
   WinClose( &w1 );
}
```

WinBitmapPut

Declaration: void **WinBitmapPut** (WINDOW * *w*, SWORD *x0*, SWORD *y0*, BITMAP * *Bitmap*, WORD *Attr*)

Category(ies): Windows

Description: Store a bitmap into a window.

Inputs:

w	—	WINDOW struct of a previously opened window.	
x0, *y0*	—	WINDOW coordinates of location to store *Bitmap*.	
Bitmap	—	BITMAP to store.	
Attr	—	A_REPLACE	Replace the destination region with the source bitmap.
		A_REVERSE	Replace the destination region with the inverse of the source bitmap.
		A_XOR	Exclusive-OR the source bitmap into the destination region.
		A_OR	OR the source bitmap into the destination region.
		A_AND	AND the source bitmap into the destination region.

Outputs: None

Assumptions: None

Side Effects: None

Availability: All versions of the TI‑89 / TI‑92 Plus.

TI‑89 / TI‑92 Plus Differences: None

See Also: **WinBitmapGet, WinBitmapSize, WinBitmapSizeExt**

Example: See **WinBitmapGet**.

WinBitmapSize

Declaration: WORD **WinBitmapSize** (WINDOW * *w*, const WIN_RECT * *WinRect*)

Category(ies): Windows

Description: Return the size in bytes of a bitmap for a window (may be smaller than the size of the region defined by *WinRect* due to clipping). This size includes the data for the bitmap and the header (2 WORDs).

Inputs:
w — WINDOW struct of a previously opened window.
WinRect — Defines region of BITMAP to determine size of.

Outputs: Size of BITMAP defined by *WinRect* taking clipping into account.

Assumptions: **WinBitmapSize** will clip any negative coordinates to zero (use **WinBitmapSizeExt** if negative coordinates are to be accounted for in the return value).

Side Effects: None

Availability: All versions of the TI-89 / TI-92 Plus.

TI-89 / TI-92 Plus Differences: None

See Also: **CalcBitmapSize, WinBitmapGet, WinBitmapPut, WinBitmapSizeExt**

Example: See **WinBitmapGet**.

WinBitmapSizeExt

Declaration: WORD **WinBitmapSizeExt** (WINDOW * *w*, const WIN_RECT * *WinRect*)

Category(ies): Windows

Description: Return the size in bytes of a bitmap for a window (may be smaller than the size of the region defined by *WinRect* due to clipping). This size includes the data for the bitmap and the header.

Inputs:
w — WINDOW struct of a previously opened window.
WinRect — Defines region of BITMAP to determine size of.

Outputs: Size of BITMAP defined by *WinRect* taking clipping into account and negative coordinates.

Assumptions: **WinBitmapSizeExt** will take negative coordinates into account, use **WinBitmapSize** if negative coordinates should be clipped to zero.

Side Effects: None

Availability: On AMS 2.00 and higher.

TI-89 / TI-92 Plus Differences: None

See Also: **CalcBitmapSize, WinBitmapGet, WinBitmapPut, WinBitmapSize**

Example:

```
WINDOW w1;
HANDLE h;
WORD sNormal, sExt;
BITMAP *Bitmap;
WIN_RECT r2 = {-16,-6,30,30};

/*.. assume w1 already opened */
sNormal = WinBitmapSize( &w1, &r2 );
sExt = WinBitmapSizeExt( &w1, &r2 ); /* will be > sNormal */
/* Since r2 has negative coordinates we must use the value from WinBitmapSizeExt to
   allocate the bitmap for this example because the WinBitmapGet call will include
   these negative portions of the window (the values stored in the bitmap outside the
   window may include the windows border, use SetWinClip to exclude the border if
   needed).
*/
if ((h = HeapAlloc(sExt)) != H_NULL) {
   Bitmap = (BITMAP *) HeapDeref(h);
   WinBitmapGet( &w1, &r2, Bitmap );
   WinBitmapPut( &w1, 0, 0, Bitmap, A_REPLACE );
   HeapFree( h );
}
```

WinChar

Declaration: void **WinChar** (WINDOW * *w*, char *c*)

Category(ies): Windows

Description: Write a character to a window at the current pen position (pixel based) using the current attribute (set with **WinAttr**) and current font (**WinFont**). If the window is in WF_TTY mode the current X, Y location is updated; and newline ('\r' or '\n') and form feed ('\f') characters are handled.

Inputs: *w* — WINDOW struct of a previously opened window.

c — Character to draw to window.

Outputs: None

Assumptions: The default attribute (A_NORMAL) can be changed with **WinAttr**; the default font (F_6x8) can be changed with **WinFont**; the current window position is set with **WinMoveTo** or **WinMoveRel**.

The supported values for character attributes are: A_NORMAL, A_REVERSE, A_XOR, A_SHADED, A_REPLACE. See **WinAttr** for a detailed description of the character attributes.

Side Effects: None

Availability: All versions of the TI-89 / TI-92 Plus.

TI-89 / TI-92 Plus Differences: None

See Also: **WinAttr, WinFont, WinOpen, WinCharXY, WinMoveTo, WinMoveRel, WinStr, WinStrXY**

Example:

```
WINDOW wTTY;
if (WinOpen( &wTTY, MakeWinRect(10,10,90,90), WF_TTY)) {
   WinActivate( &wTTY );
   WinChar( &wTTY, '\f' );   /* same as WinClr( &wTTY ); */
   WinFont( &wTTY, F_8x10 );
   WinChar( &wTTY, LF_LE );
   WinAttr( &wTTY, A_REVERSE );
   WinChar( &wTTY, '\n' );   /* newline */
   WinChar( &wTTY, '2' );
   WinMoveTo( &wTTY, 20, 20 );              /* move to specific location */
   WinAttr( &wTTY, A_REPLACE );
   WinChar( &wTTY, 'X' );
   ngetchx();
   WinMoveRel( &wTTY, -8, 0 );              /* backup one char */
   WinAttr( &wTTY, A_XOR );   /* XOR mode */
   WinChar( &wTTY, 'X');   /* wipe out previous char */
   ngetchx();
   WinClose( &wTTY );
}
```

WinCharXY

Declaration: void **WinCharXY** (WINDOW * *w*, WIN_COORDS *x*, WIN_COORDS *y*, char *c*, short *Count*)

Category(ies): Windows

Description: Write *Count* number of characters *c* to a window at position *x, y* (pixel based) using the current attribute (set with **WinAttr**) and current font (**WinFont**). If the window is in WF_TTY mode the current X, Y location is updated; and newline ('\r' or '\n') and form feed ('\f') characters are handled.

Inputs:

w — WINDOW struct of a previously opened window.

x, y — Window position to write to.

c — Character to draw.

Count — Number of characters to write.

Outputs: None

Assumptions: The default attribute (A_NORMAL) can be changed with **WinAttr**. The default font (F_6x8) can be changed with **WinFont**.

The supported values for character attributes are: A_NORMAL, A_REVERSE, A_XOR, A_SHADED, A_REPLACE. See **WinAttr** for a detailed description of the character attributes.

Side Effects: None

Availability: All versions of the TI‑89 / TI‑92 Plus.

TI‑89 / TI‑92 Plus Differences: None

See Also: **WinAttr, WinFont, WinOpen, WinChar, WinStr, WinStrXY**

(continued)

WinCharXY *(continued)*

Example:

```
/* Draw a string to a window, wrapping on spaces or newlines. Return the height in
   pixels of the text just drawn. If wwFlags.WWF_DRAW is not set then just returns
   height and does not draw. Set wwFlags.WWWF_WRAP_ON_COMMAS to also wrap on commas.
*/
enum winWriteFlags { WWF_DRAW = 0x01, WWF_WRAP_ON_COMMAS=0x2 };
short WinStrXYWrap( WINDOW *w, WIN_COORDS x, WIN_COORDS y, char *Str, WORD wwFlags )
{  char *bStr, c;
   short numChars, pixWid, startX, SysFontY;

   startX = x;
   SysFontY = ((w->CurFont == F_4x6) ? 6 : (w->CurFont == F_6x8 ? 8 : 10));
   do {
      bStr = Str;
      numChars = 0;
      while (c = *bStr) {
         if (0xA == c)
            break;
         if (' ' == c)
         if (numChars)
            break;
         numChars++;
         if ((wwFlags & WWF_WRAP_ON_COMMAS) && ',' == c)
            break;
         bStr++;
      }
      if (numChars) {
         pixWid = DrawStrWidthP( Str, numChars, w->CurFont );
         if (x + pixWid >= WinWidth(w)) {
            x = startX;
            y += SysFontY;
            if (' ' == *Str)  {
               if (numChars)
               numChars--;
               Str++;
            }
         }
         while (numChars--) {
            c = *Str++;
            if (wwFlags & WWF_DRAW)
               WinCharXY( w, x, y, c, 1 );
            x += FontCharWidth( c );
         }
      }  else {
         x = startX;
         y += SysFontY;
         Str++;
      }
   }  while (*bStr);
   return y;
}
```

WinClose

Declaration: void **WinClose** (WINDOW * *w*)

Category(ies): Windows

Description: Close a window, releasing any memory assigned to it and activating the next window.

Inputs: *w* — WINDOW struct of a previously opened window.

Outputs: None

Assumptions: All windows opened with **WinOpen** must be closed with either **WinClose** or **WinRemove**.

Side Effects: Windows are kept in a linked list. When one window is closed, the next window in the linked list is activated. This may mean redrawing portions of the screen in order to keep it up-to-date. If this is not wanted (as per virtual windows) then use **WinRemove** (w, FALSE) instead.

Availability: All versions of the TI‑89 / TI‑92 Plus.

TI‑89 / TI‑92 Plus Differences: None

See Also: **WinOpen, WinRemove**

Example: See **WinBitmapGet**, **WinChar**, and **WinDupStat**.

WinClr

Declaration: void **WinClr** (WINDOW * *w*)

Category(ies): Windows

Description: Clear the client area of the current window (using the current clip region). Reset the current x, y position to the home of the client region. The current background attribute is used to fill the client area.

Inputs: *w* — WINDOW struct of a previously opened window.

Outputs: None

Assumptions: The default background attribute is A_REVERSE (clear). A_NORMAL produces a solid background. A_XOR keeps what is on the screen but flips every bit.

Side Effects: None

Availability: All versions of the TI‑89 / TI‑92 Plus.

TI‑89 / TI‑92 Plus Differences: None

See Also: **WinBackground**. See **WinFill** to clear specific regions.

Example:

```
WinStr( &w, "TEST STRING" );
WinBackground( &w, A_XOR );
WinClr( &w );   /* reverse entire window (including existing images) */
ngetchx();   /* wait for user */
WinBackground( &w, A_REVERSE );   /* back to white background (default) */
WinClr( &w );   /* normal clear window */
```

WinDeactivate

Declaration: void **WinDeactivate** (WINDOW * *w*)

Category(ies): Windows

Description: Deactivate a window (changes its border to single-line unless full screen).

Inputs: *w* — WINDOW struct of a previously opened window.

Outputs: None

Assumptions: **WinDeactivate** is really only needed if an app has multiple windows. The purpose is to provide a visual clue to the user that a particular window has lost the current focus and that another window (which will be activated with **WinActivate**) has received the focus. When **WinActivate** is called, the window with the current focus (the last one to do a **WinActivate**) is automatically deactivated with **WinDeactivate** and so it is not necessary to explicitly call **WinDeactivate**.

Side Effects: None

Availability: All versions of the TI-89 / TI-92 Plus.

TI-89 / TI-92 Plus Differences: None

See Also: **WinActivate**

Example:

```
WINDOW win1, win2;
if (WinOpen(&win2, MakeWinRect(5,5,70,50), WF_ROUNDEDBORDER|WF_TTY)) {
   WinActivate( &win2 );   /* just to draw the border */
   if (WinOpen(&win1, MakeWinRect(75,5,140,50), WF_TTY)) {
      WinActivate( &win1 ); WinClr( &win1 );
      WinStr( &win1, "ACTIVE\nWINDOW" );
      GKeyIn( NULL, 0 );
      WinClr( &win1 ); WinStr( &win1, "NOT\nACTIVE");
      WinDeactivate( &win1 );
      WinStr( &win2, "NO WINDOWS\nACTIVE" );
      GKeyIn( NULL, 0 );
      WinClose( &win2 );
   }
   WinClose( &win1 );
}
```

WinDupStat

Declaration: BOOL **WinDupStat** (WINDOW * *w*, BOOL *Stat*)

Category(ies): Windows

Description: Turn the duplicate status on (*Stat* = TRUE) or off (*Stat* = FALSE). When the duplicate status is turned off, all writes to a window go only to the screen. When turned on, all writes go to both the screen and the backup window. This only applies to windows created with the WF_DUP_SCR flag set.

Inputs:
w — WINDOW struct of a previously opened window.
Stat — TRUE to turn duplicate writes on, FALSE to turn them off.

Outputs: If WF_DUP_SCR is set then returns the old status.

Assumptions: The WINDOW *w* was opened with the WF_DUP_SCR flag.

Side Effects: Duplicate writes slow down all writes to windows with WF_DUP_SCR set.

Availability: All versions of the TI-89 / TI-92 Plus.

TI-89 / TI-92 Plus Differences: None

See Also: **WinOpen, WinBackupToScr**

Example:

```
WINDOW w;
if (WinOpen(&w, MakeWinRect(10,10,90,80), WF_DUP_SCR|WF_TTY )) {
   WinActivate( &w );
   WinStr( &w, "THIS IS DUPLICATED\n" );   /* written to screen & backup */
   WinEllipse( &w, 40,40,20,20 );   /* ditto */
   WinDupStat( &w, FALSE );
   WinStr( &w, "THIS GOES ONLY TO THE SCREEN");   /* does not go to backup */
   ngetchx();
   WinBackupToScr( &w );   /* last WinStr text will be gone */
   ngetchx();
   WinClose( &w );
}
```

WinEllipse

Declaration: void **WinEllipse** (WINDOW * *w*, WIN_COORDS *x0*, WIN_COORDS *y0*, WIN_COORDS *a0*, WIN_COORDS *b0*)

Category(ies): Windows

Description: Draw an ellipse in a window with center at (*x0*, *y0*) and major/minor axes (*a0*, *b0*). The ellipse is drawn in the current attribute (A_NORMAL, A_REVERSE, A_XOR) as set by **WinAttr**.

Inputs:

w — WINDOW struct of a previously opened window.

x0, *y0* — Center of ellipse in window coordinates.

a0, *b0* — Major/minor axes size in pixels.

Outputs: None

Assumptions: Current attribute (set by **WinAttr**) is A_NORMAL (set), A_REVERSE (clear), or A_XOR.

Side Effects: The pixels on the TI‑89 and the TI‑92 are square so setting a0 and b0 equal will draw a circle on both platforms.

Availability: All versions of the TI‑89 / TI‑92 Plus.There is no routine to draw a filled ellipse.

TI‑89 / TI‑92 Plus Differences: None

See Also: **WinAttr**

Example: See **WinDupStat** for an example that draws an ellipse to the screen and to a second backup screen (with one call).

WinEndPaint

Declaration: void **WinEndPaint** (WINDOW * *w*)

Category(ies): Windows

Description: Restore the current screen state that was saved with the corresponding **WinBeginPaint**. **WinBeginPaint** is always paired with **WinEndPaint**.

Inputs: *w* — WINDOW struct of a previously opened window.

Outputs: None

Assumptions: **WinBeginPaint** was previously called on this window.

Side Effects: Screen state restored.

Availability: On AMS 2.00 and higher.

TI-89 / TI-92 Plus Differences: None

See Also: **WinBeginPaint**

Example: See **WinBeginPaint** for a **WinBeginPaint**/**WinEndPaint** example.

WinFill

Declaration: void **WinFill** (WINDOW * *w*, const WIN_RECT * *rWin*, BYTE *Attr*)

Category(ies): Windows

Description: Fill a region of a window with a given attribute.

Inputs:

w — WINDOW struct of a previously opened window.

rWin — Window coordinates of rectangle to fill.

Attr — Attribute to use — A_NORMAL (set), A_REVERSE (cleared), A_XOR (all pixels are reversed).

Outputs: None

Assumptions: None

Side Effects: None

Availability: All versions of the TI-89 / TI-92 Plus.

TI-89 / TI-92 Plus Differences: None

See Also: **WinFillTriangle**

Example:

```
WIN_RECT wr = { 24, 16, 42, 32 };

WinFont( &appW, F_6x8 );
WinStrXY( &appW, 24, 16, "ABC" );   /* draw some text */
WinStrXY( &appW, 24, 24, "DEF" );
WinFill( &appW, &wr, A_XOR );   /* highlight text */
ngetchx();   /* wait for user */
WinFill( &appW, &wr, A_XOR );   /* un-highlight text */
```

WinFillTriangle

Declaration: void **WinFillTriangle** (WINDOW * *w*, WIN_COORDS *ax*, WIN_COORDS *ay*, WIN_COORDS *bx*, WIN_COORDS *by*, WIN_COORDS *cx*, WIN_COORDS *cy*, BYTE *Attr*)

Category(ies): Windows

Description: Draw a filled triangle in a window. The fill includes the borders of the triangle.

Inputs:

w	—	WINDOW struct of a previously opened window.
ax, ay, bx, by, cx, cy	—	Coordinates of triangle.
Attr	—	A_NORMAL (set) or A_REVERSE (cleared).

Outputs: None

Assumptions: A_XOR is not supported.

Side Effects: None

Availability: All versions of the TI‑89 / TI‑92 Plus.

TI‑89 / TI‑92 Plus Differences: None

See Also: **WinFill**

Example: The 3D grapher in HIDDEN SURFACE mode uses this routine to shade the graph using A_REVERSE if the surface is visible and A_NORMAL if it is hidden (by splitting the graph into 6-sided polygons and splitting those into triangles).

WinFont

Declaration: void **WinFont** (WINDOW * *w*, BYTE *Font*)

Category(ies): Windows

Description: Change the current text font for a window. All subsequent characters written to the window will use this font.

Inputs:

w — WINDOW struct of a previously opened window.

Font — F_4x6, F_6x8, and F_8x10. The 4 x 6 font is a proportional font while the 6 x 8 and 8 x 10 fonts are fixed-width.

Outputs: None

Assumptions: None

Side Effects: Sets the current text font for the window until changed.

Availability: All versions of the TI‑89 / TI‑92 Plus.

TI‑89 / TI‑92 Plus Differences: Due to the smaller size of the TI‑89 display, the F_8x10 font is normally not used on that platform.

See Also: **WinChar[XY], WinStr[XY], DrawStrWidth**

See section **4.4. Fonts** for more information about fonts on the TI-89 / TI-92 Plus and a table of the font characters.

Example: See **WinChar**, **WinCharXY**, **WinFill,** and **WinStr**.

WinHeight

Declaration: WIN_COORDS **WinHeight** (WINDOW * *w*)

Category(ies): Windows

Description: Return the height of the client (drawable) area of a window.

Inputs: *w* — WINDOW struct of a previously opened window.

Outputs: Height of window's client region.

Assumptions: The window region is the region that was defined when the window was created with **WinOpen**. If the window is full screen (not counting the status bar which may not be overlapped), then the client region is equal to the window region. The client region is reduced by adding borders or a title to a window.

Side Effects: None

Availability: All versions of the TI‑89 / TI‑92 Plus.

TI‑89 / TI‑92 Plus Differences: None

See Also: **WinOpen, WinWidth**

Example:

```
/* This example uses the width and height of its window (appW) so that it can
   fill the entire window with the letter 'A'. */
WIN_COORDS wWidth, wHeight, y;

wWidth = WinWidth( &appW );
wHeight = WinHeight( &appW );
WinClr( &appW );
WinFont( &appW, F_6x8 );
for (y = 0; y <= wHeight - 8; y += 8 )
   WinCharXY( &appW, 0, y, 'A', (wWidth / 6) );
```

WinHide

Declaration: void **WinHide** (WINDOW * *w*)

Category(ies): Windows

Description: Hide a window (mark it as not-visible so that it is never activated by the system) and update the screen.

Inputs: *w* — WINDOW struct of a previously opened window.

Outputs: None

Assumptions: When a window is activated (**WinActivate**) or when it is opened (unless the WF_VIRTUAL flag is passed to **WinOpen**) it is marked as visible. All windows in the system are kept in a linked list. When a window in the system is closed, the next visible window in the system is activated and becomes the currently active window. Since virtual windows are never displayed on the screen they are never considered visible. An app's main window is always visible since that is the only view the user has of the app. An app may open other windows that it does not want to ever be activated. In that case, use **WinHide** so that they will never be activated by the system.

Side Effects: The given window will not be activated by the system but writes to it still go to the screen (unless it is a virtual window).

Availability: All versions of the TI-89 / TI-92 Plus.

TI-89 / TI-92 Plus Differences: None

See Also: **WinActivate, WinOpen, WinReOpen**

Example:

```
/* The Home screen has two overlapping windows (wHome and wPrgmIO). When it deactivates
   the wPrgmIO window it hides that window and activates its main window - wHome. */
WinDeactivate(&wPrgmIO);
WinHide(&wPrgmIO);
WinActivate(&wHome);
```

WinHome

Declaration: void **WinHome** (WINDOW * *w*)

Category(ies): Windows

Description: Move the pen location for a window to the home position. Note that in WF_TTY mode this is (1, 1) otherwise it is (0, 0).

Inputs: *w* — WINDOW struct of a previously opened window.

Outputs: None

Assumptions: None

Side Effects: None

Availability: All versions of the TI-89 / TI-92 Plus.

TI-89 / TI-92 Plus Differences: None

See Also: **WinMoveTo**

Example: This example handles simple TTY type input. It assumes *appW* has already been opened in TTY mode.

```
WORD Key, x, y;
SCR_RECT scrRect, tempRect;

WinAttr( &appW, A_REPLACE );
WinFont( &appW, F_6x8 );
do {
   x = appW.CurX; y = appW.CurY;
   ClientToScr( &appW.Client, MakeScrRect(x,y,x+5,y+7,&tempRect), &scrRect );
   Key = GKeyIn( &scrRect, 0 );
   if (Key <= 0xFF)
      WinChar( &appW, Key );
   else switch (Key) {
      case KB_BEGIN:  /* 2nd LEFT */
         WinHome( &appW ); break;
      case KB_LEFT:
         WinMoveRel( &appW, -6, 0 ); break;
      case KB_RIGHT:
         WinMoveRel( &appW,  6, 0 ); break;
      case KB_UP:
         WinMoveRel( &appW,  0, -8 ); break;
      case KB_DOWN:
         WinMoveRel( &appW,  0,  8 ); break;
   }
}  while (Key != KB_ESC);
WinClr( &appW );
```

WinLine

Declaration: void **WinLine** (WINDOW * *w*, const WIN_RECT * *Line*)

Category(ies): Windows

Description: Draw a line in a window using a WIN_RECT to define the end-points. The line is drawn in the current attribute (set with **WinAttr**).

Inputs: *w* — WINDOW struct of a previously opened window.

Line — Endpoints of line to draw.

Outputs: None

Assumptions: The valid line attributes are: A_NORMAL (set), A_REVERSE (clear), A_XOR and A_THICK1 (double thick line).

Side Effects: None

Availability: All versions of the TI‑89 / TI‑92 Plus.

TI‑89 / TI‑92 Plus Differences: None

See Also: **WinAttr, WinLineExt**

Example:

```
WIN_RECT wr = {0, 2, 90, 2};
WinAttr( &appW, A_NORMAL );   /* draw a normal line */
WinLine( &appW, &wr );
WinAttr( &appW, A_REVERSE );   /* turn pixels off */
WinLine( &appW, &wr );
WinAttr( &appW, A_XOR );   /* XOR with destination */
WinLine( &appW, &wr );
WinAttr( &appW, A_THICK1 );   /* double thick line */
WinLine( &appW, &wr );
WinAttr( &appW, A_NORMAL );   /* back to normal in case we draw characters */
```

WinLineExt

Declaration: void **WinLineExt** (WINDOW * *w*, const WIN_RECT * *Line*)

Category(ies): Windows

Description: Draw a line in a window using a WIN_RECT to define the end-points. The line is drawn in the current attribute (set with **WinAttr**). This routine is similar to **WinLine** except that it is more accurate in terms of clipping and the exact pixels drawn, at the expense of being much slower. Use it when precise pixel placement is needed especially when clipping is involved (see example).

Inputs:
w — WINDOW struct of a previously opened window.
Line — Endpoints of line to draw.

Outputs: None

Assumptions: The valid line attributes are: A_NORMAL (set), A_REVERSE (clear), and A_XOR.

Side Effects: Slower than **WinLine**, but more precise.

Availability: On AMS 2.00 and higher.

TI-89 / TI-92 Plus Differences: None

See Also: **WinAttr, WinLine, WinLineRel, WinLineTo**

(continued)

WinLineExt *(continued)*

Example:

```
/* This example draws to a window which is clipped. Because of the clipping, the
   first part of the first line drawn is clipped (not drawn). After the clipping
   region for the window is restored back to the full window; the second line drawn
   now draws the entire line. If WinLine was used in these cases the scan-line
   conversion used to draw the lines would cause the second line drawn to be slightly
   different than the first line and would leave some pixels still turned on. WinLine
   clips to the clipping region and then draws the line inside the clipping region.
   WinLineExt actually starts drawing at the line end-points and only draws those
   pixels inside the clipping region which is why it is slower.
*/
WINDOW w1;
WIN_RECT wr;
SCR_RECT sr, oldClip;

if (WinOpen( &w1, MakeWinRect(0,0,100,80), 0 )) {
   WinActivate( &w1 );
   WinClr( &w1 );
   oldClip = w1.Clip;
   SetWinClip( &w1, MakeScrRect(20,20, 80,60,&sr) );
   wr = *MakeWinRect(40,0,60,60);
   WinLineExt( &w1, &wr );   /* this will be clipped */
   GKeyIn( NULL, 0 );
   w1.Clip = oldClip;
   WinAttr( &w1, A_XOR );
   WinLineExt( &w1, &wr );   /* draw whole line */
   GKeyIn( NULL, 0 );
   WinClose( &w1 );
}
```

WinLineRel

Declaration: void **WinLineRel** (WINDOW * *w*, WIN_COORDS *x0*, WIN_COORDS *y0*)

Category(ies): Windows

Description: Draw a line from the current pen position using relative coordinates, then update the pen position to those coordinates. The current pen position can be initialized with **WinMoveTo**.

Inputs:
w — WINDOW struct of a previously opened window.
x0, y0 — End-point of line to draw.

Outputs: None

Assumptions: The valid line attributes are: A_NORMAL (set), A_REVERSE (clear), A_XOR and A_THICK1 (double thick line).

Side Effects: None

Availability: All versions of the TI‑89 / TI‑92 Plus.

TI‑89 / TI‑92 Plus Differences: None

See Also: **WinAttr, WinLine, WinLineTo, WinMoveTo**

Example:

```
/* Draw a rectangle (could use WinRect to do the same thing) */
WinMoveTo( &appW, 10, 10 );
WinLineRel( &appW, 90, 0 );
WinLineRel( &appW, 0, 50 );
WinLineRel( &appW, -90,0 );
WinLineRel( &appW, 0, -50 );
```

WinLineTo

Declaration: void **WinLineTo** (WINDOW * *w*, WIN_COORDS *x0*, WIN_COORDS *y0*)

Category(ies): Windows

Description: Draw a line from the current pen position using absolute window coordinates, then update the pen position to those coordinates. The current pen position can be initialized with **WinMoveTo**.

Inputs:
w — WINDOW struct of a previously opened window.
x0, y0 — End-point of line to draw.

Outputs: None

Assumptions: The valid line attributes are: A_NORMAL (set), A_REVERSE (clear), A_XOR and A_THICK1 (double thick line).

Side Effects: None

Availability: All versions of the TI‑89 / TI‑92 Plus.

TI‑89 / TI‑92 Plus Differences: None

See Also: **WinAttr, WinLine, WinLineRel, WinMoveTo**

Example:

```
/* Draw a rectangle (could use WinRect to do the same thing) */
WinMoveTo( &appW, 10, 10 );
WinLineTo( &appW, 100, 10 );
WinLineTo( &appW, 100, 60 );
WinLineTo( &appW, 10, 60 );
WinLineTo( &appW, 10, 10 );
```

WinMoveRel

Declaration: void **WinMoveRel** (WINDOW * *w*, WIN_COORDS *x0*, WIN_COORDS *y0*)

Category(ies): Windows

Description: Set the pen position relative to the current pen position. The pen position is set with **WinMoveTo**. It affects where **WinChar** and **WinStr** draw characters and strings as well as the line position for **WinLineRel** and **WinLineTo**.

Inputs:
w — WINDOW struct of a previously opened window.
x0, y0 — New, relative, pen position.

Outputs: None

Assumptions: None

Side Effects: Sets the current pen position for the window until changed.

Availability: All versions of the TI-89 / TI-92 Plus.

TI-89 / TI-92 Plus Differences: None

See Also: **WinMoveTo, WinChar, WinStr, WinLineRel, WinLineTo**

Example: See **WinChar** and **WinStr**.

WinMoveTo

Declaration: void **WinMoveTo** (WINDOW * *w*, WIN_COORDS *x0*, WIN_COORDS *y0*)

Category(ies): Windows

Description: Set the current pen position. The pen position affects where **WinChar** and **WinStr** draw characters and strings as well as the line position for **WinLineRel** and **WinLineTo**.

Inputs:
w — WINDOW struct of a previously opened window.
x0, y0 — New pen position.

Outputs: None

Assumptions: None

Side Effects: Sets the current pen position for the window until changed.

Availability: All versions of the TI‑89 / TI‑92 Plus.

TI‑89 / TI‑92 Plus Differences: None

See Also: **WinMoveRel, WinChar, WinStr, WinLineRel, WinLineTo**

Example: See **WinLineTo**.

WinOpen

Declaration: BOOL **WinOpen** (WINDOW * *w*, const WIN_RECT * *wRegion*, WORD *Flags*, . . .)

Category(ies): Windows

Description: Open a new window, initializing all fields of the WINDOW structure. Link this window into the current list of windows as the topmost window.

Inputs:

w — Address of a WINDOW struct.

wRegion — Screen coordinates of window's region.

Flags — Flags may be set as follow (note that WF_SAVE_SCR and WF_DUP_SCR are mutually exclusive).

WF_SAVE_SCR
: Save the screen region underneath the window (restore it when the window is closed). Return FALSE if not enough memory to allocate a save buffer.

WF_DUP_SCR
: Keep a duplicate copy of all data written to the window. Return FALSE if not enough memory to allocate a save buffer.

WF_TTY
: Write characters in TTY mode (translate '\n', '\r' to newline, '\f' to clear screen, and wrap at end of lines).

WF_TITLE
: Pointer to title follows *Flags* as optional argument.

WF_VIRTUAL
: Must also set WF_DUP_SCR, no writes to actual LCD are done only writes to DUP_SCR.

WF_NOBORDER
: Do not draw border for window.

WF_ROUNDEDBORDER
: Draw a rounded border (like dialog boxes).

Outputs: TRUE if window successfully opened, FALSE if not enough memory (WF_DUP_SCR or WF_SAVE_SCR) or *wRegion* contains invalid window coordinates.

(continued)

WinOpen *(continued)*

Assumptions: None

Side Effects: May cause heap compression.

Availability: All versions of the TI-89 / TI-92 Plus.

TI-89 / TI-92 Plus Differences: None

See Also: **WinClose, WinRemove**

Example: See **WinBitmapGet**, **WinChar**, and **WinDupStat**.

WinPixGet

Declaration: WORD **WinPixGet** (WINDOW * *w*, WIN_COORDS *x0*, WIN_COORDS *y0*)

Category(ies): Windows

Description: Return the status of a pixel in a window: 0 or 1.

Inputs: *w* — WINDOW struct of a previously opened window.

x0, y0 — WINDOW coordinates of pixel to test.

Outputs: 1 if the selected pixel is on or 0 if it is off (or outside the window).

Assumptions: Returns 0 if the given coordinates are outside the window.

Side Effects: None

Availability: All versions of the TI‑89 / TI‑92 Plus.

TI‑89 / TI‑92 Plus Differences: None

See Also: **WinAttr, WinPixSet**

Example:

```
/* The ptTest and pxlTest functions use WinPixGet to determine if a particular pixel
   is turned on in the graph screen as shown in the following piece of code. */
if (WinPixGet(gr_active->grwinp, ix, iy))
   Val = TRUE_TAG;
else
   Val = FALSE_TAG;
```

WinPixSet

Declaration: void **WinPixSet** (WINDOW * *w*, WIN_COORDS *x0*, WIN_COORDS *y0*)

Category(ies): Windows

Description: Set a pixel value depending on the current window attribute.

Inputs: *w* — WINDOW struct of a previously opened window.

x0, y0 — WINDOW coordinates of pixel to set.

Outputs: None

Assumptions: Current attribute (set by **WinAttr**) is A_NORMAL (set), A_REVERSE (clear), or A_XOR.

Side Effects: None

Availability: All versions of the TI‑89 / TI‑92 Plus.

TI‑89 / TI‑92 Plus Differences: None

See Also: **WinAttr, WinPixGet**

Example:

```
/* Given a pointer to a WIN_RECT (wRect) this piece of code will draw a rectangle
   with the corners turned off. */
WinRect( &appW, wRect, B_NORMAL | A_NORMAL );  /* draw rectangle */
WinAttr( &appW, A_REVERSE );   /* so pixels cleared */
WinPixSet( &appW, wRect->x0, wRect->y0 );    /* turn off corner pixels */
WinPixSet( &appW, wRect->x1, wRect->y0 );
WinPixSet( &appW, wRect->x1, wRect->y1 );
WinPixSet( &appW, wRect->x0, wRect->y1 );
```

WinRect

Declaration: void **WinRect** (WINDOW * *w*, const WIN_RECT * *wRect*, BYTE *BoxAttr*)

Category(ies): Windows

Description: Draw a rectangle in a window. The values for *BoxAttr* are the same values for drawing a line in a window (A_NORMAL, A_REVERSE, or A_XOR) OR'd with one of the following values.

B_NORMAL	—	Draw a normal rectangle.
B_DOUBLE	—	Draw a double thick rectangle.
B_ROUNDED	—	Draw a rectangle with rounded corners. (Dialog boxes are drawn this way.)
B_CUT	—	Draw a rectangle with the upper corners "cut". (Toolbars are drawn this way.)

Inputs:

w	—	WINDOW struct of a previously opened window.
wRect	—	WIN_RECT defining upper left (x0, y0) and lower right (x1, y1) corners of rectangle.
BoxAttr	—	A_NORMAL, A_REVERSE, A_XOR OR'd with B_NORMAL, B_DOUBLE, B_ROUNDED, or B_CUT.

Outputs: None

Assumptions: None

Side Effects: None

Availability: All versions of the TI‑89 / TI‑92 Plus.

TI‑89 / TI‑92 Plus Differences: None

See Also: None

Example:

```
/* Draw 4 rectangles each using one of the 4 different box attributes. */
WinRect( &appW, MakeWinRect(5,5,90,50), A_NORMAL | B_NORMAL );
WinRect( &appW, MakeWinRect(10,10,85,45), A_NORMAL | B_DOUBLE );
WinRect( &appW, MakeWinRect(15,15,80,40), A_NORMAL | B_ROUNDED );
WinRect( &appW, MakeWinRect(20,20,75,35), A_NORMAL | B_CUT );
```

WinRemove

Declaration: void **WinRemove** (WINDOW * *w*, BOOL *UpdateScreen*)

Category(ies): Windows

Description: Close a window, releasing any memory assigned to it.

Inputs:

w — WINDOW struct of a previously opened window.

UpdateScreen — If TRUE then the next window in the linked list of windows is activated and the screen is updated. If FALSE then no other window is activated and the screen is not updated.

Outputs: None

Assumptions: All windows opened with **WinOpen** must be closed with either **WinClose** or **WinRemove**.

Side Effects: None if *UpdateScreen* is FALSE, otherwise the same as **WinClose**.

Availability: On AMS 2.00 and higher.

TI-89 / TI-92 Plus Differences: None

See Also: **WinOpen, WinClose**

Example:

```
/* virtual windows are normally closed with WinRemove as shown in this example */
if (WinOpen(&VirtW, MakeWinRect(0,0,COL_MAX,ROW_MAX),WF_VIRTUAL | WF_NOBORDER)) {
   /* . . . use VirtW . . . */
   WinRemove( &VirtW, FALSE );   /* no screen update */
}
```

WinReOpen

Declaration: BOOL **WinReOpen** (WINDOW * *w*, const WIN_RECT * *wRegion*, WORD *Flags*, . . .)

Category(ies): Windows

Description: Reopen an existing window to a new size. Not valid for WF_SAVE_SCR (unless using just to call **WinOpen**). If the window is not in the "list of windows" then just calls **WinOpen**. Otherwise, it updates the Client, Window, Clip, and Port regions. If the new window is of the same size as the old one, then the Port region (DUP_SCR) is not cleared.

Inputs: *w* — WINDOW struct of a previously opened window.

Outputs: Returns TRUE if window reopened OK, FALSE if not (bad window or not enough memory to enlarge DUP_SCR — **WinRemove** called then).

Assumptions: None

Side Effects: May cause heap compression.

Availability: All versions of the TI-89 / TI-92 Plus.

TI-89 / TI-92 Plus Differences: None

See Also: **WinOpen**

Example: This example opens a window with WF_DUP_SCR on one side of the screen. It then reopens the same window next to that window and copies the backup image to this new window position.

```
WINDOW win1;
if (WinOpen(&win1,MakeWinRect(0,20,50,70),WF_DUP_SCR|WF_TTY )) {
   WinActivate(&win1);
   WinClr(&win1);
   WinStr(&win1,"ABCDEFGHIJKLMNOP");
   /* Since the size of win1 did not change, this WinReOpen will succeed.
      The backup image will not be cleared since the window size did not change. */
   WinDeactivate(&win1);
   WinReOpen(&win1,MakeWinRect(50,20,100,70),WF_DUP_SCR|WF_TTY);
   WinActivate(&win1);
   WinBackupToScr(&win1);
   GKeyIn(NULL,0);
   WinClose(&win1);
}
```

WinScrollH

Declaration: void **WinScrollH** (WINDOW * *w*, const WIN_RECT * *wRegion*, SWORD *numCols*)

Category(ies): Windows

Description: Scroll a region of a window horizontally. Blank areas are filled with the current background for the window. If *numCols* < 0 then scroll right otherwise scroll left. Note that if the region to be scrolled starts on a byte boundary (left-most pixel), then the region will scroll much faster.

Inputs:

w — WINDOW struct of a previously opened window.

wRegion — WINDOW region to scroll.

numCols — Number of columns to scroll (negative scrolls right, positive left).

Outputs: None

Assumptions: Background set with **WinBackground**.

Side Effects: None

Availability: All versions of the TI-89 / TI-92 Plus.

TI-89 / TI-92 Plus Differences: None

See Also: **WinBackground, WinScrollV**

(continued)

WinScrollH *(continued)*

Example:

```
#define V_SCROLL 8
#define H_SCROLL 8
/* Display the 2D output from Parse2DExpr to a window (with defined offsets)
   scrolling the output as the arrow keys are pressed. */
void View2DExpr( EStackIndex i, WINDOW *Win, SWORD x, SWORD y) {
WORD Width, Depth, Height, MaxX, MaxY, Key;
SWORD z;
WIN_RECT wr;

Parms2D( i, &Width, &Depth, &Height );
MaxX = WinWidth( Win ) - 1;
MaxY = WinHeight( Win ) - 1;
z = Width - MaxX;
wr.x0 = 0; wr.x1 = MaxX; wr.y0 = 0; wr.y1 = MaxY;
WinClr( Win );
do {
   Print2DExpr( i, Win, x, y );
   Key = GKeyIn( 0l, GKF_NORMAL );
   switch( Key ) {
      case KB_UP:
         if (y < Height) {
            y += V_SCROLL;
            WinScrollV( Win, &wr, -V_SCROLL );
         }
         break;
      case KB_DOWN:
         if (y + Depth > MaxY) {
            y -= V_SCROLL;
            WinScrollV( Win, &wr, V_SCROLL );
         }
         break;
      case KB_LEFT:
         if (x < 0) {
            x += H_SCROLL;
            WinScrollH( Win, &wr, -H_SCROLL );
         }
         break;
      case KB_RIGHT:
         if (z > 0)
            if (abs(x) < z) {
               x -= H_SCROLL;
               WinScrollH( Win, &wr, H_SCROLL );
         }
      break;
   case KB_ESC:
   case KB_ENTER: return;
   }
} while TRUE;
}
```

WinScrollV

Declaration: void **WinScrollV** (WINDOW * *w*, const WIN_RECT * *wRegion*, SWORD *numRows*)

Category(ies): Windows

Description: Scroll a region of a window vertically (blank areas are filled with current background for the window. If *NumRows* < 0 then scroll down otherwise scroll up.

Inputs:

w — WINDOW struct of a previously opened window.

wRegion — WINDOW region to scroll.

numRows — Number of rows to scroll (negative scrolls down, positive up).

Outputs: None

Assumptions: Background set with **WinBackground**.

Side Effects: None

Availability: All versions of the TI-89 / TI-92 Plus.

TI-89 / TI-92 Plus Differences: None

See Also: **WinBackground, WinScrollH**

Example: See **WinScrollH**.

WinStr

Declaration: void **WinStr** (WINDOW * *w*, char * *Str*)

Category(ies): Windows

Description: Draw a string to a window at the current pen location. The current pen location is updated to point to the end of where the string was written.

Inputs: *w* — WINDOW struct of a previously opened window.

Str — Pointer to string to write.

Outputs: None

Assumptions: The default attribute (A_NORMAL) can be changed with **WinAttr**; the default font (F_6x8) can be changed with **WinFont**; the current window position is set with **WinMoveTo** or **WinMoveRel**.

The supported values for character attributes are: A_NORMAL, A_REVERSE, A_XOR, A_SHADED, A_REPLACE. See **WinAttr** for a detailed description of the character attributes.

Side Effects: None

Availability: All versions of the TI‑89 / TI‑92 Plus.

TI‑89 / TI‑92 Plus Differences: None

See Also: **WinAttr, WinChar[XY], WinFont, WinMoveTo, WinMoveRel, WinStrXY**

(continued)

WinStr *(continued)*

Example:

The above image was created from the example below.

```
short i;
WinClr(&appW);
WinFont( &appW, F_8x10 );
for (i = 5; i <= 45; i += 10)
   WinLine( &appW, MakeWinRect( 0, i, 90, i ) );
WinAttr( &appW, A_NORMAL );  WinMoveTo( &appW, 16, 0 ); WinStr( &appW, "NORMAL" );
WinAttr( &appW, A_REPLACE ); WinMoveRel( &appW, -48, 10 ); WinStr( &appW, "REPLACE" );
/* x parm to WinMoveRel is negative since current pen position is at end of previous
   string */
WinAttr( &appW, A_REVERSE ); WinStrXY( &appW, 16, 20, "REVERSE" );
WinAttr( &appW, A_SHADED  ); WinStrXY( &appW, 16, 30, "SHADED" );
WinAttr( &appW, A_XOR     ); WinStrXY( &appW, 16, 40, "ATTR XOR" );
```

WinStrXY

Declaration: void **WinStrXY** (WINDOW * *w*, WIN_COORDS *x*, WIN_COORDS *y*, char * *Str*)

Category(ies): Windows

Description: Draw a string to a window at position *x, y* (pixel based). The current pen location is updated to point to the end of where the string was written.

Inputs:

w — WINDOW struct of a previously opened window.

x, y — Window position to write to (set with **WinMoveTo**).

Str — Pointer to string to write.

Outputs: None

Assumptions: The default attribute (A_NORMAL) can be changed with **WinAttr**; the default font (F_6x8) can be changed with **WinFont**.

The supported values for character attributes are: A_NORMAL, A_REVERSE, A_XOR, A_SHADED, and A_REPLACE. See **WinAttr** for a detailed description of the character attributes.

Side Effects: None

Availability: All versions of the TI‑89 / TI‑92 Plus.

TI‑89 / TI‑92 Plus Differences: None

See Also: **WinAttr, WinChar[XY], WinFont, WinStr**

Example: See **WinStr**.

WinStrXYWrap

Declaration: short **WinStrXYWrap**(WINDOW * *w*, WIN_COORDS *x*, WIN_COORDS *y*, char * *Str*, WORD *wwFlags*)

Category(ies): Windows

Description: Draw a word-wrapped string to a window at position *x, y* (pixel based). The current pen location is updated to point to the end of where the string was written. Words are wrapped on spaces and newlines. The height in pixels of the text drawn is returned. The text is drawn to fit from the *x* coordinate passed to the right edge of the window.

> **NOTE:** If the WWF_DRAW bit in *wwFlags* is not set then no drawing is done.

Inputs:

w — WINDOW struct of a previously opened window.

x, y — Window position to write to, the *x* coordinate also specifies the left margin to wrap on.

Str — Pointer to string to write.

wwFlags — WWF_DRAW
Do actual draw (if not set then just the height of the text drawn is returned).

WWF_WRAP_ON_COMMAS
Also wrap on commas.

Outputs: Returns the height in pixels of the text drawn.

Assumptions: The default attribute (A_NORMAL) can be changed with **WinAttr**.
The default font (F_6x8) can be changed with **WinFont**.

The supported values for character attributes are: A_NORMAL, A_REVERSE, A_XOR, A_SHADED, and A_REPLACE. See **WinAttr** for a detailed description of the character attributes.

Side Effects: None

Availability: On AMS 2.04 and higher.

TI-89 / TI-92 Plus Differences: None

See Also: **WinAttr**, **WinChar, WinCharXY**, **WinFont**, **WinStr**

Example: See **WinCharXY** for the source to this routine since it is not available before AMS 2.04.

WinWidth

Declaration: WIN_COORDS **WinWidth** (WINDOW * *w*)

Category(ies): Windows

Description: Returns the width of the client (drawable) area of a window.

Inputs: *w* — WINDOW struct of a previously opened window.

Outputs: Client width of window.

Assumptions: The window region is the region that was defined when the window was created with **WinOpen**. If the window is full screen (not counting the status bar which may not be overlapped), then the client region is equal to the window region. The client region is reduced by adding borders or a title to a window.

Side Effects: None

Availability: All versions of the TI-89 / TI-92 Plus.

TI-89 / TI-92 Plus Differences: None

See Also: **WinOpen, WinHeight**

Example: See **WinHeight**.

Appendix B: Global Variables

ARb_int_count

Declaration: Quantum **ARb_int_count**

Category(ies): Algebra Utilities

Description: This variable is set to 0 every time the calculator is reset or NewProb is selected from the Clean Up Home screen menu. **ARb_int_count** is incremented by 1 then used as the suffix in successive arbitrary-integer variables of the form @n . . . generated by function calls such as solve(sin(x)=0, x). **ARb_int_count** wraps back to 0 after 255.

Inputs: Not applicable.

Outputs: Not applicable.

Assumptions: Access_AMS_Global_Variables must be defined.

Side Effects: Not applicable.

Availability: All versions of the TI‑89 / TI‑92 Plus.

TI‑89 / TI‑92 Plus Differences: None

See Also: **ARb_real_count**

Example:

```
ARb_int_count = 0;  /* Reset counter to 0 */
```

ARb_real_count

Declaration: Quantum **ARb_real_count**

Category(ies): Algebra Utilities

Description: This variable is set to 0 every time the calculator is reset or NewProb is selected from the Clean Up Home screen menu. **ARb_real_count** is incremented by 1 then used as the suffix in successive arbitrary-integer variables of the form @n . . . generated by function calls such as zeros(0, x). **ARb_real_count** wraps back to 0 after 255.

Inputs: Not applicable.

Outputs: Not applicable.

Assumptions: Access_AMS_Global_Variables must be defined.

Side Effects: Not applicable.

Availability: All versions of the TI‑89 / TI‑92 Plus.

TI‑89 / TI‑92 Plus Differences: None

See Also: **ARb_int_count**

Example:

```
ARb_real_count = 0;  /* Reset counter to 0 */
```

NG_control

Declaration: CONTROL_BITS **NG_control**

Category(ies): Algebra Utilities

Description: The bits in this variable control various computational and formatting modes. These bits are restored at the beginning of each interaction cycle. They should be queried only via the various IS_ . . . macros and altered only via the various SET_ . . . macros described below.

In most cases it is courteous to restore existing modes as soon as you no longer need to force them. The best way to do this is to capture the entire previous settings by an assignment such as `old_NG_control = NG_control`. Then execute an assignment `NG_control = old_NG_control` as soon as you no longer need to force any modes. Encapsulate this block of code using TRY, ONERR and ENDTRY if there is a chance of an exception bypassing the restoration assignment.

SET_DOMAIN_REAL
SET_DOMAIN_COMPLEX
IS_DOMAIN_REAL
IS_DOMAIN_COMPLEX
SET_COMPLEX_RECTANGULAR_FORM
SET_COMPLEX_EXPONENTIAL_FORM
SET_COMPLEX_FORMAT_AUTO — Use whichever form is most compact for each nonreal subexpression.
IS_COMPLEX_RECTANGULAR_FORM
IS_COMPLEX_EXPONENTIAL_FORM
IS_COMPLEX_FORMAT_AUTO
SET_RADIANS
SET_DEGREES
IS_RADIANS
IS_DEGREES
SET_ARITH_EXACT
SET_ARITH_APPROX
SET_ARITH_AUTO
IS_ARITH_EXACT
IS_ARITH_APPROX
IS_ARITH_AUTO
SET_EXPAND_KERNELS — Expand ln(), e^(), abs() etc. in results.

(continued)

NG_control *(continued)*

Description: ***(continued)***

SET_COLLECT_KERNELS	—	Collect ln, e^, abs, etc. in results. (This is the default except within the expand function.)
IS_EXPAND_KERNELS		
IS_COLLECT_KERNELS		
SET_EXPAND_TRIG	—	Expand angle sums and multiple angles.
SET_COLLECT_TRIG	—	Replace powers and products of sinusoids with angle sums and multiple angles.
SET_TO_SIN	—	Replace appropriate powers of cosines with sines.
SET_TO_COS	—	Replace appropriate powers of sines with cosines.
SET_AUTO_TRIG	—	Default: Do some expansion and to_sin or to_cos where it is guaranteed to make a more compact result.
SET_NO_TRIG	—	Prevent all of the above trigonometric transformations.
IS_EXPAND_TRIG		
IS_COLLECT_TRIG		
IS_TO_SIN		
IS_TO_COS		
IS_AUTO_TRIG		
IS_NO_TRIG		

Inputs: None

Outputs: None

Assumptions: Access_AMS_Global_Variables must be defined.

Side Effects: None

Availability: All versions of the TI‑89 / TI‑92 Plus.

TI‑89 / TI‑92 Plus Differences: None

See Also: None

(continued)

NG_control *(continued)*

Example:

```
old_NG_control = NG_control;  /* Save old setting. */
SET_RADIANS;
.
.                             /* Do some work in radian mode. */
.
if (IS_DOMAIN_REAL)
   .
   .                          /* Do appropriate thing for real mode. */
   .
else
   .
   .                          /* Do appropriate thing for complex mode. */
   .
.
.
.
NG_control = old_NG_control;  /* Restore old settings. */
```

NG_such_that_index

Declaration: EStackIndex **NG_such_that_index**

Category(ies): Algebra Utilities

Description: This global variable is the EStackIndex of the currently active “such that” expression. When a user enters an expression using the “with” (|) operator, the expression to the right of the “with” operator is indexed by **NG_such_that_index**. In addition, various internal system processes may supplement or override this expression. The entry point **push_substitute_using_such_that** provides a direct way to supplement or override this expression.

Inputs: Not applicable.

Outputs: Not applicable.

Assumptions: Access_AMS_Global_Variables must be defined.

Side Effects: Not applicable.

Availability: On AMS 2.04 and higher.

TI-89 / TI-92 Plus Differences: None

See Also: **push_substitute_using_such_that**

Example:

The expression x + 1 | x = 3 produces the following external tokenized form
3 1 NONNEGATIVE_INTEGER_TAG X_VAR_TAG EQUATION_TAG X_VAR_TAG 1 1 NONNEGATIVE_INTEGER_TAG ADD_TAG SUCH_THAT_TAG

Then, during the evaluation of the expression x + 1, the expression x = 3 is indexed by **NG_such_that_index** at the bolded tag as follows.
3 1 NONNEGATIVE_INTEGER_TAG X_VAR_TAG **EQUATION_TAG**

The result is that 3 is substituted for x in the expression x + 1 yielding 4 as the result.

RAtionalize_tol

Declaration: Float **RAtionalize_tol**

Category(ies): Algebra Utilities

Description: This variable is the tolerance used to convert floating-point numbers to rational numbers or to round floating-point numbers by converting them to rational numbers then back to floating point. Appropriate values are either 0.0 or 6e-14 through 0.1 inclusive. The value is normally set to 6e-14, but it might be temporarily modified by exact (. . . , tol) or for various internal purposes.

Inputs: Not applicable.

Outputs: Not applicable.

Assumptions: Access_AMS_Global_Variables must be defined.

Side Effects: Not applicable.

Availability: All versions of the TI‑89 / TI‑92 Plus.

TI‑89 / TI‑92 Plus Differences: None

See Also: None

Example:

```
old_RAtionalize_tol = RAtionalize_tol;  /* Save old value. */
RAtionalize_tol = 2e-7;
   .
   .                                    /* Do some calculations
   .
RAtionalize_tol = old_RAtionalize_tol;  /* Restore old value. */
```

EV_appA

Declaration: AppID **EV_appA**

Category(ies): Apps, Operating System

Description: If the calculator is in full-screen mode, this variable contains the ID of the current app. If the calculator is in split-screen mode, it holds the ID of the application currently occupying the top or left window of the screen.

> **Note:** The OS maintains this variable — do not modify it in your application.

Inputs: Not applicable.

Outputs: Not applicable.

Assumptions: Access_AMS_Global_Variables must be defined.

Side Effects: Not applicable.

Availability: On AMS 2.00 and higher.

TI-89 / TI-92 Plus Differences: None

See Also: **EV_appB, EV_appSide, EV_currentApp, EV_runningApp**

Example:

```
Access_AMS_Global_Variables;
AppID appa = EV_appA;   /* Get ID of app on side A */
```

EV_appB

Declaration: AppID **EV_appB**

Category(ies): Apps, Operating System

Description: If the calculator is in full-screen mode, this variable contains AP_NONE. If the calculator is in split-screen mode, it holds the ID of the application currently occupying the bottom or right window of the screen.

> **Note:** The OS maintains this variable — do not modify it in your application.

Inputs: Not applicable.

Outputs: Not applicable.

Assumptions: Access_AMS_Global_Variables must be defined.

Side Effects: Not applicable.

Availability: On AMS 2.00 and higher.

TI-89 / TI-92 Plus Differences: None

See Also: **EV_appA, EV_appSide, EV_currentApp, EV_runningApp**

Example:

```
Access_AMS_Global_Variables;
AppID appb = EV_appB;   /* Get ID of app on side B */
```

EV_appSide

Declaration: UINT **EV_appSide**

Category(ies): Apps, Operating System

Description: If the calculator is in full-screen mode, this variable contains AP_SIDE_A. If the calculator is in split-screen mode, it holds a value indicating which side of the screen holds the focus, either AP_SIDE_A or AP_SIDE_B.

> **Note:** The OS maintains this variable — do not modify it in your application.

Inputs: Not applicable.

Outputs: Not applicable.

Assumptions: Access_AMS_Global_Variables must be defined.

Side Effects: Not applicable.

Availability: On AMS 2.00 and higher.

TI-89 / TI-92 Plus Differences: None

See Also: **EV_appA, EV_appB, EV_currentApp, EV_runningApp**

Example:

```
Access_AMS_Global_Variables;
if (EV_appSide == AP_SIDE_A)
{
  /* Side A is active */
}
else
{
  /* Side B is active */
}
```

EV_currentApp

Declaration: AppID **EV_currentApp**

Category(ies): Apps, Operating System

Description: ID of the task currently holding the focus. Most events are directed to the current application.

Inputs: Not applicable.

Outputs: Not applicable.

Assumptions: Access_AMS_Global_Variables must be defined.

Side Effects: Not applicable.

Availability: On AMS 2.00 and higher.

TI-89 / TI-92 Plus Differences: None

See Also: **EV_runningApp**

Example:

```
UCHAR *name = GetAppName(EV_currentApp); /* get name of current application */
```

EV_runningApp

Declaration: AppID **EV_runningApp**

Category(ies): Apps, Operating System

Description: ID of the task currently processing an event.

Sometimes the running app is not the current app. This can happen when an app sends a message to another app or when the OS sends a CM_WPAINT message to an app to repaint its window when it is not the current app.

Inputs: Not applicable.

Outputs: Not applicable.

Assumptions: Access_AMS_Global_Variables must be defined.

Side Effects: Not applicable.

Availability: On AMS 2.00 and higher.

TI-89 / TI-92 Plus Differences: None

See Also: **EV_currentApp**

Example:

```
UCHAR *name = GetAppName(EV_runningApp); /* get name of running app */
```

OO_firstACB

Declaration: AppID **OO_firstACB**

Category(ies): Apps

Description: ID of the first app in the application control block list.

Inputs: Not applicable.

Outputs: Not applicable.

Assumptions: Access_AMS_Global_Variables must be defined.

Side Effects: Not applicable.

Availability: On AMS 2.00 and higher.

TI-89 / TI-92 Plus Differences: None

See Also: **OO_NextACB, OO_PrevACB**

Example:

```
AppID appid;
for (appid = OO_firstACB; appid != H_NULL; appid = OO_NextACB(appid))
{
   /* process each ACB */
   .
   .
   .
}
```

OO_SuperFrame

Declaration: pFrame **OO_SuperFrame**

Category(ies): Apps

Description: Immediately after a call to **OO_GetAttr** or **OO_GetAppAttr** to get an object frame attribute, **OO_SuperFrame** contains a pointer to the parent frame of the frame from which the attribute was retrieved. This is used to call inherited methods in an object's parent hierarchy.

Inputs: Not applicable.

Outputs: Not applicable.

Assumptions: Access_AMS_Global_Variables must be defined.

Side Effects: Not applicable.

Availability: On AMS 2.00 and higher.

TI-89 / TI-92 Plus Differences: None

See Also: Not applicable

Example: This example uses a hierarchy of three object frames to illustrate how inherited methods are called. Frame C inherits from B which in turn inherits from A. Frame A is the base frame.

```
#include "tiams.h"

/* FDL source:
   func 0x10001 foo(pFrame): void;

   The FDL compiler generates the following macros given the above source.
*/
#define OO_FOO (65537)
#define foo(obj) \
   ((void (* const)(pFrame))OO_GetAttr(obj,65537))(obj)

void A_foo(pFrame);
void B_foo(pFrame);

FRAME(A, 0, 0, OO_FOO, 1)
   ATTR(OO_FOO, &A_foo)
ENDFRAME

FRAME(B, &A, 0, OO_FOO, 1)
   ATTR(OO_FOO, &B_foo)
ENDFRAME
```

(continued)

OO_SuperFrame *(continued)*

```
/* The C compiler complains about 0 length array for empty FRAMEs
   FRAME(C, &B, 0, 0, 0)
   ENDFRAME
*/
const OO_Hdr C = /* hand-coded empty FRAME C */
{
   (pFrame)&B, 0, OO_RO | OO_SEQ, 0, 0
};

void A_foo(pFrame self)
{
   /* implementation of A::foo */
}

void B_foo(pFrame self)
{
   /* implementation of B::foo */

   Access_AMS_Global_Variables;
   pFrame super = OO_SuperFrame;

   /* Call inherited foo method */
   foo(super);
}

void main(void)
{
   foo((pFrame)&C);
}
```

Main calls method foo of frame C. Frame C has no implementation of foo, so inherited method foo from frame B (routine B_foo) actually gets called.

B_foo wants to call the inherited foo method of frame self. If it calls the parent foo method of frame self, it will, in fact, be calling itself recursively — remember self points to frame C, the parent of which is frame B.

OO_SuperFrame conveniently contains a pointer to the parent frame of the frame where the latest method address was retrieved. Method foo was retrieved from frame B, so **OO_SuperFrame** contains a pointer to frame A. Now a call to foo(super) in B_foo calls the foo method of frame A (routine A_foo).

Note:	**OO_SuperFrame** is a system-wide global variable. If you intend to use it in one of your method implementations, make an immediate copy into a local variable because subsequent method calls or attribute accesses will change its value.

FLOATTAB

Declaration: BCD16 **FLOATTAB** [ ]

Category(ies): Direct Floating Point Operations

Description: This is an array of commonly used floating-point numbers that an app can access.

Inputs: FPI_TWOPI, FPI_ONEPI, FPI_PIDIV2, FPI_PIDIV4, FPI_360, FPI_180, FPI_90, FPI_45, FPI_180DIVPI, FPI_PIDIV180, FPI_0, FPI_PT001, FPI_PT1, FPI_PIDIV24, FPI_PT5, FPI_PT9, FPI_1, FPI_NEG1, FPI_SQRRT2, FPI_2, FPI_3, FPI_2PI12, FPI_10, FPI_NEG10, FPI_12, FPI_14, FPI_20, FPI_70, FPI_BIGGEST, FPI_NEGBIGGEST, FPI_16000, FPI_NEG16000, FPI_32767, FPI_NEG32768, FPI_65535, FPI_1E14, FPI_INVALID, FPI_LOGE, FPI_POS0, FPI_NEG0, FPI_POSINF, FPI_NEGINF, FPI_UNSINF

Outputs: 2*π, π, π/2, π/4, 360.0, 180.0, 90.0, 45.0, 180/π, π/180, 0.0, 0.001, 0.1, π/24, 0.5, 0.9, 1.0, -1.0, sqrt(2), 2.0, 3.0, 2*π (12 digits), 10.0, -10.0, 12.0, 14.0, 20.0, 70.0, 10^1000, -(10^1000), 16000.0, -16000.0, 32767.0, -32768.0, 65535.0 (max f.p. integer), 1.0e14, “Invalid Float”, log(e), Positive Zero, Negative Zero, +infinity, -infinity, infinity

Assumptions: Access_AMS_Global_Variables must be defined.

Side Effects: None

Availability: On AMS 2.00 and above.

TI-89 / TI-92 Plus Differences: None

See Also: None

Example: **FLOATTAB** values can be used just like normal floating-point values as shown in the following code fragment. They have the advantage of being able to represent some values the compiler does not know about such as the Invalid Float and the infinities.

```
Access_AMS_Global_Variables;
BCD16 pVal, tVal;

.
.
.
if ( pval == 0.0 )
   tVal = FLOATTAB[FPI_POSINF];
else if ( pval == 1.0 )
   tval = FLOATTAB[FPI_NEGINF];
.
.
.
```

IM_re_tol

Declaration: float **IM_re_tol**

Category(ies): Direct Floating Point Operations

Description: At the beginning of each interaction cycle, this variable is set to FLT_EPSILON, which is defined to be 5.0e-14.

Inputs: Not applicable.

Outputs: Not applicable.

Assumptions: Access_AMS_Global_Variables must be defined.

Side Effects: Not applicable.

Availability: On AMS 2.04 and higher.

TI-89 / TI-92 Plus Differences: None

See Also: **replace_top2_with_imre**

Example:

```
Float old_IM_re_tol = IM_re_tol;  /* Save current setting. */
IM_re_tol = 0.0
   .   /* Do some complex arithmetic with no artificial underflow of */
   .   /* relatively small magnitude components. */
   .
IM_re_tol = old_IM_re_tol;        /* Restore previous setting. */
```

CU_cursorState

Declaration: CU_STATE **CU_cursorState**

Category(ies): Display, Interrupts

Description: Holds the current state of the cursor, either CU_CURSOR_ON or CU_CURSOR_OFF.

Do not fetch or modify the value of **CU_cursorState** directly. Use **CU_state** and **CU_restore** to get and set the current cursor state, or **CU_start** or **CU_stop** to start or stop cursor flash.

Inputs: Not applicable.

Outputs: Not applicable.

Assumptions: Access_AMS_Global_Variables must be defined.

Side Effects: Not applicable.

Availability: On AMS 2.00 and higher.

TI-89 / TI-92 Plus Differences: None

See Also: **CU_state, CU_restore, CU_start, CU_stop**

Example:

```
CU_STATE cs = CU_state();                    /* save cursor state */
CU_stop()                                    /* turn off cursor */
   .
   . /* do something with cursor off */
   .
CU_restore(cs);                              /* restore cursor state */
```

ScrRect

Declaration: SCR_RECT **ScrRect**

Category(ies): Display

Description: Global SCR_RECT that defines the entire drawable screen area (0, 0, MAX_X, W_MAX_Y).

Inputs: None

Outputs: None

Assumptions: Access_AMS_Global_Variables must be defined.

Side Effects: None

Availability: All versions of the TI‑89 / TI‑92 Plus.

TI‑89 / TI‑92 Plus Differences: MAX_X, W_MAX_Y differ between the TI‑89 and the TI‑92 Plus.

See Also: None

Example: This code fragment redraws its window border using the entire drawable screen area as a clipping region (even though it could use &winPtr->Window).

```
Access_AMS_Global_Variables; /* needed to access ScrRect */
WINDOW *winPtr = &winMain;
.
.
.
DrawWinBorder( winPtr, &ScrRect );
```

errno

Declaration: int **errno**

Category(ies): Error Handling, Utilities

Description: Global error set by some system utility routines: **strtod**, **strtol**.

Inputs: Not applicable.

Outputs: Not applicable.

Assumptions: Access_AMS_Global_Variables must be defined.

Side Effects: Not applicable.

Availability: On AMS 2.04 and higher.

TI-89 / TI-92 Plus Differences: None

See Also: **strtod, strtol**

Example:

```
Access_AMS_Global_Variables;
long l1, l2;
short e1, e2;

l1 = strtol( "2147483647", NULL, 10 );
e1 = errno;  // 0 (result OK)
l2 = strtol( "2147483648", NULL, 10 );
e2 = errno;  // ERANGE (result is LONG_MAX)
```

EV_errorCode

Declaration: SINT **EV_errorCode**

Category(ies): Error Handling

Description: Setting this variable to an error number causes the OS to display the corresponding error message dialog box when your app returns from handling an event. Error numbers have names beginning with “ER_” in tiams.h.

ER_throwVar is the preferred mechanism for signaling errors, but there are some events during which your app must not throw an error. These events are CM_START, CM_ACTIVATE, CM_FOCUS, CM_UNFOCUS, CM_DEACTIVATE, CM_QUIT, CM_WPAINT, CM_INSTALL, CM_UNINSTALL, CM_PACK, and CM_UNPACK. If your app gets into an error state while processing one of these events, do whatever clean up you can, store an error number in **EV_errorCode**, then return to the OS. See section **10.2 Delayed Error Messages** for additional details.

If an application stores error numbers in **EV_errorCode** several times before returning to the OS, only the last stored value will be displayed as an error message.

Inputs: Not applicable.

Outputs: Not applicable.

Assumptions: Access_AMS_Global_Variables must be defined.

Side Effects: Not applicable.

Availability: On AMS 2.00 and higher.

TI-89 / TI-92 Plus Differences: None

See Also: **ER_throwVar**

Example:

```
EV_errorCode = ER_MEMORY;
```

bottom_estack

Declaration: EStackIndex **bottom_estack**

Category(ies): EStack Utilities

Description: EStackIndex of the bottom (lowest address) of the expression stack. This is a read-only variable. The estack resides in a fixed location, and **bottom_estack** must not be changed. The value indexed by **bottom_estack** is always END_OF_SEGMENT_TAG and must not be changed.

Inputs: Not applicable.

Outputs: Not applicable.

Assumptions: Access_AMS_Global_Variables must be defined.

Side Effects: Not applicable.

Availability: On AMS 2.00 and higher.

TI-89 / TI-92 Plus Differences: None

See Also: **top_estack, estack_max_index, reset_estack_size**

Example:

```
EStackDisplacement used = top_estack - bottom_estack;
```

Computes the amount of estack in use and assigns it to the variable used.

```
EStackDisplacement esize = estack_max_index – bottom_estack;
```

Computes the amount of usable estack space and assigns it to the variable esize.

estack_max_index

Declaration: EStackIndex **estack_max_index**

Category(ies): EStack Utilities

Description: EStackIndex of the highest usable address of the expression stack.

Inputs: Not applicable.

Outputs: Not applicable.

Assumptions: Access_AMS_Global_Variables must be defined.

Side Effects: Not applicable.

Availability: On AMS 2.04 and higher.

TI-89 / TI-92 Plus Differences: None

See Also: **bottom_estack, top_estack, reset_estack_size**

Example:

```
EStackDisplacement esize = estack_max_index – bottom_estack;
```

Computes the amount of usable estack space and assigns it to the variable esize.

```
EStackDisplacement avail = estack_max_index – top_estack;
```

Computes the amount of available estack space and assigns it to the variable avail.

top_estack

Declaration: EStackIndex **top_estack**

Category(ies): EStack Utilities

Description: EStackIndex of the top (highest address) of the used portion of the expression stack.

Inputs: Not applicable.

Outputs: Not applicable.

Assumptions: Access_AMS_Global_Variables must be defined.

Side Effects: Not applicable.

Availability: All versions of the TI‑89 / TI‑92 Plus.

TI-89 / TI-92 Plus Differences: None

See Also: **bottom_estack, estack_max_index, reset_estack_size**

Example:

```
EStackDisplacement used = top_estack - bottom_estack;
```

Computes the amount of estack in use and assigns it to the variable used.

```
EStackDisplacement avail = estack_max_index – top_estack;
```

Computes the amount of available estack space and assigns it to the variable avail.

FlashMemoryEnd

Declaration: BYTE * **FlashMemoryEnd**

Category(ies): Flash Memory

Description: Address of the first byte after the end of Flash memory.

Inputs: Not applicable.

Outputs: Not applicable.

Assumptions: Access_AMS_Global_Variables must be defined.

Side Effects: Not applicable.

Availability: On AMS 2.00 and higher.

TI-89 / TI-92 Plus Differences: None

See Also: Not applicable

Example:

```
BYTE *pfm;
Access_AMS_Global_Variables;
   .
   .
   .
while (pfm < FlashMemoryEnd)
{
   /* Do something with next byte of Flash memory */
   .
   .
   .
   pfm += 1;
}
```

gr_active, gr_other

Declaration: GR_WIN_VARS * **gr_active**

GR_WIN_VARS * **gr_other**

Category: Graphing

Description: Pointers to GR_WIN_VARS structs that contain most of the data used by the Graph application and other graph related apps. **gr_active** points to the GR_WIN_VARS struct containing all the information for the active graph. **gr_other** points to the information for the second graph in two graph mode. As the calculator user switches between the two windows in two graph mode, the pointers in **gr_active** and **gr_other** are swapped so that **gr_active** is always referring to the active graph. The members of a GR_WIN_VARS struct are shown below along with an explanation of the contents of each. None of the data should be changed directly by an app or ASM but can be accessed for use. System routines may be called to change many items (for example, **VarStore** may be used to change the graph system variables), but some data is for internal use only and should only be changed by the appropriate system app.

Struct Members and their Contents:

BCD16 flt_xcursor	—	Graph system variable xc.
BCD16 flt_ycursor	—	Graph system variable yc.
BCD16 flt_zcursor	—	Graph system variable zc.
BCD16 flt_tcursor	—	Graph system variable tc.
BCD16 flt_rcursor	—	Graph system variable rc.
BCD16 flt_thetacursor	—	Graph system variable θc.
BCD16 flt_ncursor	—	Graph system variable nc.
BCD16 recip_delx	—	1/Δx rounded to 6 significant digits.
BCD16 recip_dely	—	1/Δy rounded to 6 significant digits.
BCD16 orgxmin	—	Original xmin, before any panning has occurred.
BCD16 orgxmax	—	Original xmax, before any panning has occurred.
BCD16 panshift	—	Number of columns panned from orgxmin.

(continued)

gr_active, gr_other *(continued)*

BCD16 orgtblst — Original tblStart, before any scrolling has occurred.

BCD16 tblshift — Number of lines scrolled in table.

BCD16 tblstart — Table system variable tblStart.

BCD16 deltatbl — Table system variable Δtbl.

BCD16 *rngp — Pointer to current Window variables array:

FUNCTION mode indices:

- GR_XMIN — System variable xmin.
- GR_XMAX — System variable xmax.
- GR_XSCL — System variable xscl.
- GR_YMIN — System variable ymin.
- GR_YMAX — System variable ymax.
- GR_YSCL — System variable yscl.
- GR_DELTAX — System variable Δx.
- GR_DELTAY — System variable Δy.
- GR_XRES — System variable xres.

PARAMETRIC mode indices:

- GR_XMIN — System variable xmin.
- GR_XMAX — System variable xmax.
- GR_XSCL — System variable xscl.
- GR_YMIN — System variable ymin.
- GR_YMAX — System variable ymax.
- GR_YSCL — System variable yscl.
- GR_DELTAX — System variable Δx.
- GR_DELTAY — System variable Δy.
- GR_TMIN — System variable tmin.
- GR_TMAX — System variable tmax.
- GR_TSTEP — System variable tstep.

POLAR mode indices:

- GR_XMIN — System variable xmin.
- GR_XMAX — System variable xmax.
- GR_XSCL — System variable xscl.

(continued)

gr_active, gr_other *(continued)*

GR_YMIN — System variable ymin.
GR_YMAX — System variable ymax.
GR_YSCL — System variable yscl.
GR_DELTAX — System variable Δx.
GR_DELTAY — System variable Δy.
GR_THETMIN — System variable θmin.
GR_THETMAX — System variable θmax.
GR_THETSTEP — System variable θstep.

SEQUENCE mode indices:

GR_XMIN — System variable xmin.
GR_XMAX — System variable xmax.
GR_XSCL — System variable xscl.
GR_YMIN — System variable ymin.
GR_YMAX — System variable ymax.
GR_YSCL — System variable yscl.
GR_DELTAX — System variable Δx.
GR_DELTAY — System variable Δy.
GR_NMIN — System variable nmin.
GR_NMAX — System variable nmax.
GR_NPLOT — System variable plotStrt.
GR_NSTEP — System variable plotStep.

3D mode indices:

GR_XMIN — System variable xmin.
GR_XMAX — System variable xmax.
GR_XGRID — System variable xgrid.
GR_YMIN — System variable ymin.
GR_YMAX — System variable ymax.
GR_YGRID — System variable ygrid.
GR_DELTAX — Internal data.
GR_DELTAY — Internal data.
GR_ZMIN — System variable zmin.
GR_ZMAX — System variable zmax.
GR_ZSCL — System variable zscl. (Note that zscl is not used on TI-89 / TI-92 Plus.)

(continued)

gr_active, gr_other *(continued)*

GR_EYE_THETA — System variable eyeθ.
GR_EYE_PHI — System variable eyeϕ.
GR_EYE_PSI — System variable eyeψ.
GR_NCONTOUR — System variable ncontour.
GR_XSCALE — Internal data.
GR_YSCALE — Internal data.
GR_ZSCALE — Internal data.

DIFF EQUATIONS mode indices:

GR_XMIN — System variable xmin.
GR_XMAX — System variable xmax.
GR_XSCL — System variable xscl.
GR_YMIN — System variable ymin.
GR_YMAX — System variable ymax.
GR_YSCL — System variable yscl.
GR_DELTAX — System variable Δx.
GR_DELTAY — System variable Δy.
GR_T0 — System variable t0.
GR_TMAX — System variable tmax.
GR_TSTEP — System variable tstep.
GR_TPLOT — System variable tplot.
GR_DIFTOL — System variable diftol.
GR_ESTEP — System variable Estep.
GR_FLDRES — System variable fldres.
GR_NCURVES — System variable ncurves.
GR_DTIME — System variable dtime.

BCD16 PrevRange[12] — Current ZoomPrev values.

BCD16 UserRange[29] — Current ZoomSto values.

(continued)

gr_active, gr_other *(continued)*

GR_MODES *gr_modep — Pointer to GR_MODES struct for current graph. Members of the GR_MODES struct and their contents are:

- WORD gr_fmt_flags — Graph Format flags:
  - GR_SEQ_TIME — SEQUENCE Axes setting (see below)
  - GR_SEQ_WEB — SEQUENCE Axes setting (see below)
    - TIME = GR_SEQ_TIME set
GR_SEQ_WEB reset
    - WEB = GR_SEQ_TIME reset
GR_SEQ_WEB set
    - CUSTOM = GR_SEQ_TIME reset
GR_SEQ_WEB reset
  - GR_BUILD_WEB — SEQUENCE WEB Build Web:
TRACE = reset, AUTO = set
  - GR_3dEXPAND — Set for 3D expanded view mode.
  - GR_COORDOFF — Graph Coordinates:
OFF = set, RECT or POLAR = reset
(see GR_COORD_POLAR)
  - GR_SIMUL — Graph Order:
SEQ = reset, SIMUL = set
  - GR_GRIDON — Graph Grid:
OFF = reset, ON = set
  - GR_AXESOFF — Graph Axes:
ON = reset, OFF = set
  - GR_AXESBOX — 3D Axes:
    - OFF = GR_AXESOFF set
    - AXES = GR_AXESOFF reset
GR_AXESBOX reset
    - BOX = GR_AXESOFF reset
GR_AXESBOX set
  - GR_LABELSON — Graph Labels:
OFF = reset, ON = set
  - GR_LEAD_CURSOR — Graph Leading Cursor:
OFF = reset, ON = set

(continued)

gr_active, gr_other *(continued)*

GR_COORD_POLAR — Graph Coordinates:

POLAR = GR_COORD_POLAR set
GR_COORDOFF reset

RECT = GR_COORD_POLAR reset
GR_COORDOFF reset

SBYTE gr_xaxis — X Axis for SEQUENCE or DIFF EQUATIONS CUSTOM Axes setting:

SEQUENCE X Axis:
n = -1
u = 0
u1, u2, . . . , u99 = 1, 2, . . . , 99

DIFF EQUATIONS X Axis:
t = 0
y = 100
y1, y2, . . . , y99 = 1, 2, . . . , 99
y' = -100
y1', y2', . . . , y99' = -1, -2, . . . , -99

SBYTE gr_yaxis — Y Axis for SEQUENCE or DIFF EQUATIONS CUSTOM Axes setting

SEQUENCE Y Axis:
n = -1
u = 0
u1, u2, . . . , u99 = 1, 2, . . . , 99

DIFF EQUATIONS Y Axis:
t = 0
y = 100
y1, y2, . . . , y99 = 1, 2, . . . , 99
y' = -100
y1', y2', . . . , y99' = -1, -2, . . . , -99

WORD gr_fmt_flags2 — Graph Format flags

GR_DE_CUSTOM — DIFF EQUATIONS Axes = CUSTOM if set

(continued)

gr_active, gr_other *(continued)*

GR_DE_FIELDS — DIFF EQUATIONS Fields (see below)
GR_DIRFLD — DIFF EQUATIONS Fields (see below)
SLPFLD = GR_DE_FIELDS set
DIRFLD = GR_DE_FIELDS set
GR_DIRFLD set
FLDOFF = GR_DE_FIELDS reset
GR_DIRFLD reset
GR_EULER — DIFF EQUATIONS Solution Method: RK = reset, EULER = set
BYTE gr_3dflags — 3D Graph Style:
GR_3D_WIRE_FRAME — Wire frame.
GR_3D_HIDDEN_SURFACE — Hidden surface.
GR_3D_CONTOUR — Contour levels.
GR_3D_CONTOUR_WIRE — Wire and contour.
GR_3D_IMPLICIT — Implicit plot.
BYTE pad — Unused.
WINDOW *grwinp — Pointer to current Graph app WINDOW struct.
WINDOW *rngwinp — Pointer to current Window Editor app WINDOW struct.
WINDOW *tblwinp — Pointer to current Table app WINDOW struct.
TABLE_WIN_VARS *tableptr — Pointer to internal Table app data.
EQU_DS equedDS — Pointer to internal Y= Editor app data.
USHORT curinc — Graph iteration counter.
USHORT curincy — 3D y trace mode iteration counter.
USHORT tblindx — Index of tblInput element at top of Table.
SSHORT yaxispix — Pixel number of y axis for panning.
USHORT TBL_WidthLimit — Format width for Table app.
HANDLE zval — Handle of the 3D z value array.
DB3 DB3z — 3D function spin database.

(continued)

gr_active, gr_other *(continued)*

HANDLE htbinput	—	Handle of the table system variable tblInput.
HANDLE hfldpic	—	Handle of the graph system variable fldpic.
WORD gr_win_flags	—	Graph app flags:
GR_REDRAW	—	Redraw 3D graph without recomputing.
GR_DIRTY	—	Current graph needs to be recomputed.
TAB_DIRTY	—	Current table needs to be recomputed.
GR_ADD_TO	—	Add a function to the current graph without recomputing.
GR_OPEN	—	Current graph window is open.
GRAPH_FOLDER	—	Temporary folder for functions created by the Graph and Table commands exists.
EYE_DIRTY	—	The eye of the 3D graph has changed.
GR_SHADE_NO_PAN	—	Panning is not valid after shading.
FLDPIC_DIRTY	—	System variable fldpic needs to be recomputed.
BYTE xmaxpix	—	Rightmost column used by graph in current window.
BYTE ymaxpix	—	Bottom row used by graph in current window.
BYTE gr_ref_mask	—	Graph reference flag mask for current graph.
BYTE graph_mode	—	Graph mode of current graph:
GR_FUNC	—	FUNCTION mode.
GR_PAR	—	PARAMETRIC mode.
GR_POL	—	POLAR mode.
GR_SEQ	—	SEQUENCE mode.
GR_3D	—	3D mode.
GR_DE	—	DIFF EQUATIONS mode.

(continued)

gr_active, gr_other *(continued)*

BYTE gr_side	—	Graph window location:
AP_SIDE_A	—	Top or left split.
AP_SIDE_B	—	Bottom or right split.
BYTE gr_folder_cnt	—	Number of functions created by the TI-BASIC Graph and Table commands.
BYTE gr_shade_pat	—	Shade pattern:
A_SHADE_V	—	Vertical.
A_SHADE_H	—	Horizontal.
A_SHADE_NS	—	Negative slope 45°.
A_SHADE_PS	—	Positive slope 45°.
BYTE rng_xpix	—	Maximum x pixel number on Window Editor screen.
BYTE rng_ypix	—	Maximum y pixel number on Window Editor screen.
BYTE tbl_flags	—	Table app flags:
TBL_CONNECT_TRC	—	Set when Graph<->Table = ON.
TBL_INDEP_ASK	—	Set when Independent = ASK.
TBL_NO_MODE_CHANGE	—	Set when executing DispTbl command.
BYTE tbl_par_flags	—	Internal Table app flags.
BYTE gr_top_flags	—	Internal Graph app flags.
BYTE ValidCursBits	—	Internal Graph app flags.
SBYTE de_twopass	—	Internal Graph app flags.
FUNCID CurFunc	—	Data for currently selected function for tracing.
BYTE PrevZoomMode	—	Graph mode of current ZoomPrev values.

(continued)

gr_active, gr_other *(continued)*

Inputs: None

Outputs: None

Assumptions: Access_AMS_Global_Variables is defined.

Side Effects: None

Availability: All versions of the TI-89 / TI-92 Plus.

TI-89 / TI-92 Plus Differences: None

See Also: None

Example:

```
/* Given a floating point independent value and a pointer to its graph Window
   variables min, max, and step values, ICvtFtoP returns the corresponding increment
   or 0xFFFF if the independent value is out of range.
*/
WORD ICvtFtoP( BCD16 f, BCD16 *indep_rng )
{  Access_AMS_Global_Variables;
   WORD NewInc;

   switch (gr_active->graph_mode) {
      case GR_FUNC:
         return( (WORD) XCvtFtoP(f, gr_active) );
      case GR_DE:
      case GR_SEQ:
      case GR_PAR:
      case GR_POL:
         if (gr_CptIndepInc(f, indep_rng, &NewInc))
            return( NewInc );
      case GR_3D:
         return ((f - gr_active->rngp[GR_XMIN]) * (gr_active->rngp[GR_XGRID] /
                 (gr_active->rngp[GR_XMAX] - gr_active->rngp[GR_XMIN])));
   }
   return 0xFFFF;
}
```

gr_flags

Declaration: GR_FLAGS **gr_flags**

Category: Graphing

Description: Global flags used by the Graph application. Each flag is a separate struct member. The contents of these flags should not be changed by an app or ASM but may be accessed for testing the value. The names of the flags and their purposes are:

BOOL gr_in_progress	A graph is currently being plotted. Among other things, this flag alerts **VarRecall** to set the graph reference flag for every user variable accessed until this flag is reset to enable the Smart Graph feature to work.
BOOL gr_zoom_fit	ZoomFit is being executed. Every graph point is computed to determine the min and max Window variable values, but while this flag is set, nothing is plotted.
BOOL gr_cpt_seq_flag	A graph sequence mode function (u1 – u99) is being executed.
BOOL stat_in_progress	A statistics calculation is currently being performed. Among other things, this flag alerts **VarRecall** to set the stat reference flag for every user variable accessed until this flag is reset to enable the calculator to determine when the statistics results are no longer valid.
BOOL gr_trace_seq	A sequence function is being traced.
BOOL de_init_conds	A differential equation is being plotted with initial conditions selected interactively using the graph cursor.
BOOL gr_cpt_de_flag	A graph differential equation mode function (y1' – y99') is being executed.
BOOL new_eqn	The Numeric Solver system variable eqn has changed. This alerts the solver graph to regraph.
BOOL de_error	An error has occurred while computing a graph differential equation mode function.

Inputs: None

Outputs: None

(continued)

gr_flags *(continued)*

Assumptions: Access_AMS_Global_Variables is defined.

Side Effects: None

Availability: On AMS 2.00 and higher.

TI-89 / TI-92 Plus Differences: None

See Also: None

Example: The function below is an example of a TI-BASIC extension function. See section **7.3.2 TI-BASIC Extensions** for more information on extensions.

```
void graphsin( void )
/* Return on the estack a list of all the values of sin that would be graphed
   with the current Window variables in function mode, or if called during graphing,
   return only the value of sin for the input.
*/
{
   Access_AMS_Global_Variables;
   BCD16 val, num;
   USHORT ctr;
   EStackIndex k, old_top = top_estack;

   if( gr_flags.gr_in_progress == TRUE )
   {    /* graph in progress - return only one value to plot */
      push_approx( top_estack );     /* simplify the input to the function */
      if( ESTACK(top_estack) != FLOAT_TAG )
         ER_throw( ER_DOMAIN );
      val = sin( estack_to_float( top_estack ) );
      top_estack = old_top;       /* remove simplified input */
      push_Float( val );      /* leave result on estack */
      return;
   }
   /* Not graphing, ignore any input on estack, return list of sin values
      that would be graphed */
   if( gr_active->graph_mode != GR_FUNC )
      ER_throw( ER_GRAPH_MODE );    /* must be function mode to access xres */
   push_quantum( END_TAG );
   for( ctr=0; ctr <= gr_active->xmaxpix; ctr +=
               (USHORT)((gr_active->rngp)[GR_XRES] ))
   {
      num = (gr_active->rngp)[GR_XMIN] + ((gr_active->rngp)[GR_DELTAX] ) *
                                ctr;   /* next input */
      val = sin( num );    /* next result */
      push_Float( val );   /* build list on estack */
   }
   k = top_estack;               /* point to last sin value */
   push_reversed_tail (k);                  /* reverse the order */
   delete_between (old_top, k);             /* delete the old copy */
   push_quantum (LIST_TAG);                 /* make it a list */
}
```

OSFastArrows

Declaration: BYTE **OSFastArrows**

Category(ies): Keyboard

Description: Normally set to 0.

Once a key value is pushed onto the key queue, the same key value is not pushed again until the key is released, unless that key is one of the following: any of the arrow keys, the contrast keys, delete, or backspace. These keys are allowed to “auto-repeat”. If one of these keys is pressed and held, after an initial delay the same key value will be pushed again. If the keypress continues to be active, the key value will continue to be pushed at a rate set by a delay which is slightly shorter than the initial delay. If a key is pushed as a result of auto-repeat, the value KB_AUTOREPEAT is OR’d with the key value prior to pushing the key value onto the key queue.

If **OSFastArrows** is zero, **ngetchx** will clear the KB_AUTOREPEAT bit from the key value.

If **OSFastArrows** is set to 2, the in-between-key delay is ignored for arrow keys, allowing them to be pushed as fast as the keyboard can be scanned.

Inputs: Not applicable.

Outputs: Not applicable.

Assumptions: Not applicable.

Side Effects: Not applicable.

Availability: All versions of the TI-89 / TI-92 Plus.

TI-89 / TI-92 Plus Differences: None

See Also: **Ngetchx, OSInitBetweenKeyDelay, OSInitKeyInitDelay**

Example:

```
TRY
   SaveFastArrows = OSFastArrows;
   OSFastArrows = 0;
   .
   .   /* do some stuff */
   .

ONERR
   OSFastArrows = SaveFastArrows;
```

OSModKeyStatus

Declaration: WORD **OSModKeyStatus**

Category(ies): Keyboard

Description: Reflects the status of the modifier keys. Set equal to the modifier key indicator in status line.

Inputs: Not applicable.

Outputs: Not applicable.

Assumptions: Not applicable.

Side Effects: Not applicable.

Availability: All versions of the TI-89 / TI-92 Plus.

TI-89 / TI-92 Plus Differences: None

See Also: None

Example:

```
switch (OSModKeyStatus) {
   case SC_SECOND:
      // second key modifier is active
      break;
   case SC_OPTION:
      // diamond key modifier is active
      break;
   case SC_SHIFT:
      // shift key modifier is active
      break;
   case SC_DRAG:
      // grab key modifier is active
      break;
   case SC_CAPSLOCK:
      // shift lock modifier is active
      break;
   case SC_DRAGLOCK:
      // drag lock modifier is active
      break;
   default:
      // no key modifier is active
}
```

index_false

Declaration: EStackIndex **index_false**

Category(ies): Logic

Description: EStackIndex of the stored constant false.

Inputs: Not applicable.

Outputs: Not applicable.

Assumptions: Not applicable.

Side Effects: Not applicable.

Availability: On AMS 2.04 and higher.

TI-89 / TI-92 Plus Differences: None

See Also: **index_true**

Example:

```
push_expression (index_false);
```

Pushes the logical constant false onto the estack such that **top_estack** points to the bolded tag as follows.
FALSE_TAG

index_true

Declaration: EStackIndex **index_true**

Category(ies): Logic

Description: EStackIndex of the stored constant true.

Inputs: Not applicable.

Outputs: Not applicable.

Assumptions: Not applicable.

Side Effects: Not applicable.

Availability: On AMS 2.04 and higher.

TI-89 / TI-92 Plus Differences: None

See Also: **index_false**

Example:

```
push_expression (index_true);
```

Pushes the logical constant true onto the estack such that **top_estack** points to the bolded tag as follows.
TRUE_TAG

Float0Index

Declaration: EStackIndex **Float0Index**

Category(ies): Math

Description: EStackIndex of the stored constant float 0.0.

Inputs: Not applicable.

Outputs: Not applicable.

Assumptions: Not applicable.

Side Effects: Not applicable.

Availability: On AMS 2.04 and higher.

TI-89 / TI-92 Plus Differences: None

See Also: **Float1Index, FloatExp1Index, FloatHalfIndex, FloatMinus1Index, FloatPiIndex**

Example:

```
push_expression (Float0Index);
```

Pushes a float 0.0 onto the estack such that **top_estack** points to the bolded tag as follows.
0x40 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 **FLOAT_TAG**

Float1Index

Declaration: EStackIndex **Float1Index**

Category(ies): Math

Description: EStackIndex of the stored constant float 1.0.

Inputs: Not applicable.

Outputs: Not applicable.

Assumptions: Not applicable.

Side Effects: Not applicable.

Availability: On AMS 2.04 and higher.

TI-89 / TI-92 Plus Differences: None

See Also: **Float0Index, FloatExp1Index, FloatHalfIndex, FloatMinus1Index, FloatPiIndex**

Example:

```
push_expression (Float1Index);
```

Pushes a float 1.0 onto the estack such that **top_estack** points to the bolded tag as follows.
0x40 0x00 0x10 0x00 0x00 0x00 0x00 0x00 0x00 **FLOAT_TAG**

FloatExp1Index

Declaration: EStackIndex **FloatExp1Index**

Category(ies): Math

Description: EStackIndex of the stored constant float approximation of e^1.

Inputs: Not applicable.

Outputs: Not applicable.

Assumptions: Not applicable.

Side Effects: Not applicable.

Availability: On AMS 2.04 and higher.

TI-89 / TI-92 Plus Differences: None

See Also: **Float0Index, Float1Index, FloatHalfIndex, FloatMinus1Index, FloatPiIndex**

Example:

```
push_expression (FloatExp1Index);
```

Pushes a float approximation of e^1 onto the estack such that **top_estack** points to the bolded tag as follows.
0x40 0x00 0x27 0x18 0x28 0x18 0x28 0x45 0x90 **FLOAT_TAG**

FloatHalfIndex

Declaration: EStackIndex **FloatHalfIndex**

Category(ies): Math

Description: EStackIndex of the stored constant float 0.5.

Inputs: Not applicable.

Outputs: Not applicable.

Assumptions: Not applicable.

Side Effects: Not applicable.

Availability: On AMS 2.04 and higher.

TI-89 / TI-92 Plus Differences: None

See Also: **Float0Index, Float1Index, FloatExp1Index, FloatMinus1Index, FloatPiIndex**

Example:

```
push_expression (FloatHalfIndex);
```

Pushes a float 0.5 onto the estack such that **top_estack** points to the bolded tag as follows. 0x3F 0xFF 0x50 0x00 0x00 0x00 0x00 0x00 0x00 **FLOAT_TAG**

FloatMinus1Index

Declaration: EStackIndex **FloatMinus1Index**

Category(ies): Math

Description: EStackIndex of the stored constant float -1.

Inputs: Not applicable.

Outputs: Not applicable.

Assumptions: Not applicable.

Side Effects: Not applicable.

Availability: On AMS 2.04 and higher.

TI-89 / TI-92 Plus Differences: None

See Also: **Float0Index, Float1Index, FloatExp1Index, FloatHalfIndex, FloatPiIndex**

Example:

```
push_expression (FloatMinus1Index);
```

Pushes a float -1 onto the estack such that **top_estack** points to the bolded tag as follows. 0xC0 0x00 0x10 0x00 0x00 0x00 0x00 0x00 0x00 **FLOAT_TAG**

FloatPiIndex

Declaration: EStackIndex **FloatPiIndex**

Category(ies): Math

Description: EStackIndex of the stored constant float approximation of π.

Inputs: Not applicable.

Outputs: Not applicable.

Assumptions: Not applicable.

Side Effects: Not applicable.

Availability: On AMS 2.04 and higher.

TI-89 / TI-92 Plus Differences: None

See Also: **Float0Index, Float1Index, FloatExp1Index, FloatHalfIndex, FloatMinus1Index**

Example:

```
push_expression (FloatPiIndex);
```

Pushes the float approximation of π onto the estack such that **top_estack** points to the bolded tag as follows.
0x40 0x00 0x31 0x41 0x59 0x26 0x53 0x58 0x98 **FLOAT_TAG**

Integer0Index

Declaration: EStackIndex **Integer0Index**

Category(ies): Math

Description: EStackIndex of the stored constant integer 0.

Inputs: Not applicable.

Outputs: Not applicable.

Assumptions: Not applicable.

Side Effects: Not applicable.

Availability: On AMS 2.04 and higher.

TI-89 / TI-92 Plus Differences: None

See Also: **Integer1Index, Integer2Index, IntegerMinus1Index**

Example:

```
push_expression (Integer0Index);
```

Pushes an integer zero onto the estack as follows.
0 **NONNEGATIVE_INTEGER_TAG**

Integer1Index

Declaration: EStackIndex **Integer1Index**

Category(ies): Math

Description: EStackIndex of the stored constant integer 1.

Inputs: Not applicable.

Outputs: Not applicable.

Assumptions: Not applicable.

Side Effects: Not applicable.

Availability: On AMS 2.04 and higher.

TI-89 / TI-92 Plus Differences: None

See Also: **Integer0Index, Integer2Index, IntegerMinus1Index**

Example:

When the second argument of the **push_shift** function is positive, the shift is to the left. So, if m is the EStackIndex at the bolded tag in the string “goodbye” as follows
0 g o o d b y e 0 **STR_DATA_TAG**

then

```
push_shift (m, Integer1Index);
```

pushes the left shifted string “oodbye ” onto the estack such that **top_estack** points to the bolded tag as follows.
0 o o d b y e ␣ 0 **STR_DATA_TAG**

Integer2Index

Declaration: EStackIndex **Integer2Index**

Category(ies): Math

Description: EStackIndex of the stored constant integer 2.

Inputs: Not applicable.

Outputs: Not applicable.

Assumptions: Not applicable.

Side Effects: Not applicable.

Availability: On AMS 2.04 and higher.

TI-89 / TI-92 Plus Differences: None

See Also: **Integer0Index, Integer1Index, IntegerMinus1Index**

Example:

When the second argument of the **push_shift** function is positive, the shift is to the left. So, if m is the EStackIndex at the bolded tag in the string “goodbye” as follows
0 g o o d b y e 0 **STR_DATA_TAG**

then

```
push_shift (m, Integer2Index);
```

pushes the left shifted string “odbye ” onto the estack such that **top_estack** points to the bolded tag as follows.
0 o d b y e ␣ ␣ 0 **STR_DATA_TAG**

IntegerMinus1Index

Declaration: EStackIndex **IntegerMinus1Index**

Category(ies): Math

Description: EStackIndex of the stored constant integer -1.

Inputs: Not applicable.

Outputs: Not applicable.

Assumptions: Not applicable.

Side Effects: Not applicable.

Availability: On AMS 2.04 and higher.

TI-89 / TI-92 Plus Differences: None

See Also: **Integer0Index, Integer1Index, Integer2Index**

Example:

When the second argument of the **push_shift** function is negative, the shift is to the right. So, if m is the EStackIndex at the bolded tag in the string “goodbye” as follows
0 g o o d b y e 0 **STR_DATA_TAG**

then

```
push_shift (m, IntegerMinus1Index);
```

pushes the right shifted string “ goodby” onto the estack such that **top_estack** points to the bolded tag as follows.
0 ␣ g o o d b y 0 **STR_DATA_TAG**

MO_option

Declaration: WORD **MO_option** [MO_OPT_ITEM_COUNT]

Category(ies): Mode Screen Settings

Description: This global array is the buffer used to communicate mode settings between OS routines **MO_currentOptions** and **MO_digestOptions** and the Mode screen dialog box. Symbols are defined in tiams.h which index **MO_option** and define permissible values for each mode setting.

MO_OPT_SPLIT_SCREEN
Split Screen mode setting, how the screen is divided to display app windows.

D_MODE_SPLIT_FULL
FULL, full screen (only one app window).
D_MODE_SPLIT_HORIZONTAL
TOP-BOTTOM, horizontal split, one window above the other.
D_MODE_SPLIT_VERTICAL
LEFT-RIGHT, vertical split, windows side by side.

MO_OPT_NUMBER_OF_GRAPHS
Number of Graphs mode setting, 1 or 2.

MO_OPT_GRAPH_TYPE_1
GRAPH mode setting, type of graph 1.

GR_FUNC — FUNCTION mode
GR_PAR — PARAMETRIC mode
GR_POL — POLAR mode
GR_SEQ — SEQUENCE mode
GR_3D — 3D mode
GR_DE — DIFF EQUATIONS mode

MO_OPT_GRAPH_TYPE_2
GRAPH 2 mode setting, type of graph 2. This value is only used if MO_OPT_NUMBER_OF_GRAPHS is 2. It takes the same values as MO_OPT_GRAPH_TYPE_1.

MO_OPT_SPLIT_1
Split 1 App, ID of app in window 1. Window 1 is the only window in D_MODE_SPLIT_FULL mode, the upper window in horizontal window splits or left window in vertical window splits.

(continued)

MO_option *(continued)*

Description: *(continued)*

MO_OPT_SPLIT_2

Split 2 App, ID of app in window 2. Window 2 is the lower window in horizontal window splits or right window in vertical window splits.

MO_OPT_SPLIT_RATIO

Split Screen Ratio mode setting, how screen area is divided between windows in split screen mode.

D_SPLIT_RATIO_1_1

1:1, each window gets half the screen.

D_SPLIT_RATIO_1_2

1:2, the left (or upper) window gets $^1/_3$ of the screen, the right (or lower) window gets $^2/_3$ of the screen (TI‑92 Plus only).

D_SPLIT_RATIO_2_1

2:1, the left (or upper) window gets $^2/_3$ of the screen, the right (or lower) window gets $^1/_3$ of the screen (TI‑92 Plus only).

MO_OPT_ANGLE

Angle mode setting, trigonometric angle calculation mode.

D_ANGLE_RAD	—	RADIAN
D_ANGLE_DEGREE	—	DEGREE

MO_OPT_PRECISION

Exact/Approx mode setting.

D_PREC_AUTO	—	AUTO mode (automatically select between exact and approximate mode)
D_PREC_RATIONAL	—	EXACT
D_PREC_APPROX	—	APPROXIMATE

MO_OPT_FIX

Display Digits mode setting.

D_PREC_FIX_0 through D_PREC_FIX_12

Fixed point display with 0 through 12 digits displayed after the decimal point.

D_PREC_FLOAT

Floating point, trailing zeros suppressed.

D_PREC_FLOAT_1 through D_PREC_FLOAT_12

Floating point, trailing zeros suppressed, result rounded to display precision digits.

(continued)

MO_option *(continued)*

Description: *(continued)*

MO_OPT_NUMBER_FORMAT
Exponential Format mode setting.
- D_EXP_FORMAT_NORMAL — NORMAL
- D_EXP_FORMAT_SCI — SCIENTIFIC
- D_EXP_FORMAT_ENG — ENGINEERING

MO_OPT_VECTOR_FORMAT
Vector Format mode setting.
- D_VECT_RECT — RECTANGULAR
- D_VECT_CYL — CYLINDRICAL
- D_VECT_SPH — SPHERICAL

MO_OPT_COMPLEX_FORMAT
Complex Format mode setting.
- D_COMPLEX_OFF — REAL
- D_COMPLEX_RECT — RECTANGULAR
- D_COMPLEX_POLAR — POLAR

MO_OPT_PRETTY_PRINT
Pretty Print mode setting.
- D_OFF — OFF
- D_ON — ON

MO_OPT_BASE
Base mode setting.
- D_DEC — DEC
- D_HEX — HEX
- D_BIN — BIN

MO_OPT_UNIT_SYSTEM
Unit System mode setting.
- D_UNIT_SI — SI
- D_UNIT_US — ENG/US
- D_UNIT_CUSTOM — CUSTOM

MO_OPT_LANGUAGE
Language mode setting, the ID of the current language localizer app.

Inputs: None

Outputs: None

Assumptions: Access_AMS_Global_Variables must be defined.

Side Effects: None

(continued)

MO_option *(continued)*

Availability: All versions of the TI-89 / TI-92 Plus.

TI-89 / TI-92 Plus Differences: None

See Also: **MO_currentOptions, MO_digestOptions**

Example:

```
Access_AMS_Global_Variables;
WORD angleMode;
.
.
.
MO_currentOptions();
angleMode = MO_option[MO_OPT_ANGLE];  /* Get current angle mode setting */
```

EV_flags

Declaration: EV_FLAGS **EV_flags**

Category(ies): Operating System

Description: The event manager maintains state information in the bits of **EV_flags**.

EV_OFF	—	The default event handler sets this flag when the [2nd] [OFF] key is pressed. Macro EV_TST_OFF returns TRUE if this flag is set. When an application receives the CM_QUIT event, it can test the EV_OFF flag to determine if the calculator is being turned off. The EV_OFF flag will never be set when any other events are sent to an app.
EV_SUSPEND_PAINTING	—	This flag is set by **EV_suspendPainting** and restored by **EV_restorePainting**. The event manager quits sending CM_WPAINT messages when this flag is set, and resumes when this flag is clear.

Inputs: Not applicable.

Outputs: Not applicable.

Assumptions: Access_AMS_Global_Variables is defined.

Side Effects: Not applicable.

Availability: On AMS 2.04 and higher.

TI-89 / TI-92 Plus Differences: None

See Also: **EV_captureEvents, EV_suspendPainting, EV_restorePainting**

Example: See the example for **EV_captureEvents**.

RM_Type

Declaration: BYTE **RM_Type**

Category(ies): Statistics, Variables

Description: Global variable that specifies the type of the current stat operation.

Inputs: The possible valid values are: RM_NONE, RM_MEDMED, RM_LIN, RM_LN, RM_EXP, RM_POWER, RM_QUAD, RM_CUBIC, RM_QUART, RM_LOGISTIC, RM_SIN

Outputs: None

Assumptions: Access_AMS_Global_Variables must be defined.

Side Effects: Determines how **cmd_showstat** displays the statistics variables.

Availability: On AMS 2.04 and higher.

TI-89 / TI-92 Plus Differences: None

See Also: **cmd_showstat**

Example: See **statStart**.

ST_flags

Declaration: ST_FLAGS **ST_flags**

Category(ies): Status Line

Description: This variable contains flag bits which describe the current state of the status line indicators.

Bit	Shift	Mask	Indicator
0		ST_2ND	2ND
1		ST_SHIFT ST_DRAG	⇧
2		ST_OPTION	◆
3		ST_ALPHA	a
4		ST_CAPSLOCK	🔒
5		ST_DRAGLOCK ST_ALPHALOCK	a / 🔒
6	ST_ANGLE_SHIFT	ST_ANGLE	ST_RAD = RAD ST_DEG = DEG
7 – 8	ST_PRECISION_SHIFT	ST_PRECISION	ST_AUTO = AUTO ST_RATNL = EXACT ST_APPROX = APPROX
9	ST_GRAPH_SIDE_SHIFT	ST_GRAPH_SIDE	0 = GR1 1 = GR2
10 – 12	ST_GRAPH_TYPE_SHIFT	ST_GRAPH_TYPE	0 = FUNC 1 = PAR 2 = POL 3 = SEQ 4 = 3D 5 = DE
13 – 14	ST_BUSY_SHIFT	ST_BUSY_INDIC	ST_IDLE = no indicator ST_BUSY = BUSY ST_PAUSE = PAUSE
15	ST_HELP_SHIFT	ST_HELP	help message in status line
16	ST_CHANGED_SHIFT	ST_CHANGED	status line has changed
17 – 18	ST_BATTERY_SHIFT	ST_BATTERY	ST_BATT_OFF = no indicator ST_BATT_ALERT1 = BATT ST_BATT_ALERT2 = BATT
19	ST_READONLY_SHIFT	ST_READONLY	🔒
20	ST_INIT_SHIFT	ST_INIT	status line initialized

(continued)

ST_flags *(continued)*

Description: Note that bits 1-5 are different on the TI-92 Plus. See table below.

(continued)

Bit	Shift	Mask	Indicator
1		ST_OPTION	
2		ST_SHIFT	
3		ST_DRAG	
4		ST_CAPSLOCK	no indicator
5		ST_DRAGLOCK	

Inputs: Not applicable.

Outputs: Not applicable.

Assumptions: Not applicable.

Side Effects: Not applicable.

Availability: On AMS 2.00 and higher.

TI-89 / TI-92 Plus Differences: The shift, alpha, option, and drag indicator flags are different between the two platforms. See flags 1 – 5 in the table above for details.

See Also: Not applicable

Example:

```
/* Is help displayed in the status line? */
if (ST_flags & ST_HELP)
{
   /* Yes, . . .  */
}

/* Get the current graph type from the status line */
grtype = (ST_flags & ST_GRAPH_TYPE) >> ST_GRAPH_TYPE_SHIFT;
```

RF_ . . .

Declaration:
#define **RF_NUL** "\x00"
#define **RF_SOH** "\x01"
.
.
.
#define **RF_Y_UMLAUT** "\xFF"

Category(ies): Strings

Description: The equates SF_ . . . , LF_ . . . , and HF_ . . . define the characters available for the small, large, and huge fonts as numeric values. The RF_ . . . equates also define the characters but as strings which may be used in other strings to embed a particular character.

Inputs: Not applicable.

Outputs: Not applicable.

Assumptions: Not applicable.

Side Effects: Not applicable.

Availability: All versions of the TI-89 / TI-92 Plus.

TI-89 / TI-92 Plus Differences: None

See Also: None

Example: This example embeds the copyright character in a string.

```
sprintf(msgbuf, "%s " RF_COPYRIGHT " 2000 %s.", XR_stringPtr(XR_Copyright),
XR_stringPtr(XR_TexasInstruments));
```

FiftyMsecTic

Declaration: DWORD **FiftyMsecTic**

Category(ies): Timer

Description: This variable is incremented approximately every 50 milliseconds by the OS.

Inputs: Not applicable.

Outputs: Not applicable.

Assumptions: Not applicable.

Side Effects: Not applicable.

Availability: All versions of the TI-89 / TI-92 Plus.

TI-89 / TI-92 Plus Differences: None

See Also: None

Example:

```
// Spin for about a second
DWORD numberOfTicks;
int i;
numberOfTicks = FiftyMsecTic;   // Get a starting number
do {
   asm(“nop”, 2);
}  while ((numberOfTicks = FiftyMsecTick – NumberOfTicks) < 20);
```

ReleaseDate

Declaration: char * **ReleaseDate**

Category(ies): Utilities

Description: Global string pointer to the release date of the current AMS code.

Inputs: None

Outputs: Pointer to current release date.

Assumptions: Access_AMS_Global_Variables must be defined.

Side Effects: None

Availability: On AMS 2.00 or later.

TI-89 / TI-92 Plus Differences: None

See Also: **ReleaseVersion**

Example:

```
Access_AMS_Global_Variables;
char Buf[80];

sprintf( Buf, "Release date: %s\nVersion: %s\n", ReleaseDate, ReleaseVersion );
DlgNotice( "AMS", Buf );
```

ReleaseVersion

Declaration: char * **ReleaseVersion**

Category(ies): Utilities

Description: Global string pointer to the release version of the current AMS code.

Inputs: None

Outputs: Pointer to current release version.

Assumptions: Access_AMS_Global_Variables must be defined.

Side Effects: None

Availability: On AMS 2.00 or later.

TI-89 / TI-92 Plus Differences: None

See Also: **ReleaseDate**

Example: See **ReleaseDate**.

Appendix C: Macros

isalnum

Declaration: BYTE **isalnum** (BYTE *c*)

Category(ies): Character Classification / Conversion

Description: Return non-zero (true) if the given character is alpha-numeric.

Inputs: *c* — Character to test.

Outputs: Non-zero if *c* is alpha-numeric, zero otherwise.

Assumptions: Access_AMS_Global_Variables is defined.

Side Effects: None

Availability: AMS 2.00 or above.

TI-89 / TI-92 Plus Differences: None

See Also: **isalpha, isdigit**

Example: This example function returns 0 . . . 8 if passed the digits '1' . . . '9' and 9 . . . 34 if passed 'A' . . . 'Z' or 'a' . . . 'z'; all other characters return -1. Note that the **isascii** function prevents any of the international alphabetical characters from being used.

```
SINT SubChar2I( SINT c )
{  Access_AMS_Global_Variables;
   if (isascii(c)) {
      c = toupper(c);
      if (isalnum(c) && c != '0')
         return( ((c < 'A') ? (c - '1') : (c - ('0' + 'A'-'9'))) );
   }
   return -1;
}
```

isalpha

Declaration: BYTE **isalpha** (BYTE *c*)

Category(ies): Character Classification / Conversion

Description: Return non-zero (true) if the given character is alphabetic (A . . . Z, and all of the international alphabetic characters).

Inputs: *c* — Character to test.

Outputs: Non-zero if *c* is alphabetic, zero otherwise.

Assumptions: Access_AMS_Global_Variables is defined.

Side Effects: None

Availability: AMS 2.00 or above.

TI-89 / TI-92 Plus Differences: None

See Also: **isalnum**

Example: This example function converts a symbol name into tokenized format and stores it in TokName. IT DOES NOT HANDLE RESERVED NAMES! TokName must point to a buffer of MAX_SYM_LEN bytes, the tokenized named is stored there starting at the end of the buffer.

```
BYTE *StrToTokN( BYTE *StrSymName, BYTE *TokName )
{  Access_AMS_Global_Variables;
   BYTE c, *TokPtr = TokName+MAX_SYM_LEN;

   c = tolower(*StrSymName);
   if ((*StrSymName+1) == '\0' && isalpha(c) && isascii(c))
      *TokPtr = ENCODE_LETTER(*StrSymName);
   else {
      TokPtr -= (short) (strlen((char *) StrSymName) + 2);
      *TokPtr++= '\0';
      strcpy( (char *) TokPtr, (char *) StrSymName );
   }
   return TokName + (MAX_SYM_LEN - 1);
}
```

isascii

Declaration: BYTE **isascii** (BYTE *c*)

Category(ies): Character Classification / Conversion

Description: Return non-zero (true) if the given character is an ASCII character (in the range 0 through 0x7F).

Inputs: *c* — Character to test.

Outputs: Non-zero if *c* is an ASCII character, zero otherwise.

Assumptions: Access_AMS_Global_Variables is defined.

Side Effects: None

Availability: AMS 2.00 or above.

TI-89 / TI-92 Plus Differences: None

See Also: **toascii**

Example: See **isalpha**.

iscsym

Declaration: BYTE **iscsym** (BYTE *c*)

Category(ies): Character Classification / Conversion

Description: Return non-zero (true) if the given character is a valid character in a symbol.

Inputs: *c* — Character to test.

Outputs: Non-zero if *c* is a valid character in a symbol, zero otherwise.

Assumptions: Access_AMS_Global_Variables is defined.

Side Effects: None

Availability: AMS 2.00 or above.

TI-89 / TI-92 Plus Differences: None

See Also: **iscsymf**

Example:

iscsymf

Declaration: BYTE **iscsymf** (BYTE *c*)

Category(ies): Character Classification / Conversion

Description: Return non-zero (true) if the given character is a valid character to start a symbol.

Inputs: *c* — Character to test.

Outputs: Non-zero if *c* is a valid first character in a symbol, zero otherwise.

Assumptions: Access_AMS_Global_Variables is defined.

Side Effects: None

Availability: AMS 2.00 or above.

TI-89 / TI-92 Plus Differences: None

See Also: **iscsym**

Example: In VAR-LINK, if a key is pressed and that key can start a symbol then the current position is moved to the symbol starting with that character as shown in the following code fragment.

```
if (Key <= 0xFF && iscsymf(Key))
   Key = tolower(Key);
```

isdigit

Declaration: BYTE **isdigit** (BYTE *c*)

Category(ies): Character Classification / Conversion

Description: Return non-zero (true) if the given character is a digit (‘0’ . . . ‘9’).

Inputs: *c* — Character to test.

Outputs: Non-zero if *c* is a digit, zero otherwise.

Assumptions: Access_AMS_Global_Variables is defined.

Side Effects: None

Availability: AMS 2.00 or above.

TI-89 / TI-92 Plus Differences: None

See Also: **isalnum**

Example:

isgreek

Declaration: BYTE **isgreek** (BYTE *c*)

Category(ies): Character Classification / Conversion

Description: Return non-zero (true) if the given character is a Greek character (alpha, beta, gamma, delta, epsilon, zeta, theta, lambda, xi, pi, rho, sigma, tau, phi, psi, omega, or mu).

Inputs: *c* — Character to test.

Outputs: Non-zero if *c* is a Greek character, zero otherwise.

Assumptions: Access_AMS_Global_Variables is defined.

Side Effects: None

Availability: AMS 2.00 or above.

TI-89 / TI-92 Plus Differences: None

See Also: **isdigit, isalpha, isascii**

Example:

islower

Declaration: BYTE **islower** (BYTE *c*)

Category(ies): Character Classification / Conversion

Description: Return non-zero (true) if the given character is a lower-case alphabetic character (including the international alphabetic characters).

Inputs: *c* — Character to test.

Outputs: Non-zero if *c* is a lower-case alphabetic, zero otherwise.

Assumptions: Access_AMS_Global_Variables is defined.

Side Effects: None

Availability: AMS 2.00 or above.

TI-89 / TI-92 Plus Differences: None

See Also: **isalnum, isupper**

Example: The **toupper** function uses **islower** to check for lower-case characters as defined below. Because of this, its arguments can be evaluated twice (once by **islower** and once by the _**toupper** MACRO).

```
#define toupper(c)  ((islower(c)) ? _toupper(c) : (c))
```

isprint

Declaration: BYTE **isprint** (BYTE *c*)

Category(ies): Character Classification / Conversion

Description: Return non-zero (true) if the given character is a printable character. The only non-printable characters are the codes 0 . . . 0xA, and 0xC, 0xD.

Inputs: *c* — Character to test.

Outputs: Non-zero if *c* is a printable character, zero otherwise.

Assumptions: Access_AMS_Global_Variables is defined.

Side Effects: None

Availability: AMS 2.00 or above.

TI-89 / TI-92 Plus Differences: None

See Also:

Example: The text-draw menu function in the grapher uses **isprint** to determine which characters can be displayed on the graph screen as shown below.

```
WORD inKey;

.
.
.
/* inKey is the key-code for the key pressed by the user */
if (inKey < 0xFF && isprint(inKey)) {
   WinAttr( w, A_REPLACE );
   WinCharXY( w, x, y, (char) inKey, 1 );
.
.
.
```

isupper

Declaration: BYTE **isupper** (BYTE *c*)

Category(ies): Character Classification / Conversion

Description: Return non-zero (true) if the given character is a upper-case alphabetic character (including the international alphabetic characters).

Inputs: *c* — Character to test.

Outputs: Non-zero if *c* is a upper-case alphabetic, zero otherwise.

Assumptions: Access_AMS_Global_Variables is defined.

Side Effects: None

Availability: AMS 2.00 or above.

TI-89 / TI-92 Plus Differences: None

See Also: **isalnum, islower**

Example: The **tolower** function uses **isupper** to check for upper-case characters as defined below. Because of this, its arguments can be evaluated twice (once by **isupper** and once by the _**tolower** MACRO).

```
#define tolower(c)  ((isupper(c)) ? _tolower(c) : (c))
```

toascii

Declaration: BYTE **toascii** (BYTE *c*)

Category(ies): Character Classification / Conversion

Description: Mask off all but the lower seven bits of a character.

Inputs: *c* — Character to convert.

Outputs: Converted character.

Assumptions: Access_AMS_Global_Variables is defined.

Side Effects: None

Availability: AMS 2.00 or above.

TI-89 / TI-92 Plus Differences: None

See Also: **isascii**

Example:

tolower

Declaration: BYTE **tolower** (BYTE *c*)

Category(ies): Character Classification / Conversion

Description: Convert a character to lower-case if it was upper-case (including the international alphabetic characters).

Inputs: *c* — Character to convert.

Outputs: Character converted to lower-case if it was upper-case otherwise the same character is returned.

Assumptions: Access_AMS_Global_Variables is defined.

Side Effects: None

Availability: AMS 2.00 or above.

TI-89 / TI-92 Plus Differences: None

See Also: **isupper**

Example: See **isalpha**.

toupper

Declaration: BYTE **toupper** (BYTE *c*)

Category(ies): Character Classification / Conversion

Description: Convert a character to upper-case if it was lower-case (including the international alphabetic characters).

Inputs: *c* — Character to convert.

Outputs: Character converted to upper-case if it was lower-case otherwise the same character is returned.

Assumptions: Access_AMS_Global_Variables is defined.

Side Effects: None

Availability: AMS 2.00 or above.

TI-89 / TI-92 Plus Differences: None

See Also: **islower**

Example: See **isalnum**.

DlgNotice

Declaration: WORD **DlgNotice** (const char * *Title*, const char * *Message*)

Category(ies): Dialog

Description: Issue a dialog with a given *Title*, a word-wrapped *Message*, and a single OK button. The *Message* string may contain newline constants. The dialog box will be sized to fit the screen with a predefined width for the TI-89 and the TI-92 Plus.

Inputs:

Title	—	String pointer for title of dialog box (no title if NULL).
Message	—	String pointer message to be word wrapped in dialog box.

Outputs:

KB_ENTER	—	User pressed ENTER to close dialog box.
KB_ESC	—	User pressed ESC to close dialog box.

Assumptions: **Access_AMS_Global_Variables** is defined. If there is not enough memory for the dialog box, a low memory version will be used (no word wrapping); so this dialog will always succeed.

Side Effects: May cause the heap to be compressed.

Availability: All versions of the TI-89 / TI-92 Plus. However, on AMS 2.04 and higher, word wrap also occurs on commas and spaces.

TI-89 / TI-92 Plus Differences: None

See Also: **DlgMessage**

Example:

```
DlgNotice("ERROR IN SETUP", "Invalid value" );
```

ENDFINAL

Declaration: Not applicable.

Category(ies): Error Handling

Description: The **ENDFINAL** macro marks the end of a TRY . . . FINALLY . . . ENDFINAL block. See chapter **9. Error Handling** for a complete discussion of throwing and catching exceptions.

Inputs: Not applicable.

Outputs: Not applicable.

Assumptions: Not applicable.

Side Effects: Not applicable.

Availability: All versions of the TI‑89 / TI‑92 Plus.

TI‑89 / TI‑92 Plus Differences: None

See Also: **ER_throw, ER_throwFrame, ER_throwVar, ENDTRY, FINALLY, ONERR, TRY**

Example:

```
TRY
   /* code which can throw an error */
FINALLY
   /* clean-up code */
ENDFINAL
```

ENDTRY

Declaration: Not applicable.

Category(ies): Error Handling

Description: The **ENDTRY** macro marks the end of a TRY . . . ONERR . . . ENDTRY block. See chapter **9. Error Handling** for a complete discussion of throwing and catching exceptions.

Inputs: Not applicable.

Outputs: Not applicable.

Assumptions: Not applicable.

Side Effects: Not applicable.

Availability: All versions of the TI‑89 / TI‑92 Plus.

TI‑89 / TI‑92 Plus Differences: None

See Also: **ER_throw, ER_throwFrame, ER_throwVar, ENDFINAL, FINALLY, ONERR, TRY**

Example:

```
TRY
   /* code which can throw an error */
ONERR
   /* execution continues here only if an error was thrown above */
ENDTRY
```

ER_throw

Declaration: **ER_throw** (int *errorCode*)

Category(ies): Error Handling

Description: This macro throws an exception to be caught in the ONERR or FINALLY section of an enclosing TRY block. See chapter **9. Error Handling** for a complete discussion of throwing and catching exceptions.

> **Note:** This is a macro version of **ER_throwVar**, but *errorCode* must be an integer constant.

ER_throw compiles to shorter code than **ER_throwVar** — about two bytes shorter per call.

Inputs: *errorCode* — Error number signifying cause of exception.

Outputs: None. This macro never returns — execution resumes at the ONERR or FINALLY section of the enclosing TRY block.

Assumptions: None

Side Effects: None

Availability: All versions of the TI-89 / TI-92 Plus.

TI-89 / TI-92 Plus Differences: None

See Also: **ER_throwFrame, ER_throwVar, FINALLY, ONERR, TRY**

Example:

```
TRY
   HANDLE h;
   h = HeapAlloc(A_BIG_BUFFER);
   if (h == H_NULL)
      ER_throw(ER_MEMORY);
   .
   .
   .
   HeapFree(h)
ONERR
   /* Do something about low memory condition */
   .
   .
   .
ENDTRY
```

FINALLY

Declaration: Not applicable.

Category(ies): Error Handling

Description: TRY . . . FINALLY . . . ENDFINAL allows a program to execute a section of clean-up code whether or not an exception is thrown in the TRY section. The **FINALLY** macro marks the beginning of the clean-up section. See chapter **9. Error Handling** for a complete discussion of throwing and catching exceptions.

Inputs: Not applicable.

Outputs: Not applicable.

Assumptions: Not applicable.

Side Effects: Not applicable.

Availability: All versions of the TI‑89 / TI‑92 Plus.

TI‑89 / TI‑92 Plus Differences: None

See Also: **ER_throw, ER_throwFrame, ER_throwVar, ENDFINAL, ENDTRY, ONERR, TRY**

Example:

```
TRY
   /* code which can throw an error */
FINALLY
   /* Execution continues here. If an error was thrown in the TRY section, the error
      is re-thrown after executing this section. */
ENDFINAL
```

ONERR

Declaration: Not applicable.

Category(ies): Error Handling

Description: The ONERR macro marks the beginning of the section of code which catches errors thrown in the TRY section of a TRY . . . ONERR . . . ENDTRY block. See chapter **9. Error Handling** for a complete discussion of throwing and catching exceptions.

Inputs: Not applicable.

Outputs: Not applicable.

Assumptions: Not applicable.

Side Effects: Not applicable.

Availability: All versions of the TI-89 / TI-92 Plus.

TI-89 / TI-92 Plus Differences: None

See Also: **ER_throw, ER_throwFrame, ER_throwVar, ENDFINAL, ENDTRY, FINALLY, TRY**

Example:

```
TRY
   /* code which can throw an error */
ONERR
   /* execution continues here only if an error was thrown above */
ENDTRY
```

TRY

Declaration: Not applicable.

Category(ies): Error Handling

Description: The TRY macro marks the beginning of an error handling block. There are two kinds of error handling blocks: TRY . . . ONERR . . . ENDTRY and TRY . . . FINALLY . . . ENDFINAL. See chapter **9. Error Handling** for a complete discussion of throwing and catching exceptions.

Inputs: Not applicable.

Outputs: Not applicable.

Assumptions: Not applicable.

Side Effects: Not applicable.

Availability: All versions of the TI‑89 / TI‑92 Plus.

TI‑89 / TI‑92 Plus Differences: None

See Also: **ER_throw, ER_throwFrame, ER_throwVar, ENDFINAL, ENDTRY, FINALLY, ONERR**

Example:

```
TRY
   /* code which can throw an error */
ONERR
   /* execution continues here only if an error was thrown above */
ENDTRY
```

Access_AMS_Global_Variables

Declaration: **Access_AMS_Global_Variables**

Category(ies): Operating System

Description: Defines linkage to many OS data structures, flags, and global variables.

Global variable and macro descriptions in **Appendix B** and **Appendix C** state in their assumptions if **Access_AMS_Global_Variables** must be defined. This means that macro **Access_AMS_Global_Variables** must be placed in the declaration section of the block containing the global variable or macro reference.

Any global OS variables may be accessed or modified within the scope of a block containing a **Access_AMS_Global_Variables** declaration.

> **Note:** Please exercise caution and restraint when modifying OS variables. Most OS variables are modified by other routines in the operating system as a side-effect of their actions. Modifying OS variables directly from your program could seriously destabilize the calculator.

Inputs: None

Outputs: None

Assumptions: None

Side Effects: None

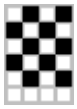
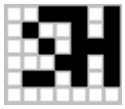
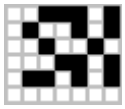
Availability: All versions of the TI-89 / TI-92 Plus.

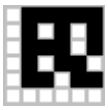
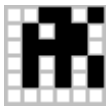
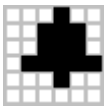
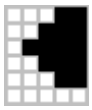
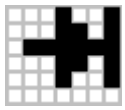
TI-89 / TI-92 Plus Differences: None

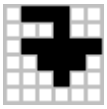
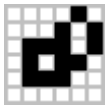
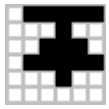
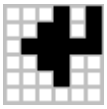
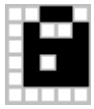
See Also: Not applicable

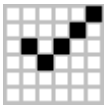
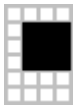
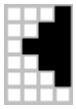
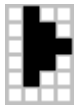
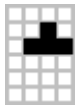
Example: See example for **gr_flags** in **Appendix B: Global Variables — Graphing**.

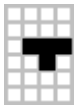
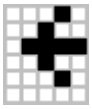
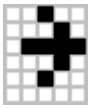
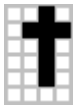
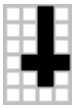
Appendix D:
TI-89 / TI-92 Plus “Small” Character Font

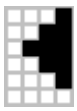
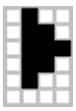
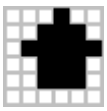
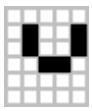
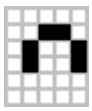
0x00 NUL	0x01 SOH	0x02 STX	0x03 ETX	0x04 EOT
				

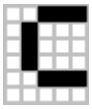
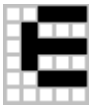
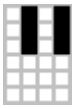
0x05 ENQ	0x06 ACK	0x07 BEL	0x08 BS	0x09 HT
				

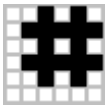
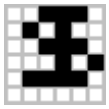
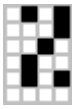
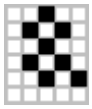
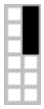
0x0A LF	0x0B UPLOAD	0x0C FF	0x0D CR	0x0E LOCKED
				

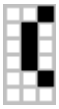
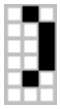
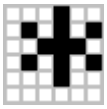
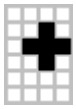
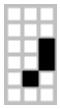
0x0F CHECK	0x10 BLOCK	0x11 ALT_ARROW_LEFT	0x12 ALT_ARROW_RIGHT	0x13 ALT_ARROW_UP
				

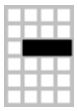
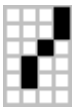
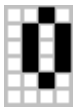
0x14 ALT_ARROW_DOWN	0x15 ARROW_LEFT	0x16 ARROW_RIGHT	0x17 ARROW_UP	0x18 ARROW_DOWN
				

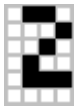
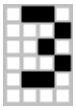
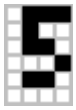
0x19 MORE_LEFT	0x1A MORE_RIGHT	0x1B SHIFT_INDIC	0x1C UNION	0x1D INTERSECTION
				

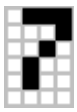
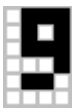
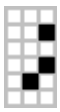
0x1E SUBSET	0x1F ELEMENT	0x20 SPACE	0x21 EXCLAM	0x22 QUOTE
				

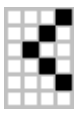
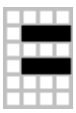
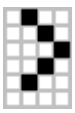
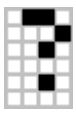
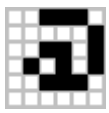
0x23 POUND	0x24 DOLLAR	0x25 PERCENT	0x26 AMPERSAND	0x27 APOSTROPHE
				

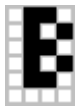
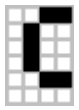
0x28 LPAREN	0x29 RPAREN	0x2A ASTERISK	0x2B PLUS	0x2C COMMA
				

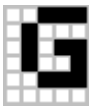
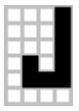
0x2D MINUS	0x2E PERIOD	0x2F SLASH	0x30 0	0x31 1
				

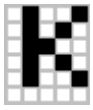
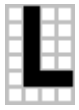
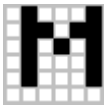
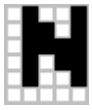
0x32 2	0x33 3	0x34 4	0x35 5	0x36 6
				

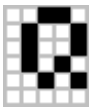
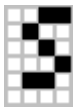
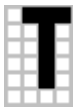
0x37 7	0x38 8	0x39 9	0x3A COLON	0x3B SEMICOLON
				

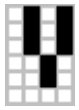
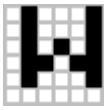
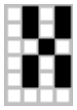
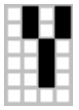
0x3C LT	0x3D EQ	0x3E GT	0x3F QUESTION	0x40 ATSIGN
				

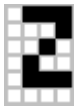
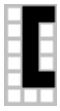
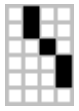
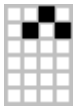
0x41 CAP_A	0x42 CAP_B	0x43 CAP_C	0x44 CAP_D	0x45 CAP_E
				

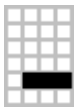
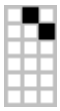
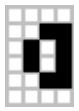
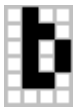
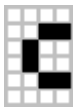
0x46 CAP_F	0x47 CAP_G	0x48 CAP_H	0x49 CAP_I	0x4A CAP_J
				

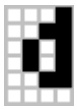
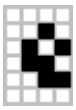
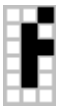
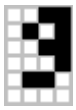
0x4B CAP_K	0x4C CAP_L	0x4D CAP_M	0x4E CAP_N	0x4F CAP_O
				

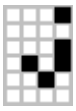
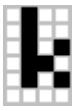
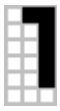
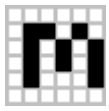
0x50 CAP_P	0x51 CAP_Q	0x52 CAP_R	0x53 CAP_S	0x54 CAP_T
				

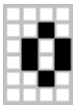
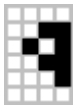
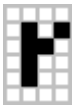
0x55 CAP_U	0x56 CAP_V	0x57 CAP_W	0x58 CAP_X	0x59 CAP_Y
				

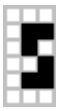
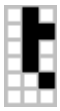
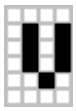
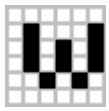
0x5A CAP_Z	0x5B LBRACKET	0x5C BACKSLASH	0x5D RBRACKET	0x5E CARET
				

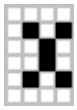
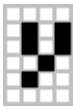
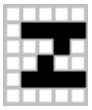
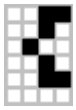
0x5F UNDERSCORE	0x60 BACKQUOTE	0x61 A	0x62 B	0x63 C
				

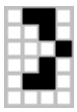
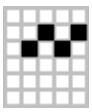
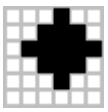
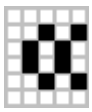
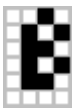
0x64 D	0x65 E	0x66 F	0x67 G	0x68 H
				

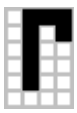
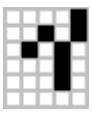
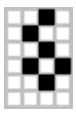
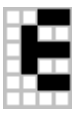
0x69 I	0x6A J	0x6B K	0x6C L	0x6D M
				

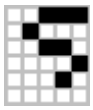
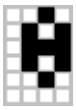
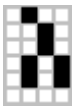
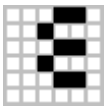
0x6E N	0x6F O	0x70 P	0x71 Q	0x72 R
				

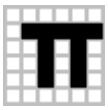
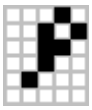
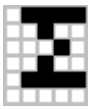
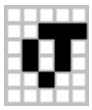
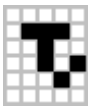
0x73 S	0x74 T	0x75 U	0x76 V	0x77 W
				

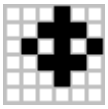
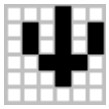
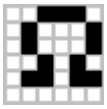
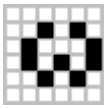
0x78 X	0x79 Y	0x7A Z	0x7B LBRACE	0x7C BAR
				

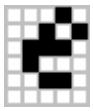
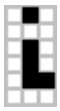
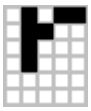
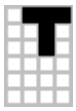
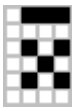
0x7D RBRACE	0x7E TILDE	0x7F OPTION	0x80 ALPHA	0x81 BETA
				

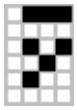
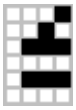
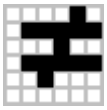
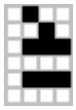
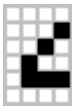
0x82 CAP_GAMMA	0x83 GAMMA	0x84 CAP_DELTA	0x85 DELTA	0x86 EPSILON
				

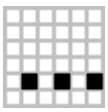
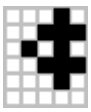
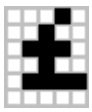
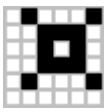
0x87 ZETA	0x88 THETA	0x89 LAMBDA	0x8A XI	0x8B CAP_PI
				

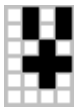
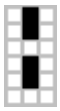
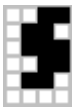
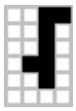
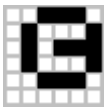
0x8C PI	0x8D RHO	0x8E CAP_SIGMA	0x8F SIGMA	0x90 TAU
				

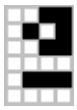
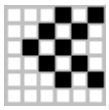
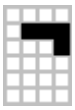
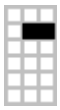
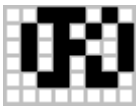
0x91 PHI	0x92 PSI	0x93 CAP_OMEGA	0x94 OMEGA	0x95 EXPONENT
				

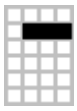
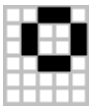
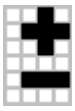
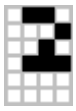
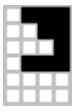
0x96 CONST_E	0x97 IMAG	0x98 RADIANS	0x99 TRANSPOSE	0x9A X_MEAN
				

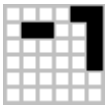
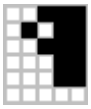
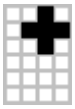
0x9B Y_MEAN	0x9C LE	0x9D NE	0x9E GE	0x9F ANGLE
				

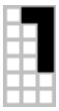
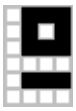
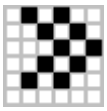
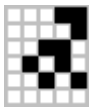
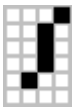
0xA0 ELLIPSIS	0xA1 EXCLAM_DOWN	0xA2 CENT	0xA3 POUND_STIRLING	0xA4 SUNBURST
				

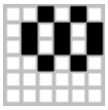
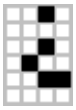
0xA5 YEN	0xA6 SPLIT_BAR	0xA7 SECTION	0xA8 ROOT	0xA9 COPYRIGHT
				

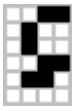
0xAA SUPER_A	0xAB FRENCH_LBRACKET	0xAC LOGICAL_NOT	0xAD NEGATIVE	0xAE REGISTERED
				

0xAF SUPER_MINUS	0xB0 DEGREES	0xB1 PLUS_OR_MINUS	0xB2 SQUARED	0xB3 CUBED
				

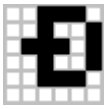
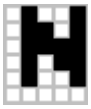
0xB4 INVERSE	0xB5 MU	0xB6 PARAGRAPH	0xB7 MULTIPLY	0xB8 SUPER_PLUS
				

0xB9 SUPER_1	0xBA SUPER_0	0xBB FRENCH_RBRACKET	0xBC DIFF	0xBD INTEGRAL
				

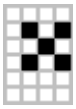
0xBE INFINITY	0xBF QUES_DOWN	0xC0 CAP_A_GRAVE	0xC1 CAP_A_ACUTE	0xC2 CAP_A_CARET
				

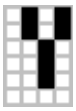
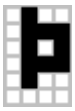
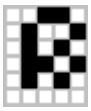
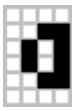
0xC3 CAP_A_TILDE	0xC4 CAP_A_UMLAUT	0xC5 CAP_A_RING	0xC6 CAP_AE	0xC7 CAP_C_CEDILLA
				

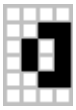
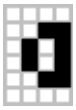
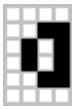
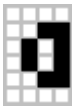
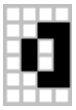
0xC8 CAP_E_GRAVE	0xC9 CAP_E_ACUTE	0xCA CAP_E_CARET	0xCB CAP_E_UMLAUT	0xCC CAP_I_GRAVE
				

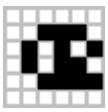
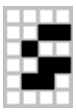
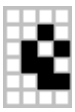
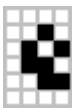
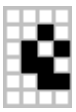
0xCD CAP_I_ACUTE	0xCE CAP_I_CARET	0xCF CAP_I_UMLAUT	0xD0 CAP_D_BAR	0xD1 CAP_N_TILDE
				

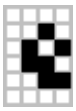
0xD2 CAP_O_GRAVE	0xD3 CAP_O_ACUTE	0xD4 CAP_O_CARET	0xD5 CAP_O_TILDE	0xD6 CAP_O_UMLAUT
				

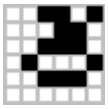
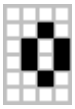
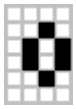
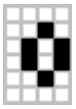
0xD7 TIMES	0xD8 CAP_O_SLASH	0xD9 CAP_U_GRAVE	0xDA CAP_U_ACUTE	0xDB CAP_U_CARET
				

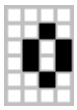
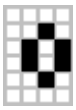
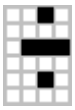
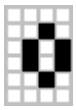
0xDC CAP_U_UMLAUT	0xDD CAP_Y_ACUTE	0xDE CAP_THORN	0xDF LONG_S	0xE0 A_GRAVE
				

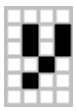
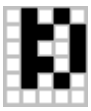
0xE1 A_ACUTE	0xE2 A_CARET	0xE3 A_TILDE	0xE4 A_UMLAUT	0xE5 A_RING
				

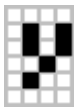
0xE6 AE	0xE7 C_CEDILLA	0xE8 E_GRAVE	0xE9 E_ACUTE	0xEA E_CARET
				

0xEB E_UMLAUT	0xEC I_GRAVE	0xED I_ACUTE	0xEE I_CARET	0xEF I_UMLAUT
				

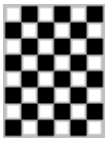
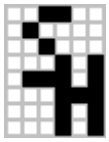
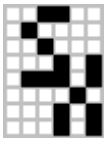
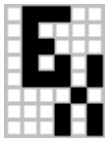
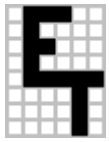
0xF0 D_BAR	0xF1 N_TILDE	0xF2 O_GRAVE	0xF3 O_ACUTE	0xF4 O_CARET
				

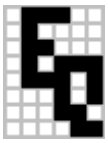
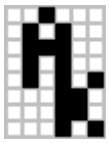
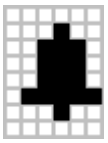
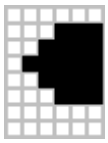
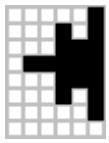
0xF5 O_TILDE	0xF6 O_UMLAUT	0xF7 DIVIDE	0xF8 O_SLASH	0xF9 U_GRAVE
				

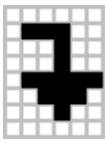
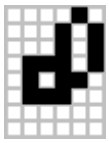
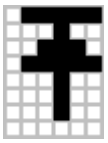
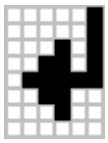
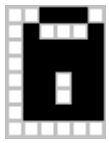
0xFA U_ACUTE	0xFB U_CARET	0xFC U_UMLAUT	0xFD Y_ACUTE	0xFE THORN
				

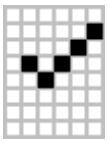
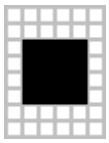
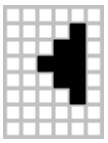
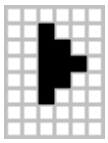
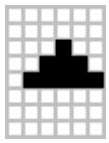
0xFF Y_UMLAUT				
				

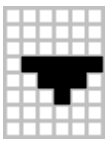
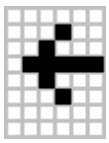
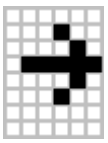
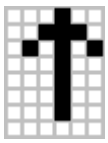
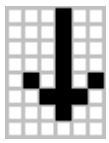
Appendix E:
TI-89 / TI-92 Plus “Large” Character Font

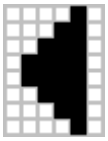
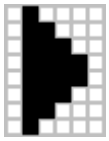
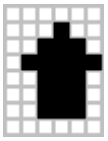
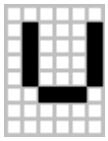
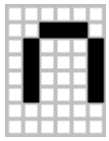
0x00 NUL	0x01 SOH	0x02 STX	0x03 ETX	0x04 EOT
				

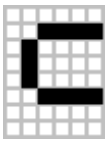
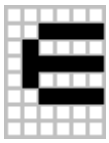
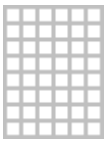
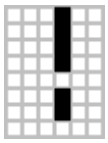
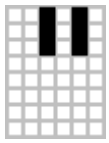
0x05 ENQ	0x06 ACK	0x07 BEL	0x08 BS	0x09 HT
				

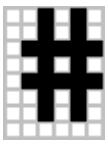
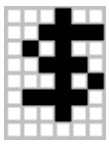
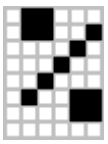
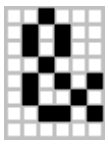
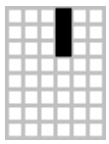
0x0A LF	0x0B UPLOAD	0x0C FF	0x0D CR	0x0E LOCKED
				

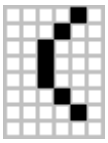
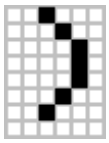
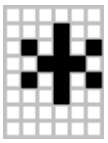
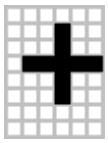
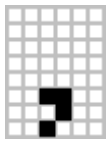
0x0F CHECK	0x10 BLOCK	0x11 ALT_ARROW_LEFT	0x12 ALT_ARROW_RIGHT	0x13 ALT_ARROW_UP
				

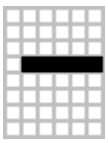
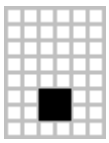
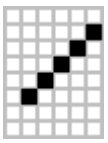
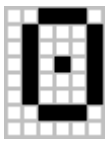
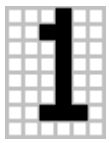
0x14 ALT_ARROW_DOWN	0x15 ARROW_LEFT	0x16 ARROW_RIGHT	0x17 ARROW_UP	0x18 ARROW_DOWN
				

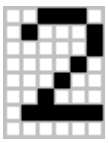
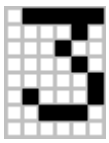
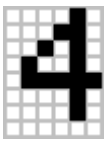
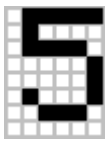
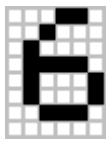
0x19 MORE_LEFT	0x1A MORE_RIGHT	0x1B SHIFT_INDIC	0x1C UNION	0x1D INTERSECTION
				

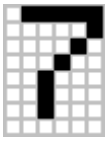
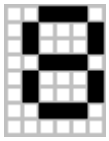
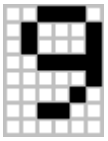
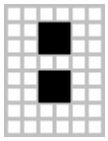
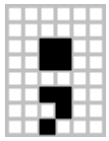
0x1E SUBSET	0x1F ELEMENT	0x20 SPACE	0x21 EXCLAM	0x22 QUOTE
				

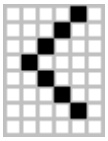
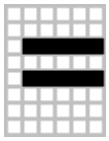
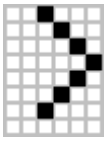
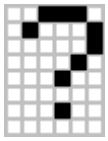
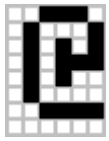
0x23 POUND	0x24 DOLLAR	0x25 PERCENT	0x26 AMPERSAND	0x27 APOSTROPHE
				

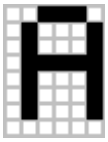
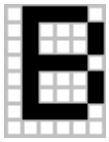
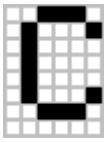
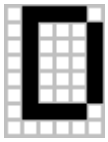
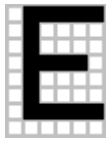
0x28 LPAREN	0x29 RPAREN	0x2A ASTERISK	0x2B PLUS	0x2C COMMA
				

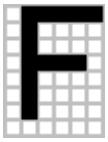
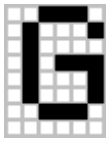
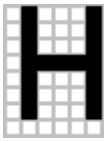
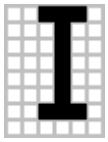
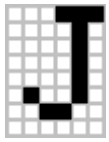
0x2D MINUS	0x2E PERIOD	0x2F SLASH	0x30 0	0x31 1
				

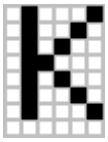
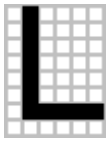
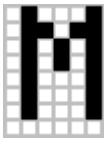
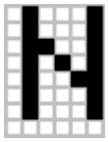
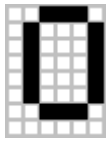
0x32 2	0x33 3	0x34 4	0x35 5	0x36 6
				

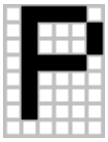
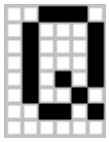
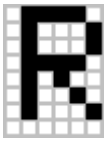
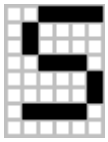
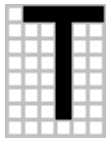
0x37 7	0x38 8	0x39 9	0x3A COLON	0x3B SEMICOLON
				

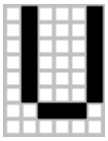
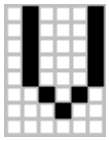
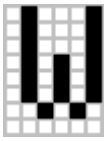
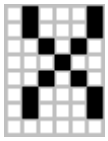
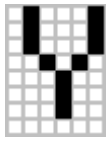
0x3C LT	0x3D EQ	0x3E GT	0x3F QUESTION	0x40 ATSIGN
				

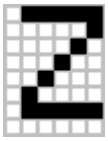
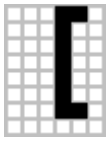
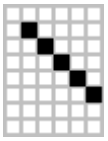
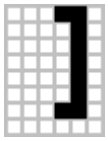
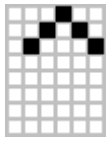
0x41 CAP_A	0x42 CAP_B	0x43 CAP_C	0x44 CAP_D	0x45 CAP_E
				

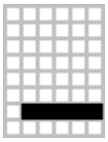
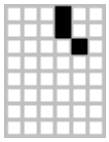
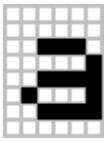
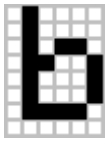
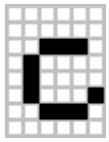
0x46 CAP_F	0x47 CAP_G	0x48 CAP_H	0x49 CAP_I	0x4A CAP_J
				

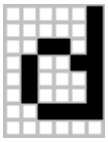
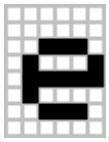
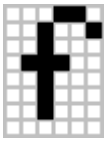
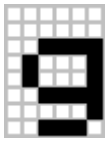
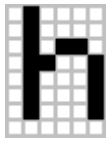
0x4B CAP_K	0x4C CAP_L	0x4D CAP_M	0x4E CAP_N	0x4F CAP_O
				

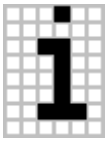
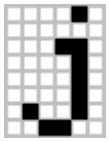
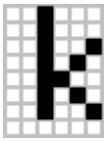
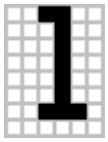
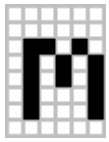
0x50 CAP_P	0x51 CAP_Q	0x52 CAP_R	0x53 CAP_S	0x54 CAP_T
				

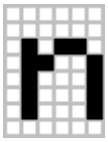
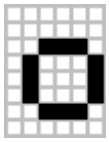
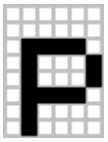
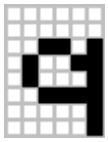
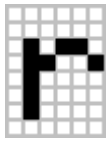
0x55 CAP_U	0x56 CAP_V	0x57 CAP_W	0x58 CAP_X	0x59 CAP_Y
				

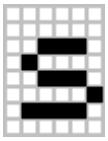
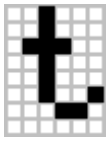
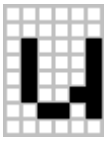
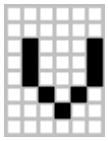
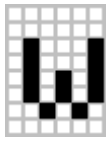
0x5A CAP_Z	0x5B LBRACKET	0x5C BACKSLASH	0x5D RBRACKET	0x5E CARET
				

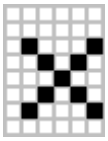
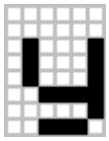
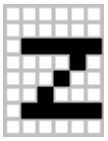
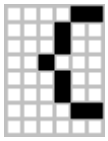
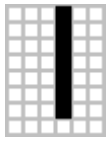
0x5F UNDERSCORE	0x60 BACKQUOTE	0x61 A	0x62 B	0x63 C
				

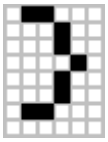
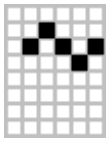
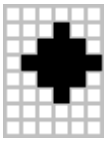
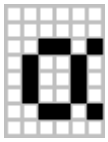
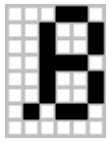
0x64 D	0x65 E	0x66 F	0x67 G	0x68 H
				

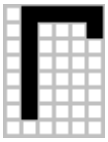
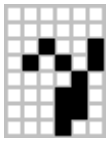
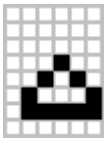
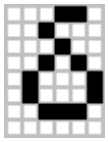
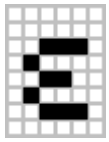
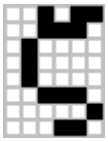
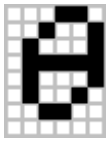
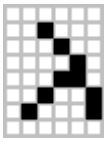
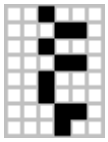
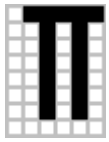
0x69 I	0x6A J	0x6B K	0x6C L	0x6D M
				

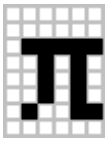
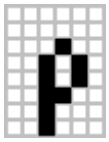
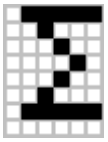
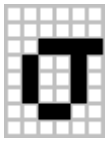
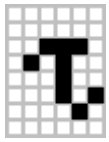
0x6E N	0x6F O	0x70 P	0x71 Q	0x72 R
				

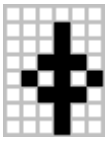
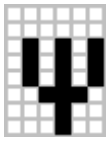
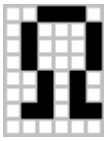
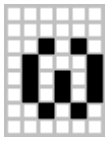
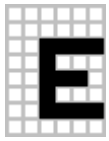
0x73 S	0x74 T	0x75 U	0x76 V	0x77 W
				

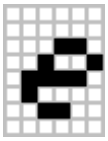
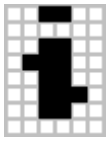
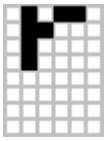
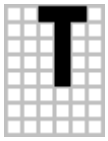
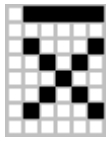
0x78 X	0x79 Y	0x7A Z	0x7B LBRACE	0x7C BAR
				

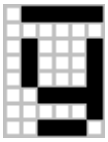
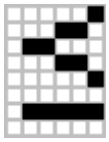
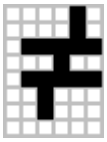
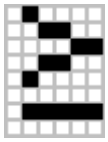
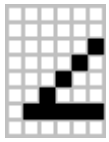
0x7D RBRACE	0x7E TILDE	0x7F OPTION	0x80 ALPHA	0x81 BETA
				

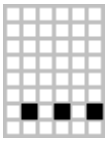
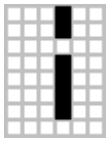
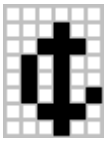
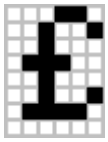
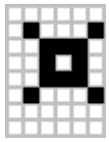
0x82 CAP_GAMMA	0x83 GAMMA	0x84 CAP_DELTA	0x85 DELTA	0x86 EPSILON
				
0x87 ZETA	0x88 THETA	0x89 LAMBDA	0x8A XI	0x8B CAP_PI
				

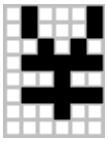
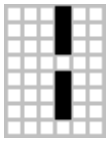
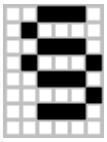
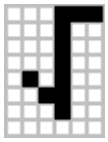
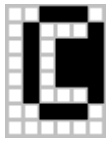
0x8C PI	0x8D RHO	0x8E CAP_SIGMA	0x8F SIGMA	0x90 TAU
				

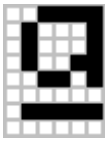
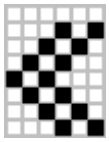
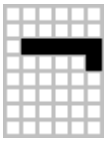
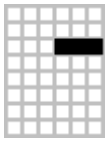
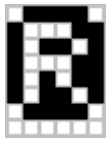
0x91 PHI	0x92 PSI	0x93 CAP_OMEGA	0x94 OMEGA	0x95 EXPONENT
				

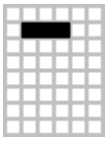
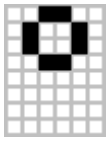
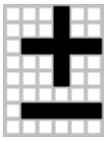
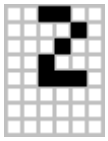
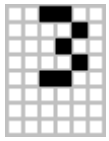
0x96 CONST_E	0x97 IMAG	0x98 RADIANS	0x99 TRANSPOSE	0x9A X_MEAN
				

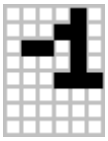
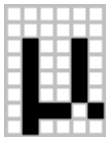
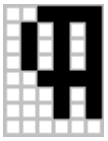
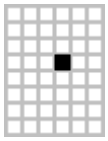
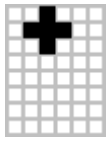
0x9B Y_MEAN	0x9C LE	0x9D NE	0x9E GE	0x9F ANGLE
				

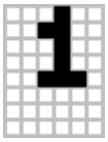
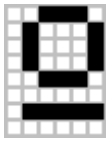
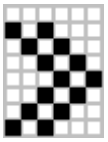
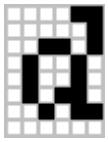
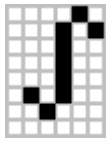
0xA0 ELLIPSIS	0xA1 EXCLAM_DOWN	0xA2 CENT	0xA3 POUND_STIRLING	0xA4 SUNBURST
				

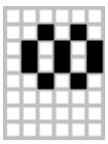
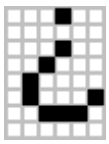
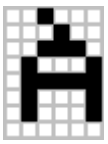
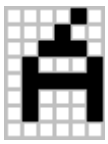
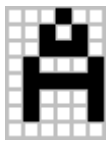
0xA5 YEN	0xA6 SPLIT_BAR	0xA7 SECTION	0xA8 ROOT	0xA9 COPYRIGHT
				

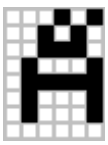
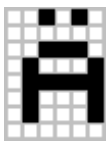
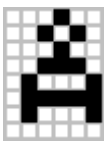
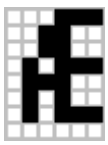
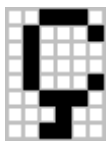
0xAA SUPER_A	0xAB FRENCH_LBRACKET	0xAC LOGICAL_NOT	0xAD NEGATIVE	0xAE REGISTERED
				

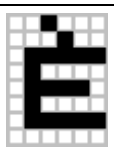
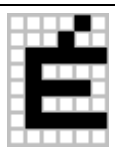
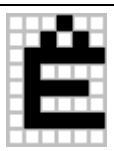
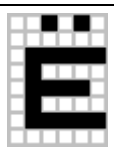
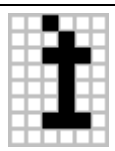
0xAF SUPER_MINUS	0xB0 DEGREES	0xB1 PLUS_OR_MINUS	0xB2 SQUARED	0xB3 CUBED
				

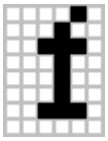
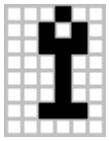
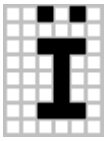
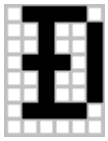
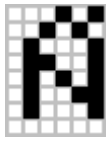
0xB4 INVERSE	0xB5 MU	0xB6 PARAGRAPH	0xB7 MULTIPLY	0xB8 SUPER_PLUS
				

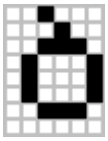
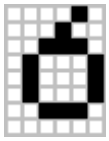
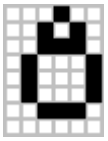
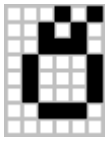
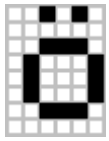
0xB9 SUPER_1	0xBA SUPER_O	0xBB FRENCH_RBRACKET	0xBC DIFF	0xBD INTEGRAL
				

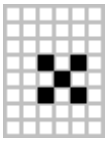
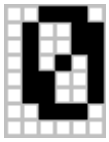
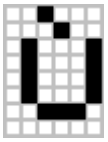
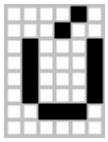
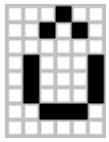
0xBE INFINITY	0xBF QUES_DOWN	0xC0 CAP_A_GRAVE	0xC1 CAP_A_ACUTE	0xC2 CAP_A_CARET
				

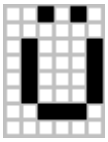
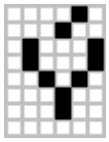
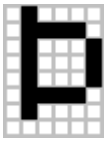
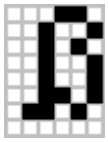
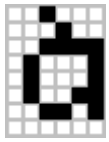
0xC3 CAP_A_TILDE	0xC4 CAP_A_UMLAUT	0xC5 CAP_A_RING	0xC6 CAP_AE	0xC7 CAP_C_CEDILLA
				

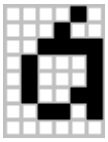
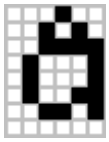
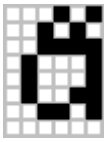
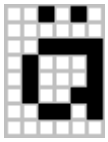
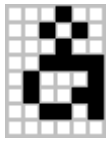
0xC8 CAP_E_GRAVE	0xC9 CAP_E_ACUTE	0xCA CAP_E_CARET	0xCB CAP_E_UMLAUT	0xCC CAP_I_GRAVE
				

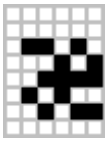
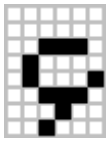
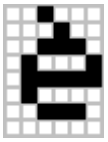
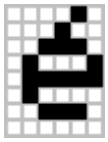
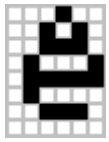
0xCD CAP_I_ACUTE	0xCE CAP_I_CARET	0xCF CAP_I_UMLAUT	0xD0 CAP_D_BAR	0xD1 CAP_N_TILDE
				

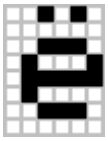
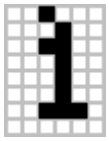
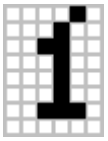
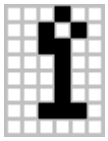
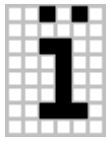
0xD2 CAP_O_GRAVE	0xD3 CAP_O_ACUTE	0xD4 CAP_O_CARET	0xD5 CAP_O_TILDE	0xD6 CAP_O_UMLAUT
				

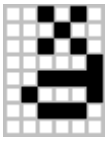
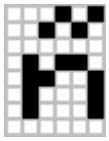
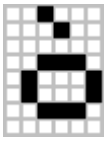
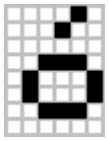
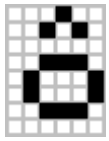
0xD7 TIMES	0xD8 CAP_O_SLASH	0xD9 CAP_U_GRAVE	0xDA CAP_U_ACUTE	0xDB CAP_U_CARET
				

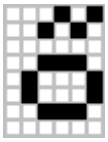
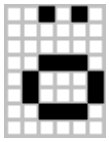
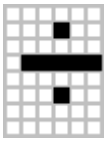
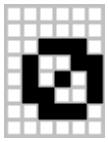
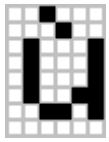
0xDC CAP_U_UMLAUT	0xDD CAP_Y_ACUTE	0xDE CAP_THORN	0xDF LONG_S	0xE0 A_GRAVE
				

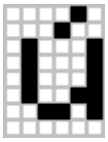
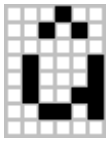
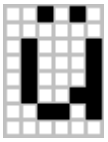
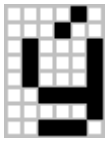
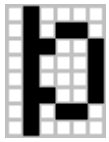
0xE1 A_ACUTE	0xE2 A_CARET	0xE3 A_TILDE	0xE4 A_UMLAUT	0xE5 A_RING
				

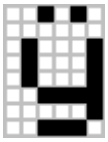
0xE6 AE	0xE7 C_CEDILLA	0xE8 E_GRAVE	0xE9 E_ACUTE	0xEA E_CARET
				

0xEB E_UMLAUT	0xEC I_GRAVE	0xED I_ACUTE	0xEE I_CARET	0xEF I_UMLAUT
				

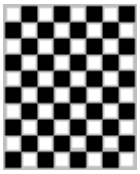
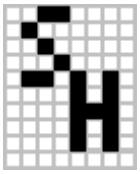
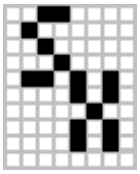
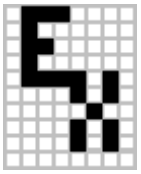
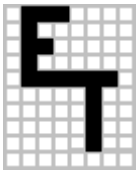
0xF0 D_BAR	0xF1 N_TILDE	0xF2 O_GRAVE	0xF3 O_ACUTE	0xF4 O_CARET
				

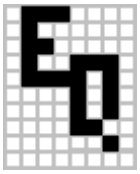
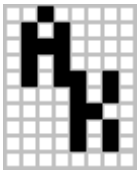
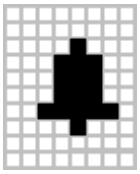
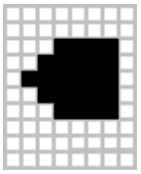
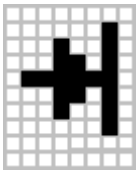
0xF5 O_TILDE	0xF6 O_UMLAUT	0xF7 DIVIDE	0xF8 O_SLASH	0xF9 U_GRAVE
				

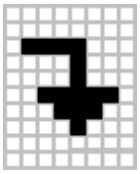
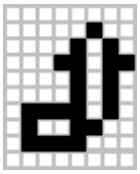
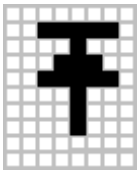
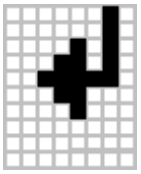
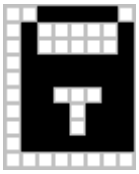
0xFA U_ACUTE	0xFB U_CARET	0xFC U_UMLAUT	0xFD Y_ACUTE	0xFE THORN
				

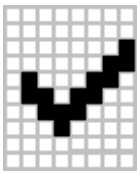
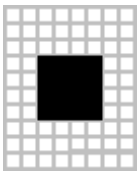
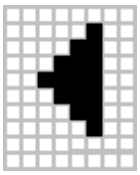
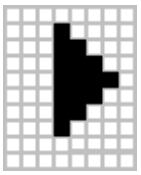
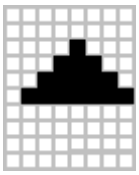
0xFF Y_UMLAUT				
				

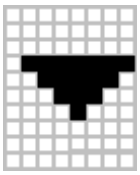
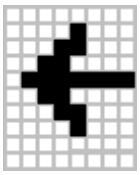
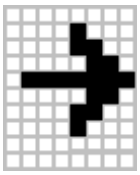
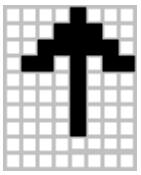
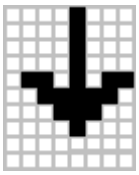
Appendix F:
TI-89 / TI-92 Plus “Huge” Character Font

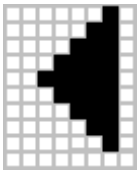
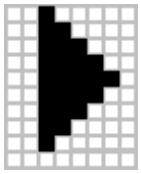
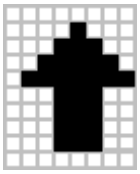
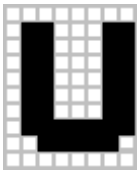
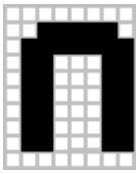
0x00 NUL	0x01 SOH	0x02 STX	0x03 ETX	0x04 EOT
				

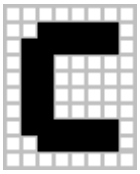
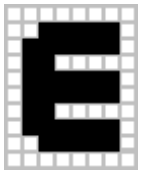
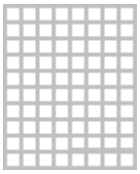
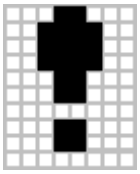
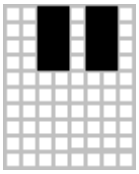
0x05 ENQ	0x06 ACK	0x07 BEL	0x08 BS	0x09 HT
				

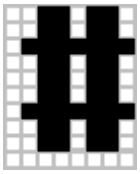
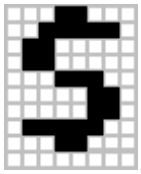
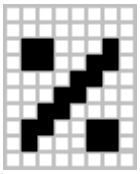
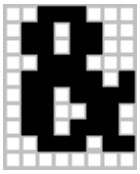
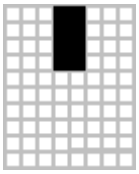
0x0A LF	0x0B UPLOAD	0x0C FF	0x0D CR	0x0E LOCKED
				

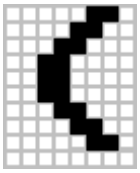
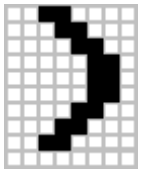
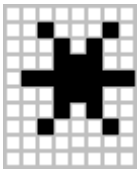
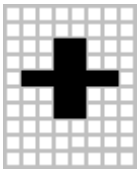
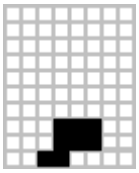
0x0F CHECK	0x10 BLOCK	0x11 ALT_ARROW_LEFT	0x12 ALT_ARROW_RIGHT	0x13 ALT_ARROW_UP
				

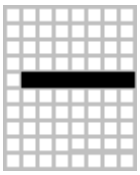
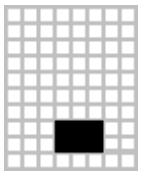
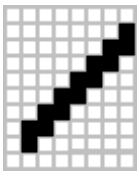
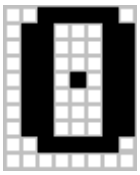
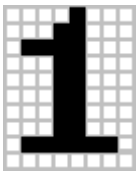
0x14 ALT_ARROW_DOWN	0x15 ARROW_LEFT	0x16 ARROW_RIGHT	0x17 ARROW_UP	0x18 ARROW_DOWN
				

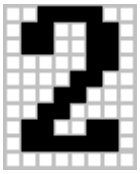
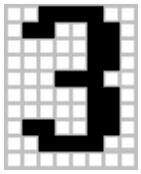
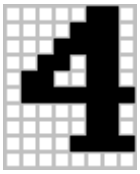
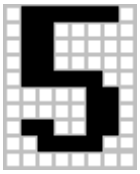
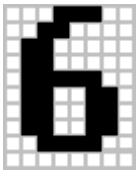
0x19 MORE_LEFT	0x1A MORE_RIGHT	0x1B SHIFT_INDIC	0x1C UNION	0x1D INTERSECTION
				

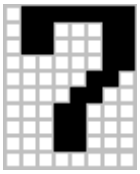
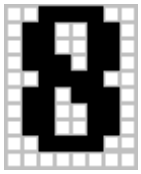
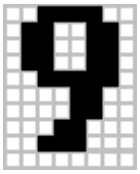
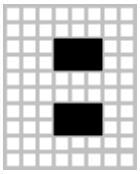
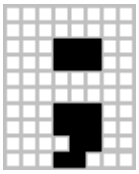
0x1E SUBSET	0x1F ELEMENT	0x20 SPACE	0x21 EXCLAM	0x22 QUOTE
				

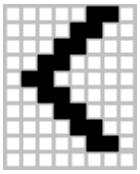
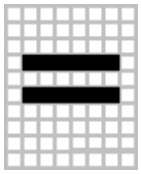
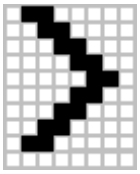
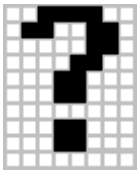
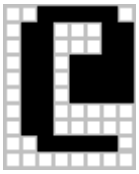
0x23 POUND	0x24 DOLLAR	0x25 PERCENT	0x26 AMPERSAND	0x27 APOSTROPHE
				

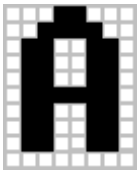
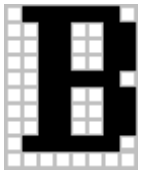
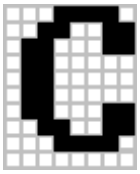
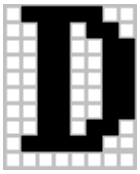
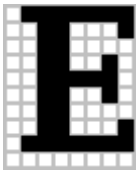
0x28 LPAREN	0x29 RPAREN	0x2A ASTERISK	0x2B PLUS	0x2C COMMA
				

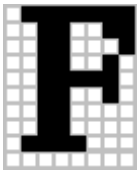
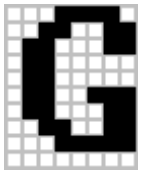
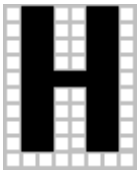
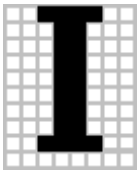
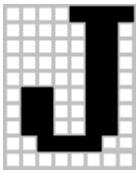
0x2D MINUS	0x2E PERIOD	0x2F SLASH	0x30 0	0x31 1
				

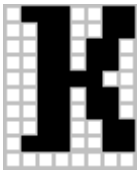
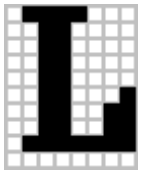
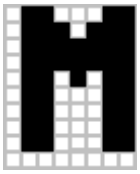
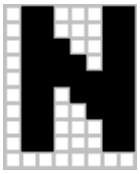
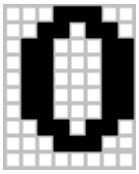
0x32 2	0x33 3	0x34 4	0x35 5	0x36 6
				

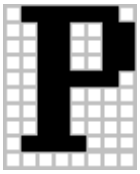
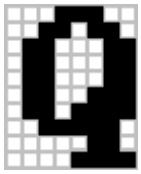
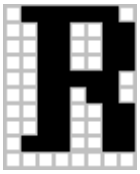
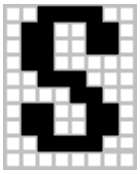
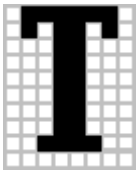
0x37 7	0x38 8	0x39 9	0x3A COLON	0x3B SEMICOLON
				

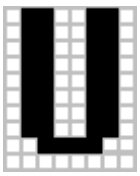
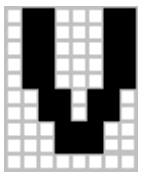
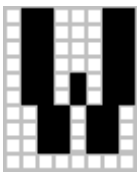
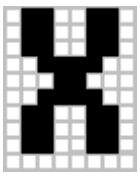
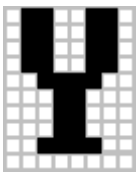
0x3C LT	0x3D EQ	0x3E GT	0x3F QUESTION	0x40 ATSIGN
				

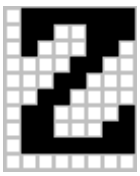
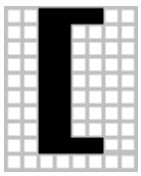
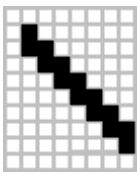
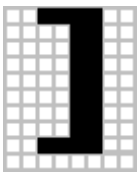
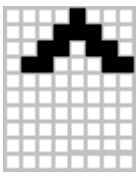
0x41 CAP_A	0x42 CAP_B	0x43 CAP_C	0x44 CAP_D	0x45 CAP_E
				

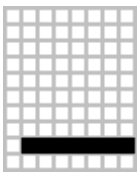
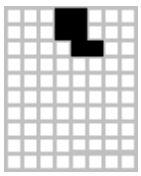
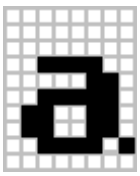
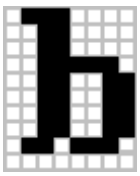
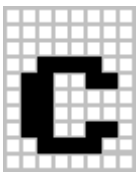
0x46 CAP_F	0x47 CAP_G	0x48 CAP_H	0x49 CAP_I	0x4A CAP_J
				

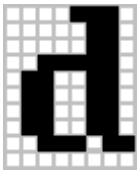
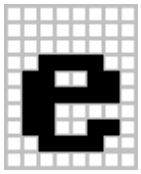
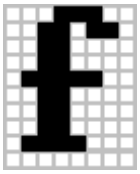
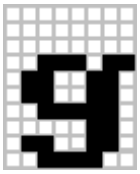
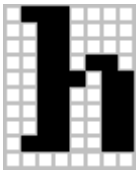
0x4B CAP_K	0x4C CAP_L	0x4D CAP_M	0x4E CAP_N	0x4F CAP_O
				

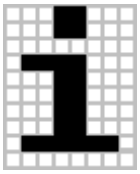
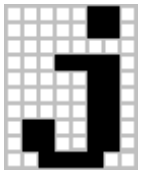
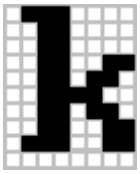
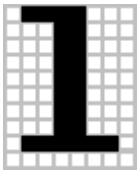
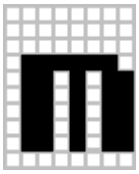
0x50 CAP_P	0x51 CAP_Q	0x52 CAP_R	0x53 CAP_S	0x54 CAP_T
				

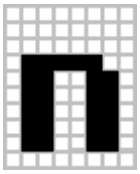
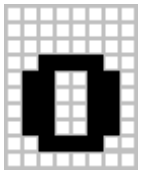
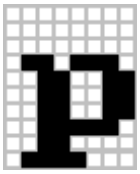
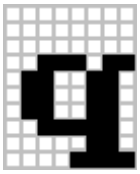
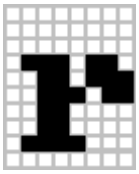
0x55 CAP_U	0x56 CAP_V	0x57 CAP_W	0x58 CAP_X	0x59 CAP_Y
				

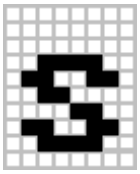
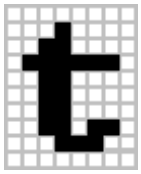
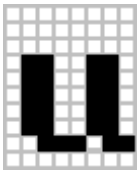
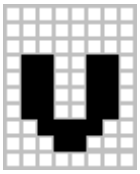
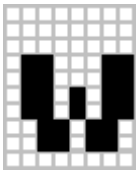
0x5A CAP_Z	0x5B LBRACKET	0x5C BACKSLASH	0x5D RBRACKET	0x5E CARET
				

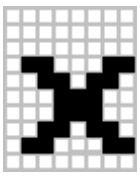
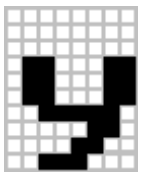
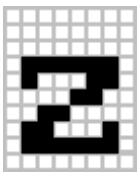
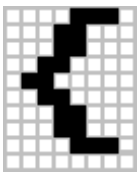
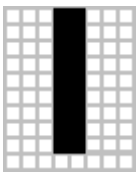
0x5F UNDERSCORE	0x60 BACKQUOTE	0x61 A	0x62 B	0x63 C
				

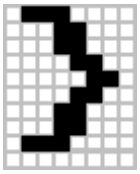
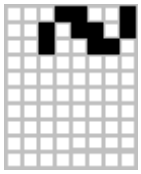
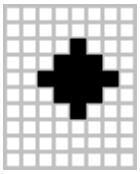
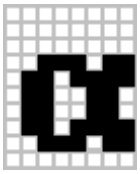
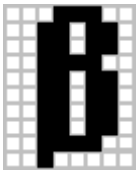
0x64 D	0x65 E	0x66 F	0x67 G	0x68 H
				

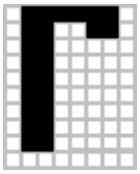
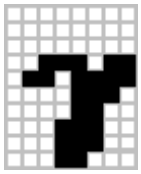
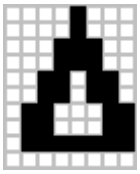
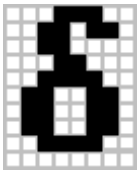
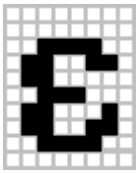
0x69 I	0x6A J	0x6B K	0x6C L	0x6D M
				

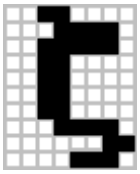
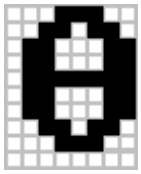
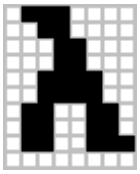
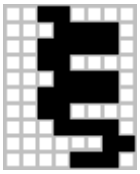
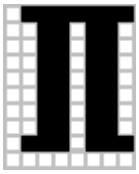
0x6E N	0x6F O	0x70 P	0x71 Q	0x72 R
				

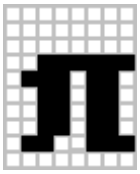
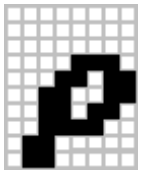
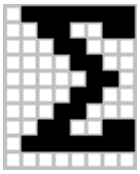
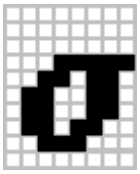
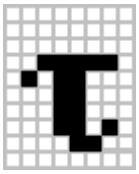
0x73 S	0x74 T	0x75 U	0x76 V	0x77 W
				

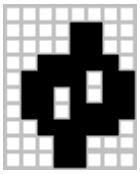
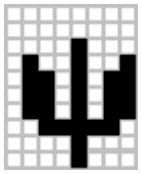
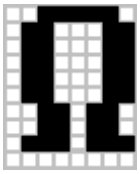
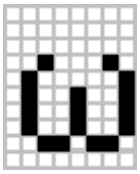
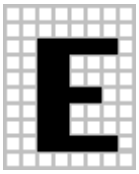
0x78 X	0x79 Y	0x7A Z	0x7B LBRACE	0x7C BAR
				

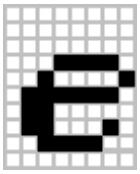
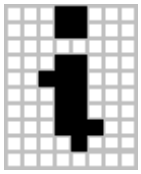
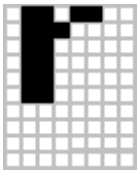
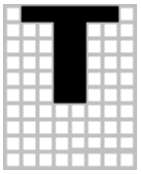
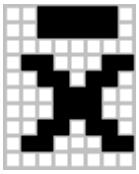
0x7D RBRACE	0x7E TILDE	0x7F OPTION	0x80 ALPHA	0x81 BETA
				

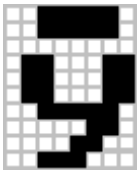
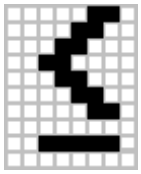
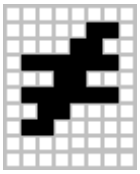
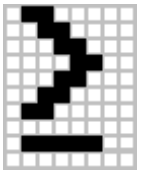
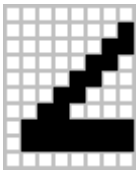
0x82 CAP_GAMMA	0x83 GAMMA	0x84 CAP_DELTA	0x85 DELTA	0x86 EPSILON
				

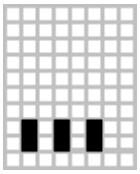
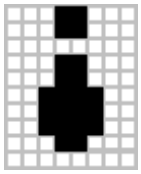
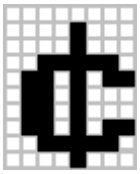
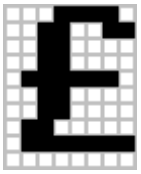
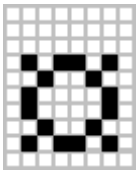
0x87 ZETA	0x88 THETA	0x89 LAMBDA	0x8A XI	0x8B CAP_PI
				

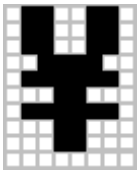
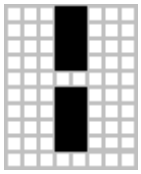
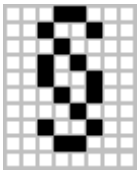
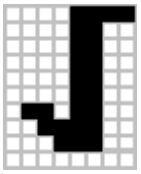
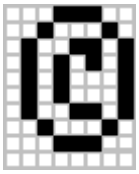
0x8C PI	0x8D RHO	0x8E CAP_SIGMA	0x8F SIGMA	0x90 TAU
				

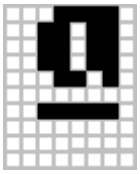
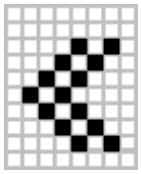
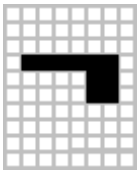
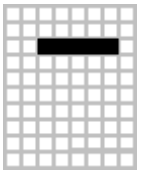
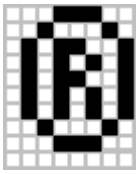
0x91 PHI	0x92 PSI	0x93 CAP_OMEGA	0x94 OMEGA	0x95 EXPONENT
				

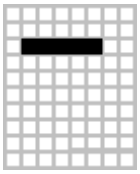
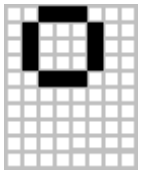
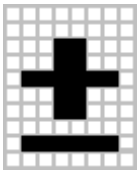
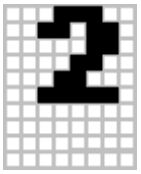
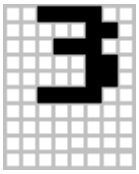
0x96 CONST_E	0x97 IMAG	0x98 RADIANS	0x99 TRANSPOSE	0x9A X_MEAN
				

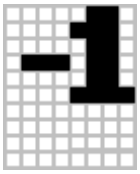
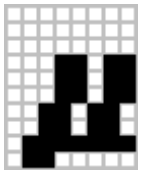
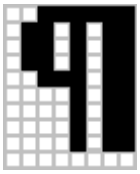
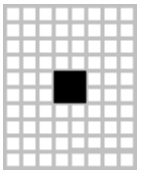
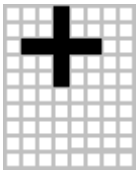
0x9B Y_MEAN	0x9C LE	0x9D NE	0x9E GE	0x9F ANGLE
				

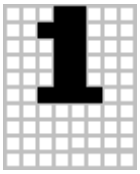
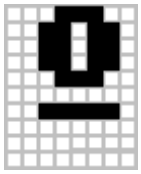
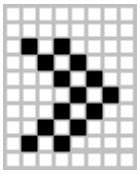
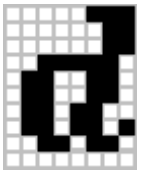
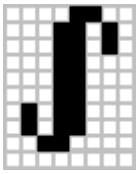
0xA0 ELLIPSIS	0xA1 EXCLAM_DOWN	0xA2 CENT	0xA3 POUND_STIRLING	0xA4 SUNBURST
				

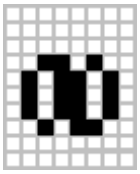
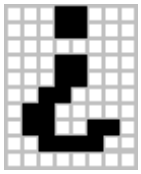
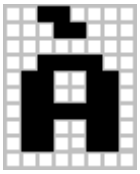
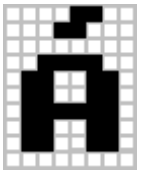
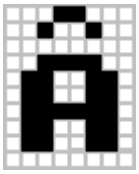
0xA5 YEN	0xA6 SPLIT_BAR	0xA7 SECTION	0xA8 ROOT	0xA9 COPYRIGHT
				

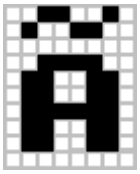
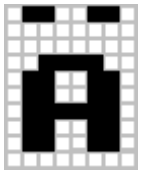
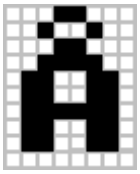
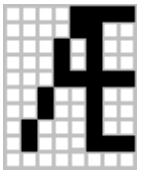
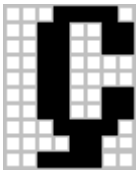
0xAA SUPER_A	0xAB FRENCH_LBRACKET	0xAC LOGICAL_NOT	0xAD NEGATIVE	0xAE REGISTERED
				

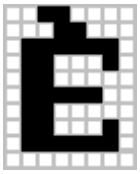
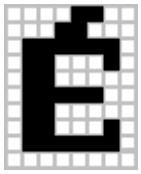
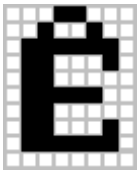
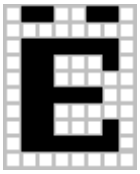
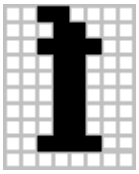
0xAF SUPER_MINUS	0xB0 DEGREES	0xB1 PLUS_OR_MINUS	0xB2 SQUARED	0xB3 CUBED
				

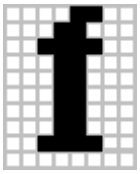
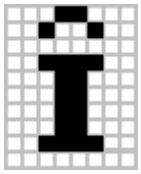
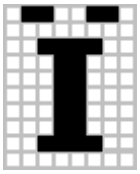
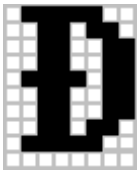
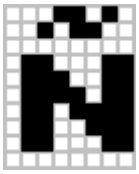
0xB4 INVERSE	0xB5 MU	0xB6 PARAGRAPH	0xB7 MULTIPLY	0xB8 SUPER_PLUS
				

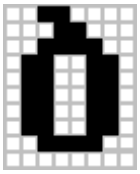
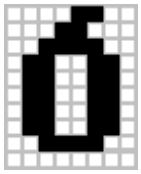
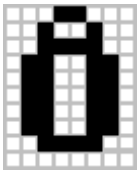
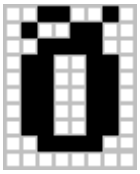
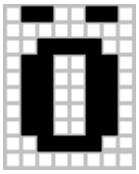
0xB9 SUPER_1	0xBA SUPER_0	0xBB FRENCH_RBRACKET	0xBC DIFF	0xBD INTEGRAL
				

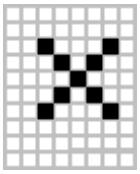
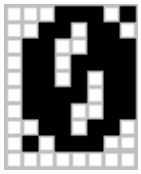
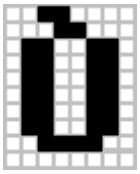
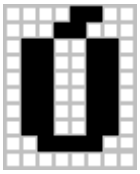
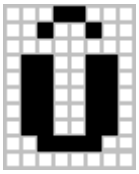
0xBE INFINITY	0xBF QUES_DOWN	0xC0 CAP_A_GRAVE	0xC1 CAP_A_ACUTE	0xC2 CAP_A_CARET
				

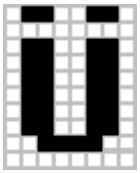
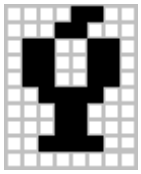
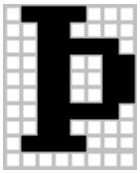
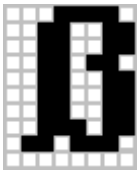
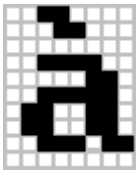
0xC3 CAP_A_TILDE	0xC4 CAP_A_UMLAUT	0xC5 CAP_A_RING	0xC6 CAP_AE	0xC7 CAP_C_CEDILLA
				

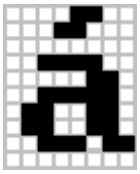
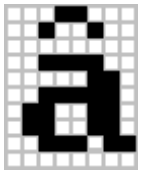
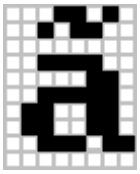
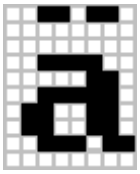
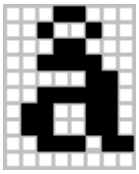
0xC8 CAP_E_GRAVE	0xC9 CAP_E_ACUTE	0xCA CAP_E_CARET	0xCB CAP_E_UMLAUT	0xCC CAP_I_GRAVE
				

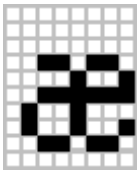
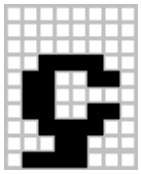
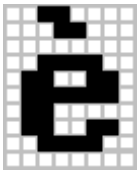
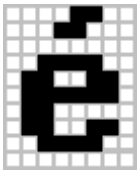
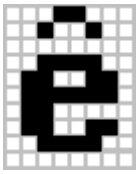
0xCD CAP_I_ACUTE	0xCE CAP_I_CARET	0xCF CAP_I_UMLAUT	0xD0 CAP_D_BAR	0xD1 CAP_N_TILDE
				

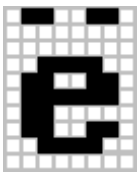
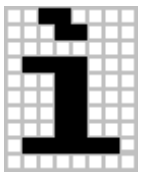
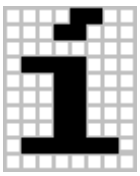
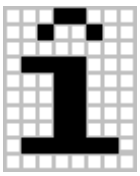
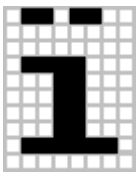
0xD2 CAP_O_GRAVE	0xD3 CAP_O_ACUTE	0xD4 CAP_O_CARET	0xD5 CAP_O_TILDE	0xD6 CAP_O_UMLAUT
				

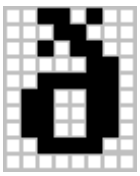
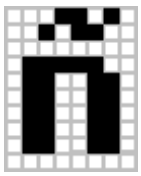
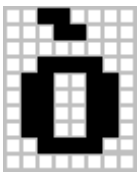
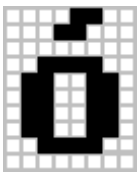
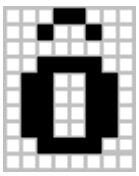
0xD7 TIMES	0xD8 CAP_O_SLASH	0xD9 CAP_U_GRAVE	0xDA CAP_U_ACUTE	0xDB CAP_U_CARET
				

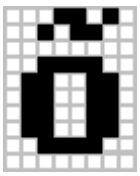
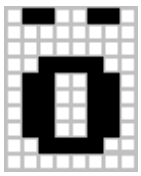
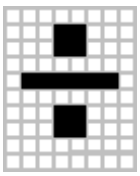
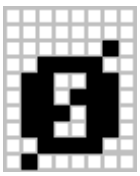
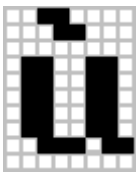
0xDC CAP_U_UMLAUT	0xDD CAP_Y_ACUTE	0xDE CAP_THORN	0xDF LONG_S	0xE0 A_GRAVE
				

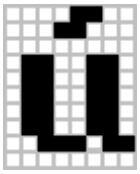
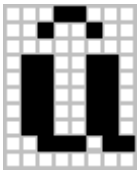
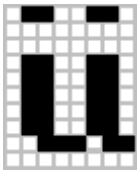
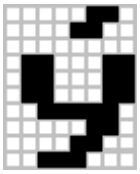
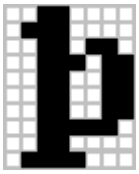
0xE1 A_ACUTE	0xE2 A_CARET	0xE3 A_TILDE	0xE4 A_UMLAUT	0xE5 A_RING
				

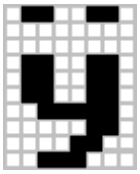
0xE6 AE	0xE7 C_CEDILLA	0xE8 E_GRAVE	0xE9 E_ACUTE	0xEA E_CARET
				

0xEB E_UMLAUT	0xEC I_GRAVE	0xED I_ACUTE	0xEE I_CARET	0xEF I_UMLAUT
				

0xF0 D_BAR	0xF1 N_TILDE	0xF2 O_GRAVE	0xF3 O_ACUTE	0xF4 O_CARET
				

0xF5 O_TILDE	0xF6 O_UMLAUT	0xF7 DIVIDE	0xF8 O_SLASH	0xF9 U_GRAVE
				

0xFA U_ACUTE	0xFB U_CARET	0xFC U_UMLAUT	0xFD Y_ACUTE	0xFE THORN
				

0xFF Y_UMLAUT				
				

Reference List — System Routines

A

B

C

D

E

F

G

H

I

K

L

M

N

O

P

Q

R

S

T

V

W

X

Y

Reference List — Global Variables

G

I

M

N

O

R

S

T

Reference List — Macros

A

D

E

F

I

O

T