JAM Release 5.04 Upgrade Guide

TABLE OF CONTENTS

This addendum describes several new features available in JAM 5.04:

- Menu bars are now available for character–mode applications, including JAM's authoring tools. You can use menu bars developed for graphics environments in character-mode applications, and vice-versa.
- Drop shadows and graying out underlying windows emphasize the active window's appearance.
- The setup variable `SCR_KEY_OPT` is now available for general usage. You can use this variable to enable or disable "remote" scrolling of scrolling arrays.

This addendum contains five chapters:

1 – *Menu Bars* shows how to define menu bars and attach them to your application. It also shows how to manage menu bars at runtime.

2 – *Menu Bar Reference* shows how JAM defines menu bar data and describes the menu bar routines that you can use in your applications.

3 – *Menu Bar Utilities* describes utilities that JAM provides to create and install menu bars.

4 – *Window Display Emphasis* shows how to give drop shadows to windows and gray out their contents.

5 – *Remote Scrolling* shows how to enable scrolling for arrays when the cursor is located outside an arrayed field.

1 Menu Bars

A menu bar is a horizontal menu at the top of the screen that has one or more items. Each item can invoke a pulldown menu—that is, a vertical menu that appears directly below its parent item. Pulldown menu items can themselves invoke submenus. You can nest pulldown menus and their submenus multiple levels deep.

You define menu bars and their pulldowns in ASCII scripts. The script describes the content of the menu bar, the action associated with each item on the menu bar, and its initial status. When you finish defining the menu, you convert the menu script to binary format through the utility `menu2bin`.

After you define a menu bar, you attach it to your application through either JAM's screen editor or JAM library routines. When you attach a menu, you specify whether it is available to the entire program, to other menus, or only to a specific screen or context.

JAM also provides library routines that let you change the content and selection of menu bars at runtime. These are described in Chapter 2 of this addendum.

1.1 MENU BAR SYSTEM

JAM's menu bar subsystem manages the display and behavior of menu bars and their pulldowns. In graphical environments such as Windows and Motif, menu bars use the environments windowing system. In character-mode applications, JAM uses its own resources to display the menus.

If you have menu bars enabled, JAM initializes the menu bar subsystem at startup. It then reads, or loads, menu bars as they are called by the application, either through their associated screens, or through explicit calls to `sm_mn_r_menu` or `sm_mn_d_menu`.

When JAM loads a menu bar, it examines its scope value to decide whether to display it, and when. A menu bar gets its scope value when you attach it to the application. For example, this statement:

```
sm_r_menu (warning, KS_OVERRIDE);
```

specifies to load the menu `warning` at a scope of KS_OVERRIDE.

JAM menu bars can have one of these five scopes:

- KS_FORM associates a menu bar with a screen. This menu bar is displayed with the screen, and with sibling and child windows that lack their own menu bars. JAM can load only one screen-level menu bar at a time. Thus, if two screens with their own menu bars open in succession, JAM unloads the first screen's menu bar before it loads the second.

You can attach a menu bar to a screen through the screen's keyset field, found on the Screen Characteristics screen. When JAM displays the screen, it automatically loads its menu bar at KS_FORM scope. Alternatively, you can load a screen-level menu bar through a call to `sm_r_menu` or `sm_d_menu` in the screen's entry procedure.

- KS_APPLIC associates a menu bar with the application. Application menu bars are accessible to the application and to screens that lack their own menu bar. You load an application-level menu bar through a call to `sm_r_menu` or `sm_d_menu` in the application's main routine (`jmain.c` or `jxmain.c`), in the area reserved for code to execute before the first screen appears.
- KS_OVERRIDE specifies a menu bar that is independent of any stage of program execution. An override menu has precedence over any other menu bars that are loaded at the same time.

  You can load and unload override menus at any stage of program execution. While multiple override menus can be loaded simultaneously, only one override menu can be active at a time. The active override menu is the most recently loaded one.

  JAM reads all override menus into a save stack, where the program can access them in last-in/first-out order. The save stack can hold up to 10 override menus. JAM sometimes uses the save stack for its own override menus. Consequently, JAM might temporarily push its own override menus on top of user-defined menus loaded earlier.
- KS_MEMRES specifies a menu bar script that is memory-resident and available to other menus at runtime. JAM can maintain as many of these scripts as your system's resources allow. You typically load these scripts at startup, in the application's main routine.

  A memory-resident menu bar is available to other menu bars only as an external menu—that is, a menu that is defined outside the menu that references it. You should install at this scope any menu that is used repetitively by more than one menu.
- KS_SYSTEM specifies the menu bar that JAM uses in its authoring environment. The system menu is loaded by default at startup in the application's main routine, `jmain.c` or `jxmain.c`. Users can then toggle between display of the system menu and the application-level or screen-level menu through the SFTS key or the **Switch Scope** menu item. When JAM switches to the system menu, it closes any screen-level or application-level menu bars that might be loaded, and vice-versa.

Although multiple menus can be loaded simultaneously, only one menu can be displayed at a time. One exception applies: Motif allows simultaneous display of the application-level and screen-level menus.

Within each of the following groups, JAM can simultaneously load menus of these types:

KS_OVERRIDE	KS_OVERRIDE
KS_SYSTEM	KS_FORM
	KS_APPLIC

For example, an application can have one or more override menu bars loaded along with one system-level menu. Or, the application can have one screen-level and one application-level menu bar loaded along with several override menu bars. Note that menu bars of KS_SYSTEM scope are loaded to the exclusion of KS_FORM and KS_APPLIC menu bars, and vice-versa.

The previous groups of menu scopes also show their order of precedence. JAM uses this order to determine which of the loaded menus to display. Thus, if an override menu and a screen menu are loaded, JAM displays the override menu. When the override menu closes, JAM displays the screen menu.

If a window without a screen-level menu bar opens, the previously active menu bar remains displayed. This can be the screen-level menu bar from the previous screen, or the application-level menu bar if no screen-level menu bar is loaded.

Motif and OPEN LOOK treat menu bars of different scopes as follows:

- Menu bars can appear on individual screens or on the base screen, depending on their scope and the value of the `formMenus` resource. If `formMenus` is true, the application-level menu bar appears on the base screen while the screen-level menu bar appears local to the screen. Therefore, both can be active at the same time. If a screen without a screen-level menu bar opens, then no menu bar appears local to the screen. Screen-level and override-level menu bars can appear either local to the screen or on the base screen.
- Application-level and system-level menu bars are restricted to the base screen. However, users can always access them as pop up menus by pressing the third mouse button.

1.2 MENU SCRIPTS

When you create a menu bar, you first define it in an ASCII script. You then convert the script to binary with the `menu2bin` utility. A menu script specifies a menu bar and its pulldowns. The first menu that you specify in a script defines the the top level menu, or menu bar; subsequent menu definitions define pulldown menus and their submenus. You can can nest multiple menus any number of levels deep.

You define a menu with this syntax:

```
menuname[ separator [type]] [ display-option]...
{
    "label" action[ display-option]...
    ...
}
```

Each menu definition begins with a unique identifier, *menuname*. The contents of the menu consist of menu items and separators, enclosed by curly braces `{}`. Except for title items, you can specify items and separators in any order.

Alternatively, a menu script can reference an external menu—that is, a menu that is defined outside the current script. JAM searches for an external menu among currently open menus, then among those menus loaded at the scope `KS_MEMRES`. You specify an external menu as follows:

menuname `external`

The `external` keyword lets you build menu bars in a modular fashion and helps ensure consistency across different menu bars. For example, you can write a script for a pulldown that is repeatedly used by different menu bars. The menu bar scripts can then reference the pulldown as an external menu.

Menu scripts can also include lines of commented text. Prefix each comment line with a pound sign #.

The menu bar compiler ignores all white space characters—spaces, tabs, and line returns—except when they separate key words. You can use white space to improve the menu definition's legibility.

The following sections describe individual components of a menu definition.

1.2.1 Menu Name

A menu definition begins with a unique identifier. Each identifier can take one or more display option arguments which JAM applies uniformly to the entire menu. For more information about display options, see "Text Options" on page 9, and "Separator Types" on page 10.

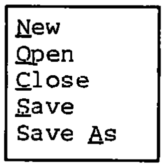
1.2.2 Label

Labels specify the displayed text of menu items. Each label is enclosed in double quote marks (" "). The menu bar compiler accepts labels with up to 255 characters. The label can include backslash escape characters—for example, `\n` and `\t`—to specify newlines, tabs, and quotes. JAM uses these formats only if the environment supports them; otherwise, their actual display is terminal-dependent.

To specify a keyboard mnemonic for a menu item, place an ampersand (`&`) in front of the desired character. Users can select the menu item from the keyboard by typing this character. JAM sets off the mnemonic character according to the display emphasis style that you choose—for example, by underlining or highlighting it. For example, given the following pulldown menu definition:

```
FormMenu
{
    "&New"      key PF1
    "&Open"     control "^jm_filebox file /usr/home * File"
    "&Close"    key PF3   inactive
    "&Save"     key PF3
    "Save &As"  key PF4
}
```

JAM might display the menu like this:



For more information on setting off the display of mnemonic characters in character-mode applications, see "Setting Menu Display and Behavior" on page 16.

1.2.3 Action

The action that you assign to a menu item specifies its behavior. You can use one of the following keywords:

`control` *control-string*	Associates the JAM control string *control-string* with this menu item.
`edit`	Specifies that this menu item invokes the edit pulldown. The edit pulldown typically contains these items: `Cut`, `Copy`, `Paste`, `Delete`, `Select All`. Character-mode applications ignore this action.
`key` *keystroke*	Specifies to return *keystroke* when users select this item. Selection of this menu item is equivalent to pressing the key. The value of *keystroke* can be a JAM logical key. You can also specify a hex, binary or octal number through one of these leading characters: *0x* hex *0b* binary *0* octal
`menu` *menuname*	Specifies that this menu item invokes the submenu *menuname.*
`separator` [*type*]	Inserts a separator of the specified type between menu items. If you omit the type, JAM draws a single-line separator. See "Separator Types" later in this chapter for information on different separator display options.

`title`	Specifies that *label* is the title of this menu. The title must be the first entry in the menu. Pi/Windows applications ignore the `title` keyword.
`windows`	Specifies that this menu item invokes the Windows menu. This menu lists the names of the open screens. When you select a screen from this menu, JAM brings it to the top of the display. If the selected screen is a sibling of the screen at the top of the window stack, it becomes the top JAM window. In Windows, the Win-dows menu also contains these items: `Cascade`, `Tile` and `Arrange Icons`. These let you arrange screens and icons within the frame. In Motif and OPEN LOOK, the Windows menu contains a `raise all` option that raises all JAM screens to the top of the display, and layers them according to the window stack. In character-based applications, the Windows menu lists only sibling windows, with the last-opened window listed first.

1.2.4

Text Options

You can tell JAM how you want menu items to appear. The following display options are available:

`grayed/greyed`	Mutes, or grays out, the menu item text and prevents users from selecting it.
`help`	Makes a menu item the rightmost item on a menu bar. You can specify this display option for only one menu item, and only if it appears on a menu bar—that is, the script's first, or main, menu definition. If this item is not the last-specified item in the menu definition, JAM rearranges the menu item order so that it appears last.
`inactive`	Inactivates the menu item. When users click on this item, nothing happens.
`indicator`	Indents all menu items to the right and reserves the indented space for the indicator symbol.

`indicator_on`	Turns on the indicator symbol for this item. The indicator—typically an *X* or check mark √—indicates the state of a menu item that serves as a toggle switch. To change the mark character for character mode applications, assign a new value to `MARKCHAR` in the video file. Use this option only if you have also specified the display option `indicator`.
`showkey`	If the menu item's action is `key`, shows the keytop label from the key file to the right of the item's text. If the key file has no keytop, then the key mnemonic is shown.

Some options are valid only for certain actions. The following table shows which display options are valid for each action:

Action	**Display Options**					
	`grayed`	`help`	`inactive`	`indicator`	`indicator_on`	`showkey`
`control`	●	●	●	●	●	
`edit`	●	●	●			
`key`	●	●	●	●	●	●
`menu`	●	●	●			
`title`	●					
`windows`	●	●	●			

1.2.5

Separator Types

JAM offers several ways to separate menu items. An unqualified `separator` action inserts a single or blank line, depending on your system, between the previous and next menu items. You can specify one of the following separator types:

`double`	Inserts a double line.
`double_dashed`	Inserts a double-dashed line.
`etchedin`	In Motif, draws a single line that appears to be etched into the menu. In character-based applications, draws a dotted line.

`etchedout`	In Motif, draws a single line that appears to protrude from the menu. In character-based applications, draws underscores.
`menubreak`	Starts a new line in a horizontal menu, or a new column in a vertical menu.
`noline`	Inserts extra space between the menu items.
`single`	Draws a single line.
`single_dashed`	Draws a single dashed line.

In character-based applications, `single` and `double` can use characters defined in the video file. If no definition exists in the video file, JAM uses its own default values: _, =, | and ||.

You must enter separator types in lower case.

Some separator options are valid only within certain environments. If you specify a separator that your environment does not recognize, JAM uses a blank line (`noline`). The following table shows which separator types are valid for each environment.

Separator type	**Environment**			
	Character	Motif	OPEN LOOK	Windows
`double`	●	●		
`double_dashed`	●	●		
`etchedin`	●	●		
`etchedout`	●	●		
`menubreak`				●
`noline`	●	●		
`single`	●	●	●	●
`single_dashed`	●	●		

1.2.6 Global Menu Settings

Set the separator type and other display options for the entire menu by specifying them after the menu name. For example, if you specify global options `noline` and `showkey`, all separators in the menu default to `noline` and all keys in the menu have `showkey`.

If you specify a global separator type, you can override it for individual separators in the menu.

1.2.7 Comments

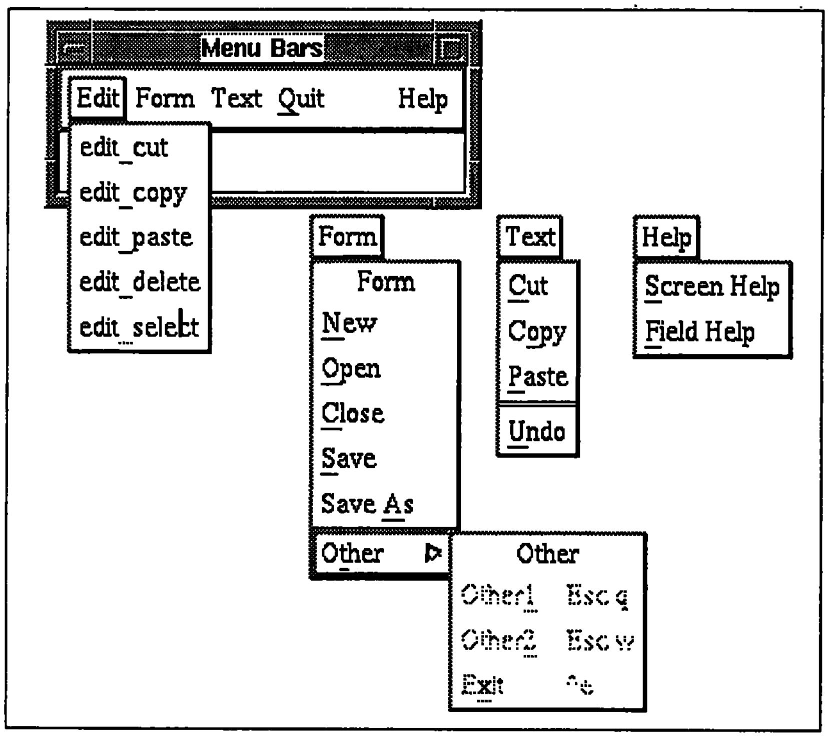
You can insert one or more comment lines anywhere in the script. Each comment line must begin with the pound sign `#`.

1.3 SAMPLE MENU SCRIPT

This section contains a sample menu script. The figure after it shows how this menu appears in Motif.

```
# The first menu definition specifies the menu bar items

Main
{
        "Edit"  edit
        "Form"  menu FormMenu
        "Text"  menu TextMenu
        "Help"  menu HelpMenu help
        "&Quit" key  0x103
}

FormMenu
{
        "Form"     title
        "&New"     key PF1
        "&Open"    control "^jm_filebox file /usr/home * File"
        "&Close"   key PF3   inactive
        "&Save"    key PF3
        "Save &As" key PF4
        ""        separator etchedin
        "O&ther"  menu      OtherMenu
}

OtherMenu grayed showkey
{
        "Other" title
        "Other&1" key PF1
        "Other&2" key PF2
        "E&xit" KEY EXIT
}

TextMenu
{
        "&Cut"   KEY PF1
        "C&opy"  key PF2
        "&Paste" Key PF3
        ""       sEpArAtOR double menubreak
        "&Undo"  Key SPF1
}

# An external menu is one that is defined elsewhere, either
# in an open menu or at the scope KS_MEMRES.

HelpMenu external
```

This script produces the following output in Motif:



Menu bar and pulldown menus produced by the sample menu script.

1.4 CONVERTING AND STORING MENU BARS

After you write a menu script, use the menu2bin utility to convert the script file to binary format. You can store the binary file to a disk file or to a library. This done, you can attach the converted file to an application, as described later.

You can also store menu bar data in memory. To do this, convert the binary file to a C structure with the bin2c utility, then register it to JAM with sm_formlist. When you store a menu bar's data in memory, JAM compiles it with the application. You can then read these files with sm_d_menu.

For information on menu2bin, see "Menu Bar Utilities" later in this addendum. For information on bin2c and registration of C structures, see the JAM *Programmer's Guide*.

1.5 ATTACHING MENU BARS TO AN APPLICATION

An application that uses menu bars must call each menu bar at the appropriate execution stage. Through JAM's screen editor, you can attach a menu bar to a screen by simply entering the menu name in the Screen Attributes **keyset** field. At runtime, JAM loads the menu bar when the screen opens either as a form or as a window, or when it becomes the topmost window. JAM unloads the menu bar when the screen closes or is superseded by a window with its own menu bar.

Alternatively, you can explicitly load and unload menu bars through calls to one of these routines:

sm_r_menu	Reads menu bar data from memory, a library or disk.
sm_d_menu	Reads menu bar data from memory. Call this routine only for menu bar scripts that are compiled with the application, as described on page 14, "Converting and Storing Menu Bars."
sm_c_menu	Unloads menu bar data and frees the memory associated with it. If the menu bar is still displayed, JAM removes it at the next screen write.

Both sm_r_menu and sm_d_menu require you to supply a scope value. Menu bars that you attach to a screen through JAM's screen editor automatically get a scope of KS_FORM.

If you attach a menu bar to a screen through JAM routines, you load and unload it in the screen's entry and exit functions, respectively. Override menus (KS_OVERRIDE) can be read at any stage of program execution. Menu bars of all other scopes—KS_APPLIC, KS_SYSTEM, and KS_MEMRES—are typically read in the program's main routine.

1.6 MANAGING MENU BARS

JAM provides library routines to manipulate menu bars and their pulldowns at runtime. For example, you can use `sm_mnchange` to gray out or activate items according to changes in the screen's context. These library routines are summarized in the following table. See detailed descriptions of each routine in Chapter 2, "Menu Bar Reference."

Runtime routine	Description
`sm_mnadd`*	Adds an item to the end of a menu.
`sm_mnchange`*	Alters a menu item (for example, grays out an item).
`sm_mndelete`	Deletes a menu item.
`sm_mnget`*	Gets menu item information.
`sm_mninsert`*	Inserts a new menu item.
`sm_mnitems`	Gets the number of items on a menu.
`sm_mnnew`	Creates a menu by name.

* Cannot be prototyped because the routine uses an external data structure.

Because changes to a shared menu bar are passed on to all other screens that use it afterward, be sure that menu bar changes for one screen are correct for all later screens. If you want to omit a menu bar for a specific screen, create a dummy menu bar for it.

You can also refresh the menu to its original state by closing it with `sm_c_menu`, then reopening it with `sm_d_menu` or `sm_r_menu`.

You must prototype any menu bar functions that you call directly from control strings and JPL procedures. The *Programmer's Guide* shows how to prototype and install functions. See also *JPL Guide* for more information on how JPL uses prototyped functions.

1.7 SETTING MENU DISPLAY AND BEHAVIOR

You can control the way menu bars appear and behave in character-based applications by setting various options. You can set these options through `sm_option` or by editing SMSETUP or SMVARS.

You can specify different display attributes for menu items and their keyboard mnemonic characters so that users can easily distinguish between available and unavailable items. Unavailable items typically appear to be grayed out.

JAM uses the following algorithm for graying menu items and keyboard mnemonic characters:

1. The original display attributes are AND'd with the set of display attributes that you specify to retain.
2. The remaining attributes are OR'd with the attributes that you specify for graying.
3. The result of steps 1 and 2 is exclusive-OR'd with a mask of switch attributes.

You can set different display attributes for a menu item string and its keyboard mnemonic character. If you specify display attributes only for the menu item string, the mnemonic character inherits its attributes.

The following tables show the variables that control display attributes and their default values.

Table 1. Graying attributes.

Variable	Description	Default Attributes
`FE_KEEPATTRS`	Display attributes that are kept for grayed items.	All attributes
`FE_SETATTRS`	Display attributes that are set for grayed items.	None
`FE_SWATTRS`	Display attributes that are toggled when graying is turned on and off.	`HILIGHT`

Table 2. Keyboard mnemonic character emphasis attributes.

Variable	Description	Default Attributes
`AC_KEEPATTRS`	Display attributes that are kept for a keyboard mnemonic character	All Attributes
`AC_SETATTRS`	Display attributes that are turned on for a keyboard mnemonic character.	None
`AC_SWATTRS`	Display attributes that are switched for a keyboard mnemonic character.	`HILIGHT|WHITE`

The following table describes options for controlling user interaction and setting menu styles:

Table 3. Settings for menu styles and user interaction.

Variable	Description	Default Attributes
MB_BORDSTYLE	Border style to use in menus (0-9). NOBORDER specifies no border.	1
MB_BORDATT	Menu border attributes.	B_WHITE | BLACK
MB_DISPATT	Display text attributes.	B_WHITE | BLACK
MB_FLDATT	Menu field attributes	B_WHITE | BLACK
MB_HBUTDIST	Distance between two buttons in a horizontal windows-style menu.	2
MB_LINES_PROT	Number of top lines reserved for a menu bar.	1
MB_SYSTEM	Determines whether the System item (==) appears on the menu bar or not. Two options are available: OK_SYSTEM specifies to display System. NO_SYSTEM specifies to omit System.	OK_SYSTEM

2.8 ENABLING MENU BAR SUPPORT

An application that uses menu bars must be enabled to handle them:

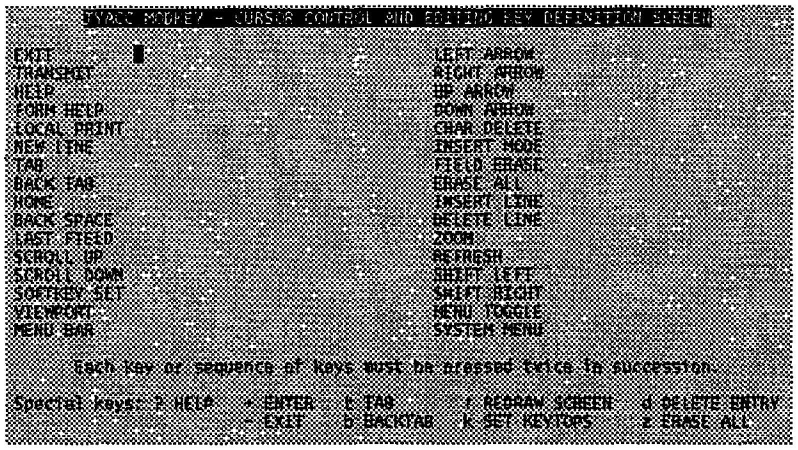
- Create a key file or modify an existing one that defines the keys used to access menu items.
- Optionally, edit your video file and change the default settings for the menu marker character MARKCHAR and submenu indicator SUBMNSTRING.
- Recompile `jmain.c` and, optionally, `jxmain.c`, and rebuild the executable.

2.8.1 Modifying the Key File

You can give users keyboard access to menu items by defining two new types of keys in your key file:

- MNBR lets users access the menu bar. If you are enabling system menu bars, you can also define ALSYS. This gives users keyboard access to the System menu, represented by two equals signs ==.
- ALT keys give users direct access to menu bar items from the screen.

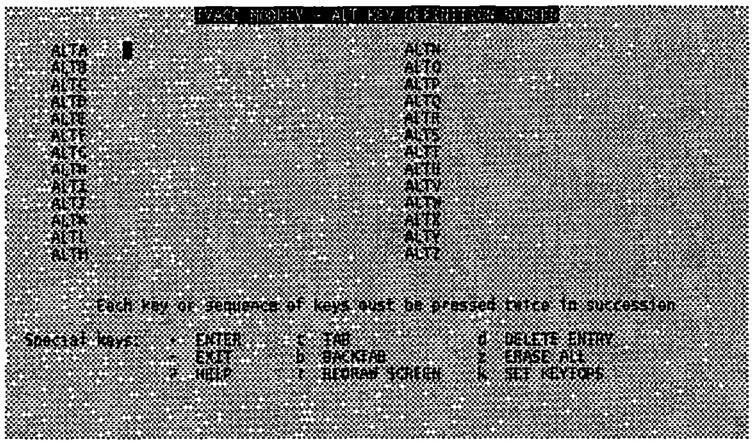
You perform both tasks by modifying an existing key file. You can do this directly, or use JAM's MODKEY utility. Start MODKEY and edit the desired key file. Select *Define Cursor Control and Editing Keys.* JAM displays this screen:



```
         JYACC MODKEY - CURSOR CONTROL AND EDITING KEY DEFINITION SCREEN

EXIT                                         LEFT ARROW
TRANSMIT                                     RIGHT ARROW
HELP                                         UP ARROW
FORM HELP                                    DOWN ARROW
LOCAL PRINT                                  CHAR DELETE
NEW LINE                                     INSERT MODE
TAB                                          FIELD ERASE
BACK TAB                                     ERASE ALL
HOME                                         INSERT LINE
BACK SPACE                                   DELETE LINE
LAST FIELD                                   ZOOM
SCROLL UP                                    REFRESH
SCROLL DOWN                                  SHIFT LEFT
SOFTKEY SET                                  SHIFT RIGHT
VIEWPORT                                     MENU TOGGLE
MENU BAR                                     SYSTEM MENU

      Each key or sequence of keys must be pressed twice in succession.

Special keys: ? HELP   + ENTER   t TAB       r REDRAW SCREEN   d DELETE ENTRY
                       - EXIT    b BACKTAB   k SET KEYTOPS     z ERASE ALL
```

The bottom of the screen contains two entries, MENU BAR and SYSTEM MENU. Define the desired key sequences for MENU BAR and, optionally, SYSTEM MENU—for example, ALT-X and ALT-=. The key sequence for SYSTEM MENU is optional because users can access it indirectly through the MENU BAR key.

After you define the key sequences, exit the screen and return to the menu. Select *Define Alt Keys.* MODKEY displays this screen:



```
                [illegible] MODKEY - ALT KEY DEFINITION SCREEN

     ALTA                                   ALTN
     ALTB                                   ALTO
     ALTC                                   ALTP
     ALTD                                   ALTQ
     ALTE                                   ALTR
     ALTF                                   ALTS
     ALTG                                   ALTT
     ALTH                                   ALTU
     ALTI                                   ALTV
     ALTJ                                   ALTW
     ALTK                                   ALTX
     ALTL                                   ALTY
     ALTM                                   ALTZ

        Each key or sequence of keys must be pressed twice in succession

   Special keys:   + ENTER     t TAB             d DELETE ENTRY
                   - EXIT      b BACKTAB         z ERASE ALL
                   ? HELP      r REDRAW SCREEN   k SET KEYTOPS
```

You can now define key sequences that give direct access to menu bar items. For example, you might specify ALT-F to access the menu item **File**.

After you define ALT keys, save the definitions to a file. Then convert this file to binary format with `key2bin`. If the key file is new, add it to the `SMVARS` file.

2.8.2
Modifying the Video File

You can edit the video file as follows:

- Change the value of `MARKCHAR`, which specifies the character used to check menu items. For example, under MS-DOS and JTERM, the following statement specifies the square root symbol (√) as the mark character:

  ```
  MARKCHAR = 0xFB
  ```

- Change the value of `SUBMNSTRING`, which defines the string that menus use to indicate that a menu item invokes a submenu.

After you edit the video file, convert it to binary format with `vid2bin`.

2.8.3
Rebuilding the Executable

After you modify your key and video files, you can build an executable that uses menu bars:

1. Recompile jmain.c with MENUS defined to 1. If you want JAM's authoring environment to include menu bars, recompile `jxform.c` with the same change. Edit the section that specifies optional subsystems to include this statement:

   ```
   #define MENUS 1
   ```

   This tells JAM to initialize the menu bar subsystem at startup.

   Alternatively, add a definition on the compile command line. On UNIX systems, be sure that CFLAGS is set to this value:

   ```
   -DMENUS=1
   ```

   For example:

   ```
   CFLAGS = -I$(HPATH) -O -Aa -DSM_SCCSID -D$(MACHINE_NAME)\
   -DMENUS=1
   ```

2. Relink with the new 5.04 libraries.

1.9 TESTING MENU BARS

You can test menu bars in application mode of JAM if the following conditions are true:

- The SFTS key is defined in the key translation file. This key lets you toggle between the menu bars in your application and JAM's own menu bar.
- If you are testing character-mode applications, you must edit `jxmain.c` to enable menu bars, and rebuild `jxform`. Section 1.8.3 shows how to do this.

1.10 USING MENU BARS AND PULLDOWNS

You can use either the mouse or the keyboard to access menu bars and their pulldowns.

Use the mouse with menus as follows:

1. Click on the menu bar. If you click on a menu item, its pulldown menu appears. If the menu item has no pulldown menu, JAM executes the action associated with the menu item. If you click outside a menu item, JAM selects the menu bar's first item, but does not open its pulldown menu.
2. Exit the menu bar by clicking on the previously active screen.

Access the menu bar from the keyboard through one of keys listed below. Recall that you must first define these keys in the key file.

- `MNBR` selects the menu bar's first item and leaves its pulldown menu closed.
- `ALSYS` selects System (═) and opens its menu.
- `ALT`-*char* selects the menu bar item with the keyboard mnemonic *char*.

You can exit the menu bar by pressing either `MNBR` or `EXIT`.

When you switch between screens and menus, JAM maintains the following controls over your work space:

- When you activate a menu bar, JAM saves the state of the screen and keeps all submenus already open.
- When you exit a menu bar and return to its screen, JAM closes all of the menu bar's pulldowns and submenus. Only the menu bar remains visible.
- When the menu bar is active, you cannot manipulate any windows—for example, move or resize them.
- If a message requires user action, JAM prevents you from switching to a menu until you perform the required action.

1.11 MENU BARS AND SOFT KEYS

Soft keys and menu bars are mutually exclusive, because they share the same programmatic hooks. You must choose one or the other. The selection of soft keys versus menu bars is made in the `main` routine, either `jmain.c` or `jxmain.c`, by initializing either soft key support or menu bar support.

JAM provides the `kset2mnu` utility to help you convert keysets to menu bars. This utility converts the keyset to an ASCII menu script. Because the organization of menu bars and keysets can differ greatly, you will probably want to edit `kset2mnu`'s initial output. You can then convert the script to binary format and install it as described earlier.

The `kset2mnu` utility is described in "Menu Bar Utilities" later in this addendum.

2 Menu Bar Reference

This section shows the data structure that JAM uses to modfy or examine menu bar data. It also describes the routines that you can use to create, install, change, and display menu bars. These descriptions appear in alphabetical order.

2.1 MENU BAR DATA

JAM's `item_data` structure lets you change the display or behavior of menu bar items, or to examine a menu item's current state. Some of the runtime routines described in this chapter use `item_data` as a parameter. This structure has the following definition in `smmenu.h`:

```
struct item_data
{
   short *type
   char *label
   short accel
   short key
   char *submenu
   short option
}
```

Each of the structure's members is described below.

`short *type`

Specifies this menu item's type through one of the following defines:

Constant	Value	Description
`MT_SEPARATOR`	0	Inserts a separator of the specified type between menu items.
`MT_TITLE`	1	Uses the item's label as the menu title.
`MT_KEY`	2	Specifies to return a keystroke when users select this item.
`MT_SUBMENU`	3	Invokes a submenu.
`MT_EDIT`	4	Specifies that this menu item invokes the Edit pulldown menu.

Constant	Value	Description
MT_WINDOWS	5	Specifies that this menu item invokes the Windows pull-down menu.
MT_CTRLSTRNG	6	Associates a JAM control string with this menu item.

`char *label`

The text of this menu item, ignored if `type` has a value of `MT_SEPARATOR`. Text beyond 255 characters is truncated. The default value is `0`.

`short accel`

The offset of the character in `label` that is used as to select this menu item from the keyboard. The default value is `-1`.

`short key`

The logical key number of the key that is returned on selection of this menu item, valid only if `type` has a value of `MT_KEY`. See `smkeys.h` for a listing of valid key mnemonics. The default value is `0`.

`char *submenu`

If `type` is `MT_SUBMENU`, specifies the menu invoked from this menu item. If `type` is `MT_CTRLSTRNG`, specifies the control string to execute when this item is selected.

`short option`

Display options for this menu item. If the menu item displays the text of `label`—that is, `type` has any value except `MT_SEPARATOR`—you can bitwise OR together the following text display options:

Constant	Value	Description
MO_INDICATOR_ON	0x0200	Turns on the indicator symbol for this item. The indicator—typically a check mark √—indicates the state of a menu item that serves as a toggle switch.
MO_INDICATOR	0x0800	Indents all menu items to the right and reserves the indented space for the indicator symbol.
MO_GRAYED	0x1000	Grays out the menu item text and prevents users from selecting this item.
MO_INACTIVE	0x2000	Makes the entry inactive. When users click on this item, nothing happens.

Constant	Value	Description
MO_SHOWKEY	0x4000	Shows the keytop label from the key file to the right of the item's text. If the key file has no keytop, then the key mnemonic is shown
MO_HELP	0x8000	Makes a menu item the rightmost item on the menu bar. Valid only for one item, and only if it appears on the menu bar.

If the menu item has type set to MT_SEPARATOR, you can set one of the following options:

Constant	Value	Description
MO_SINGLE	0x0000	Draws a single line.
MO_DOUBLE	0x0001	Inserts a double line.
MO_NOLINE	0x0002	Inserts extra space between the menu items.
MO_SINGLE_DASHED	0x0003	Inserts a single–dashed line.
MO_DOUBLE_DASHED	0x0004	Inserts a double–dashed line.
MO_ETCHEDIN	0x0005	Draws a single line that appears to be etched into the menu.
MO_ETCHEDOUT	0x0006	Draws a single line that appears to protrude from the menu.
MO_MENUBREAK	0x0400	Starts a new line in a horizontal menu, or a new column in a vertical menu.

2.2 MENU BAR ROUTINES

The following routines create, alter, install and display menu bars:

`sm_c_menu`	Closes a menu bar.
`sm_d_menu`	Displays a menu bar stored in memory.
`sm_mncrinit`	Initializes menu bar support.
`sm_mn_forms`	Installs menu bars in memory.
`sm_mnadd`	Adds an item to the end of a menu.
`sm_mnchange`	Changes an item.
`sm_mndelete`	Deletes an item.
`sm_mnget`	Gets information about a menu item.
`sm_mninsert`	Inserts a new item in a menu.
`sm_mnitems`	Gets the number of items in a menu.
`sm_mnnew`	Creates a menu bar.
`sm_r_menu`	Reads and displays a menu bar from memory, a library or disk.

Like other JAM library routines, menu bar routines require the header file `smdefs.h`. Some routines also require you to include other header files, as indicated in the descriptions that follow.

You must prototype any menu bar functions that you call directly from control strings and JPL procedures. JAM's *Programmer's Guide* shows how to prototype and install functions.

The following sections describe menu bar routines in greater detail. The routines are listed alphabetically.

c_menu

close a menu bar

SYNOPSIS

```
#include "smsoftk.h"

int sm_c_menu(int scope);
```

PARAMETERS

`int scope`

Specifies when this menu is available to the application with one of these arguments:

```
KS_FORM
KS_APPLIC
KS_OVERRIDE
KS_MEMRES
KS_SYSTEM
```

DESCRIPTION

This routine closes the menu bar at the specified scope level and frees all memory allocated for it. If the menu bar is displayed, JAM removes it at the next delayed write.

When a menu bar with a scope of `KS_OVERRIDE` closes, JAM pops the next menu, if any, off the override stack.

If the closed menu's scope is `KS_MEMRES`, JAM closes the last menu bar loaded at that scope.

To refresh a menu bar, close it with `c_menu`, then reload it with `r_menu` or `d_menu`.

RETURNS

0	Success.
–2	Menu bar does not exist at this scope.
–3	Menu bars are unsupported or `scope` is out of range.

RELATED FUNCTIONS

`sm_d_menu, sm_r_menu`

r

EXAMPLE

```
#include "smdefs.h"
#include "smsoftk.h"

/* Close the current JAM window's menu. */

sm_c_menu( KS_FORM );
```

d_menu

load a menu bar that is stored in memory

SYNOPSIS

```
#include "smsoftk.h"

int sm_d_menu(char *menu, int scope);
```

PARAMETERS

`char *menu`

The address of a menu bar stored in memory.

`int scope`

Specifies when this menu is available to the application with one of these arguments:

```
KS_FORM
KS_APPLIC
KS_OVERRIDE
KS_MEMRES
KS_SYSTEM
```

DESCRIPTION

This function can load any menu bar that exists as a C data structure. Use the `bin2c` utility to create program data structures from disk–based menus. You can then compile these into your application and add them to the memory–resident screen list, as described in Chapter 9 of the JAM *Programmer's Guide.*

If a menu bar is already active at the specified scope, JAM compares its name to the value of *menu* and takes one of the following actions:

- If the menu names are the same, the routine returns immediately. Note that you can use this function to refresh the current menu bar display only if you first close it by calling `c_menu`.
- If the menu names are different and `scope` is KS_OVERRIDE, JAM pushes the currently active menu bar into the override stack and makes the newly read menu bar the current menu bar.
- If the menu names are different and `scope` is KS_MEMRES, JAM loads the menu bar along with other memory–resident menus that are loaded and available for use as external menus.

- If the menu names are different and scope is KS_SYSTEM, KS_APPLIC, or KS_FORM, JAM closes the previously loaded menu bar and frees the memory allocated for it, then loads the newly read menu bar. If the previous menu bar is displayed, JAM removes it at the next screen refresh.

RETURNS

0	Success.
–1	Not a menu bar.
–3	Menu bars are unsupported or scope is out of range.
–5	A malloc error occurred.

If the routine returns with an error, JAM retains the previous menu bar loaded at scope, if any.

For all errors except –3, a message is posted to the operator.

RELATED FUNCTIONS

sm_c_menu, sm_r_menu

EXAMPLE

```
#include "smdefs.h"
#include "smsoftk.h"
   ...
extern char customer_menu[];
   ...

/* Display the customer menu as the application-level menu.
 * Customer_menu was created using bin2c.
 */

sm_d_menu( customer_menu, KS_APPLIC );
```

mncrinit

initialize menu bar support

SYNOPSIS

```
void sm_menuinit();
```

DESCRIPTION

This routine is typically called automatically when you enable menu bars in your application. You enable menu bar support by setting MENUS to 1 in the main routine.

`sm_mmcrinit` sets a global variable to point to a control function. All screen manager functions that need menu bar support check the variable and, if it is non–zero, call indirectly with the request.

You should call this routine explicitly only if you are writing your own executive routine. You call `sm_mncrinit` in the main routine before the call to `sm_initcrt`.

RELATED FUNCTIONS

`sm_mn_forms, sm_mncrinit`

mn_forms

install menu bars in memory

SYNOPSIS

```
void sm_mn_forms();
```

DESCRIPTION

This routine is typically called automatically by JAM's executive. You should call this routine explicitly only if you write your own executive routine and want to load menu bars from memory. You must compile these menu bars into the application and add them to the memory-resident list, as described in Chapter 9 of the JAM *Programmer's Guide*. You can then load these menu bars by calling either `sm_d_menu` or `sm_r_menu`.

You call `sm_mn_forms` in the application's main routine. If you write your own custom executive, you must also call `sm_menuinit` to initialize menu bar support.

RELATED FUNCTIONS

`sm_menuinit, sm_d_menu, sm_r_menu`

mnadd

add an item to the end of a menu bar

SYNOPSIS

```
#include "smsoftk.h"
#include "smkeys.h"
#include "smmach.h"
#include "smmenu.h"

int sm_mnadd(int scope, char *menu_name, struct item_data
*menu_data);
```

PARAMETERS

`int scope`

Specifies when this menu is available to the application with one of these arguments:

```
KS_FORM
KS_APPLIC
KS_OVERRIDE
KS_MEMRES
KS_SYSTEM
```

`char *menu_name`

The name of the menu bar.

`struct item_data *menu_data`

A user-allocated structure that describes the appearance and function of a menu bar item. The description of `item_data` in Section 2.1 shows the values you can assign to this structure, and its default values.

DESCRIPTION

This routine adds an item at the end of the menu specified by `scope` and `menu_name`. The item gets the attributes that you supply to the `item_data` parameter. You assign attributes through identifiers that are defined in `smmenu.h`.

RETURNS

0	Success.
–2	No menu bar exists at this scope.

r

–3 Menu bars are unsupported or `scope` is out of range.
–4 `menu_name` is not found.
–6 Data in `item_data` is bad.
–7 A malloc error occurred.

RELATED FUNCTIONS

`sm_mnchange, sm_mndelete, sm_mnget, sm_mninsert, sm_mnitems, sm_mnnew`

EXAMPLE

```
#include "smdefs.h"
#include "smsoftk.h"
#include "smmach.h"
#include "smmenu.h"
#include "smkeys.h"

...

struct item_data *data;
data = ( struct item_data * ) malloc( sizeof( struct item_data )
);

/* Call sm_d_menu w/ a disk resident menu and KS_FORM.
 * Call sm_mnadd to add a title for submenu.
 */

sm_r_menu("mymenu.bin", KS_FORM);
data->type = MT_TITLE;
data->label = "Submenu";
data->accel = -1;
data->key = 0;
data->submenu = 0;
data->option = MO_INDICATOR_ON;
sm_mnadd(KS_FORM, "Submenu0", data);
free(data);

...
```

mnchange

change the text or display attributes of a menu item

SYNOPSIS

```
#include "smsoftk.h"
#include "smkeys.h"
#include "smmach.h"
#include "smmenu.h"

int sm_mnchange(int scope, char *menu_name, int item_no, struct
item_data *menu_data);
```

PARAMETERS

`int scope`

Specifies when this menu is available to the application with one of these arguments:

```
KS_FORM
KS_APPLIC
KS_OVERRIDE
KS_MEMRES
KS_SYSTEM
```

`char *menu_name`

The name of the menu bar.

`int item_no`

A positive integer that specifies the menu item to change, where the first menu item has a value of 0.

`struct item_data *menu_data`

A user-allocated structure that describes the appearance and function of a menu bar item. See "Menu Bar Data" on page 23 for more information on this structure and its values.

DESCRIPTION

Use this function to change a menu item's textual representation or display attributes. JAM modifies the contents of the menu item's data structure through the values that you supply for parameter *data*. For example, you can use this routine to gray out or check an item.

RETURNS

0	Success.
–2	No menu bar exists at this scope.
–3	Menu bars are unsupported or `scope` is out of range.
–4	`menu_name` is not found.
–6	Data in `item_data` is bad.
–7	A malloc error occurred.

RELATED FUNCTIONS

`mnadd, mndelete, mnget, mninsert, mnitems, mnnew`

EXAMPLE

```
#include "smdefs.h"
#include "smsoftk.h"
#include "smmach.h"
#include "smmenu.h"
#include "smkeys.h"

...

/* menu file stored in memory */
extern char mymenu[];

...

struct item_data *data;
data = ( struct item_data * ) malloc( sizeof( struct item_data )
);

/* Call sm_r_menu w/ a disk resident menu and KS_APPLIC.
 * Call sm_mnchange to gray out a menu item in the submenu.
 */
sm_r_menu("mymenu.bin", KS_APPLIC);
data->type = MT_KEY;
data->label = "NewItem";
data->accel = 3;
data->key = PF1;
data->submenu = 0;
data->option = MO_GRAYED|MO_SHOWKEY;
sm_mnchange(KS_APPLIC, "Submenu0", 0, data);
free(data);

...
```

mndelete

delete a menu bar item

SYNOPSIS

```
#include "smsoftk.h"
#include "smmach.h"
#include "smmenu.h"

int sm_mndelete(int scope, char *menu_name, int item_no);
```

PARAMETERS

`int scope`

Specifies when this menu is available to the application with one of these arguments:

```
KS_FORM
KS_APPLIC
KS_OVERRIDE
KS_MEMRES
KS_SYSTEM
```

`char *menu_name`

The name of the menu bar.

`int item_no`

A positive integer that specifies the menu item to delete, where the first menu item has a value of 0.

DESCRIPTION

This routine deletes the item specified by `item_no`, `menu_name`, and `scope` from the menu bar. The first item on a menu has an `item_no` value of zero.

RETURNS

0	Success.
–2	No menu bar exists at this scope.
–3	Menu bars are unsupported or `scope` is out of range.
–4	`menu_name` is not found.
–5	`item_no` is not found.

RELATED FUNCTIONS

`sm_mnadd, sm_mnchange, sm_mnget, sm_mninsert, sm_mnitems, sm_mnnew`

EXAMPLE

```
#include "smdefs.h"
#include "smsoftk.h"
#include "smmach.h"
#include "smmenu.h"

...

int count;

/*
  Delete the last item from the application menu
  called "customer"
*/

if ((count = mnitems( KS_APPLIC, "customer" )) > 0)
  sm_mndelete( KS_APPLIC, "customer", count );

...
```

mnget

get information about a menu bar item

SYNOPSIS

```
#include "smsoftk.h"
#include "smkeys.h"
#include "smmach.h"
#include "smmenu.h"

int sm_mnget(int scope, char *menu_name, int item_no, struct
item_data *menu_data);
```

PARAMETERS

`int scope`

Specifies when this menu is available to the application with one of these arguments:

```
KS_FORM
KS_APPLIC
KS_OVERRIDE
KS_MEMRES
KS_SYSTEM
```

`char *menu_name`

The name of the menu bar.

`int item_no`

A positive integer that specifies the menu item to get information on, where the first menu item has a value of 0.

`struct item_data *menu_data`

A user-allocated structure that describes the appearance and function of a menu item. See "Menu Bar Data" on page 23 for more information on this structure and its values.

DESCRIPTION

This function fills the fields in the `item_data` structure with the corresponding data of the menu item. Note that you must create buffers for the label and submenu elements of the structure that are large enough to hold the label and submenu names, as in the example shown later. The maximum length is 255 characters.

RETURNS

0	Success.
–2	No menu bar exists at this scope.

–3 Menu bars are unsupported or scope is out of range.
–4 menu_name is not found.
–5 item_no is not found.

RELATED FUNCTIONS

mnadd, mnchange, mndelete, mninsert, mnitems, mnnew

EXAMPLE

```
#include "smdefs.h"
#include "smmach.h"
#include "smmenu.h"
#include "smsoftk.h"

...

/* menu file stored in memory */
extern char mymenu[];

...

char buf1[100], buf2[100];

struct item_data *data;

data = (struct item_data *) malloc( sizeof( struct item_data ) );

data->label = buf1;
data->submenu = buf2;

/*  Call r_menu with a disk resident menu.
 *  Call mnget to get an override-level menu bar item.
 */

sm_r_menu("mymenu.bin", KS_OVERRIDE);
sm_mnget(KS_OVERRIDE, "Main", 0, data );
free(data);

...
```

mninsert

insert a new menu item

SYNOPSIS

```
#include "smsoftk.h"
#include "smkeys.h"
#include "smmach.h"
#include "smmenu.h"

int sm_mninsert (int scope, char *menu_name, int item_no, struct
item_data *menu_data);
```

PARAMETERS

`int scope`

Specifies when this menu is available to the application with one of these arguments:

```
KS_FORM
KS_APPLIC
KS_OVERRIDE
KS_MEMRES
KS_SYSTEM
```

`char *menu_name`

The name of the menu bar.

`int item_no`

A positive integer that specifies the menu item to follow the inserted item, where the first menu item has a value of 0.

`struct item_data *menu_data`

A user–allocated structure that describes the appearance and behavior of the menu item to insert. See "Menu Bar Data" on page 23 for more information on this structure and its values.

DESCRIPTION

This routine inserts a new menu bar item before the menu item specified by `item_no`, `menu_name`, and `scope`, using the data in the menu bar structure `item_data`.

RETURNS

0	Success.
–2	No menu bar exists at this scope.

–3 Menu bars are unsupported or `scope` is out of range.
–4 `menu_name` is not found.
–6 Data in `item_data` is bad.
–7 A malloc error occurred.

RELATED FUNCTIONS

mnadd, mnchange, mndelete, mnget, mnitems), mnnew

EXAMPLE

```
#include "smdefs.h"
#include "smsoftk.h"
#include "smmach.h"
#include "smmenu.h"
#include "smkeys.h"

...

struct item_data *data;

data = ( struct item_data * ) malloc( sizeof( struct item_data )
);

/* Call sm_r_menu w/ a disk resident menu and KS_FORM.
 * Call sm_mninsert to insert a submenu.
 */

sm_r_menu("mymenu.bin", KS_FORM);
data->type = MT_SUBMENU;
data->label = "NewItem";
data->accel = 3;
data->key = 0;
data->submenu = "Submenu1";
data->option = MO_INDICATOR;
sm_mninsert(KS_FORM, "Main", 1, data);
free(data);

...
```

mnitems

get the number of items on a menu bar

SYNOPSIS

```
#include "smsoftk.h"
#include "smmach.h"
#include "smmenu.h"

int sm_mnitems(int scope, char *menu_name);
```

PARAMETERS

`int scope`

Specifies when this menu is available to the application with one of these arguments:

```
KS_FORM
KS_APPLIC
KS_OVERRIDE
KS_MEMRES
KS_SYSTEM
```

`char *menu_name`

The name of the menu bar.

DESCRIPTION

This routine returns the number of items on the menu bar specified by `menu_name` and `scope`.

RETURNS

If successful, the function returns the number of items in the menu; otherwise, it returns one of these values:

–2	No menu bar exists at this scope.
–3	Menu bars are unsupported or `scope` is out of range.
–4	`menu_name` is not found.

RELATED FUNCTIONS

`sm_mnadd, sm_mnchange, sm_mndelete, sm_mnget, sm_mninsert, sm_mnnew`

EXAMPLE

```
#include "smdefs.h"
#include "smmach.h"

...

int ret;

/* Call sm_r_menu w/ a disk resident menu and KS_OVERRIDE.
 * Call sm_mnitems to get the number of items on the menu bar, and

 * place the number in the current field.
 */

sm_r_menu("mymenu.bin", KS_OVERRIDE);
ret = mnitems(KS_OVERRIDE, "Main");
   sm_n_itofield( "number", ret );

...
```

mnnew

create a menu

SYNOPSIS

```
#include "smsoftk.h"
#include "smmach.h"
#include "smmenu.h"

int sm_mnnew(int scope, char *menu_name);
```

PARAMETERS

`int scope`

The scope of the menu bar to create. Supply one of these values:

```
KS_FORM
KS_APPLIC
KS_OVERRIDE
KS_MEMRES
KS_SYSTEM
```

`char *menu_name`

The name of the menu bar.

DESCRIPTION

This routine creates a submenu in the menu bar structure at the specified scope level. After you call this routine, specify its contents by calls to `sm_mnadd` and `sm_mninsert`. After you create the menu and its contents, attach it to an existing menu by creating an item that invokes it, through a call to `sm_mnadd` or `sm_mninsert`

RETURNS

0	Success.
–2	No menu bar exists at this scope.
–3	Menu bars are unsupported or `scope` is out of range.
–4	`menu_name` is not found.
–7	A malloc error occurred.

RELATED FUNCTIONS

```
sm_mnadd, sm_mnchange, sm_mndelete, sm_mnget, sm_mninsert,
sm_mnitems;
```

EXAMPLE

```
#include "smdefs.h"
#include "smsoftk.h"
#include "smmach.h"
#include "smmenu.h"
#include "smkeys.h"

...

int ret;
struct item_data *data;

data = ( struct item_data * ) malloc( sizeof( struct item_data )
);

/* Call sm_r_menu w/ a disk resident menu and KS_OVERRIDE.
 * Call sm_mnnew to create a new menu bar .
 * Call sm_mnadd to add items to it and finally add this new menu
 * to the menu displayed as a submenu.
 */

sm_r_menu("main.bin", KS_OVERRIDE);
ret = sm_mnnew(KS_OVERRIDE, "NewItem");
   if ( ret == 0 )
   {
     data->type = MT_TITLE;
     data->label = "Submenu";
     data->accel = -1;
     data->key = 0;
     data->submenu = 0;
     data->option = MO_INDICATOR_ON;

     sm_mnadd(KS_OVERRIDE, "NewItem", data);

     data->type = MT_SUBMENU;
     data->label = "I";
     data->accel = 0;
     data->key = 0;
     data->submenu = "Submenu1";
     data->option = MO_INDICATOR;

    sm_mnadd(KS_OVERRIDE, "NewItem", data);

     data->type = MT_SUBMENU;
     data->label = "NewItem";
     data->accel = 3;
```

```
      data->key = 0;
      data->submenu = "NewItem";
      data->option = MO_INDICATOR;

      sm_mnadd(KS_OVERRIDE, "Main", data);
    }
free(data);
...
```

r_menu

read a menu bar from memory, a library or disk

SYNOPSIS

```
#include "smsoftk.h"
#include "smmach.h"
#include "smmenu.h"

int sm_r_menu(char *menu_name, int scope);
```

PARAMETERS

`char *menu_name`

The name of the menu bar to read.

`int scope`

Specifies when this menu is available to the application with one of these arguments:

```
KS_FORM
KS_APPLIC
KS_OVERRIDE
KS_MEMRES
KS_SYSTEM
```

DESCRIPTION

When you call this routine, JAM first looks for the specified menu bar in the memory–resident screen list, next in any open libraries, and finally on disk in the directories specified by the argument to `sm_initcrt` and by `SMPATH`.

If a menu bar is already active at the specified scope, JAM compares its name to the value of *menu_name* and takes one of the following actions:

- If the menu names are the same, the routine returns immediately. Note that you can use this function to refresh the current menu bar display only if you first close it by calling `sm_c_menu`.
- If the menu names are different and `scope` is KS_OVERRIDE, JAM pushes the currently active menu bar into the override stack and makes the newly read menu bar the current menu bar.
- If the menu names are different and `scope` is KS_MEMRES, JAM loads the menu bar along with other memory–resident menus that are loaded and available for use as external menus.

- If the menu names are different and `scope` is KS_SYSTEM, KS_APPLIC, or KS_FORM, JAM closes the previously loaded menu bar and frees the memory allocated for it, then loads the newly read menu bar. If the previous menu bar is displayed, JAM removes it at the next screen refresh.

RETURNS

0	Success.
–1	Not a menu bar.
–2	No menu bar exists at this scope.
–3	Menu bars are unsupported or `scope` is out of range.
–4	`menu_name` is not found.
–5	A malloc error occurred,

In the case of an error the previously displayed menu bar remains displayed.

For all errors except –3 a message is posted to the operator.

RELATED FUNCTIONS

`sm_c_menu, sm_d_menu`

EXAMPLE

```
#include "smdefs.h"
#include "smsoftk.h"
#include "smmach.h"
#include "smmenu.h"

...

/* Read in the company menu and display it at the form level. */

sm_r_menu( "company.bin", KS_FORM );

...
```

3 Menu Bar Utilities

JAM has two utilities for creating menu bars:

- `menu2bin` converts an ASCII menu script into a binary menu file.
- `kset2mnu` converts a JAM keyset into an ASCII menu script. For detailed instructions on creating menu bar scripts, see "Menu Bars" earlier in this addendum.

The following sections describe these utilities in detail.

menu2bin

convert ASCII menu scripts to binary format

SYNOPSIS

`menu2bin [-pv] [-e` *ext*`]` *menufile...*

OPTIONS

`-p`	Places the binary files in the same directories as the input files.
`-v`	Lists the name of each input file as it is processed.
`-e`	Appends *ext* to the output file name. The default extension is `bin`.

DESCRIPTION

The `menu2bin` utility converts ASCII menu scripts into binary format. Menu scripts are created as text files. Chapter 1, "Menu Bars", shows how to write a menu script.

Menu binary files can be placed in libraries with the `formlib` utility. Refer to the JAM *Utilities Guide* for more information.

ERRORS

`Too many menu definitions. Max is 128.`
Only 128 menu definitions may be included in one menu script.

`Too many item definitions. Max is 128.`
Only 128 item specifications may be included in one menu definition.

`Cannot create '%s'`
`Error writing '%s'`
An output file could not be created, due to lack of permission or perhaps lack of disk space. Correct the file system problem and retry the operation.

`Neither '%s' nor '%s' found.`
An input file was missing or unreadable. Check the spelling, presence and permissions of the file in question.

`Error in '%s' line '%d'` *error–type*
The syntax of your script on the specified line is incorrect. The value of *error–type* specifies one of these errors:

```
Expected left brace '{' after menu name.
No right brace '}' found before EOF.
No menu name specified.
Expected quoted item label.
Missing action.
```

```
Unknown action '%s'.
Unknown option '%s'.
No key specified.
Bad key '%s'.
Bad escape sequence '%s'.
Undefined submenu '%s'.
More than one option of this type (%s).
More than one accelerator character assigned.
Accelerator character at end of string - Ignored.
Menu '%s' is on menu bar so cannot be used as submenu.
```

kset2mnu

convert keysets into ASCII menu scripts.

SYNOPSIS

`kset2mnu [-pv] [-e` *`ext`*`]` *`keyset`*`...`

OPTIONS

`-p`	Places the binary files in the same directories as the input files.
`-v`	Lists the name of each input file as it is processed.
`-e`	Appends *ext* to the output file name. The default extension is `mnu`.

DESCRIPTION

The `kset2mnu` utility converts keysets into menu scripts and stores it in an ASCII text file. The utility converts a keyset according to these rules:

- The first row in the keyset becomes the menu bar.
- Subsequent rows become submenus. Submenus are named "Row*x*", where *x* is the row number.
- The `SFT`*x* key (goto row *x*) becomes an entry for the submenu named Row*x*.
- The `SFTN` (next row) and `SFTP` (previous row) keys become entries for the submenus named Row*(i+1)* or Row*(i–1)*, where *i* is the current row.

Because menu bars and keysets are often organized according to different principles, the converted menu bar often requires manual editing. For example, keyset items in the first row typically invoke actions, while menu bar items usually invoke pulldowns whose items invoke actions.

When you finish editing the menu bar script, convert it to binary format with `menu2bin` and attach it to the application.

ERRORS

`Soft key '%s' designates a nonexistent submenu.`
The keyset contains a `SFT`*n* key for a row that does not exist. Remove the offending key from the keyset and reconvert it.

`Neither '%s' nor '%s' found.`
An input file was missing or unreadable. Check the spelling, presence, and permissions of the input file.

```
Cannot create '%s'
Error writing '%s'
```

An output file could not be created, due to lack of permission or disk space. Correct the file system problem and retry the operation.

4 Display Emphasis

JAM now has two display options that let you emphasize the current, or active, screen:

- *Drop shadows* appear to cast a shadow from the active screen shade over underlying screens.
- *Graying* changes the display attributes of all screens except the active one according to a predefined algorithm—for example, highlights turn off and colors change to monochrome.

An application can use both methods singly or together.

Drop shadows and graying change only the display attributes of the background screens; the actual contents are unaffected. You can specify which display attributes to preserve and which new attributes to use for the grayed data.

To use display emphasis, set JAM as follows:

- Specify the emphasis style to use—drop shadows or graying.
- Specify the display attributes of a grayed object.

The following sections show how to perform both tasks.

4.1 SPECIFYING EMPHASIS STYLE

You specify which emphasis style to use by setting the value of the configuration variable `EMPHASIS`. You can set this value in the configuration file as follows:

```
EMPHASIS=style
```

You can also reset the emphasis style at runtime through the library function `sm_option`:

```
sm_option(EMPHASIS, style);
```

Note that after `sm_option` changes the emphasis style, you must call `sm_rescreen` to repaint the display.

You can specify one of the following values for `style`:

- `NONE` disables display emphasis.

- GRAYBKGD grays background screens. Only the active screen retains its original display attributes.
- DROPSHADOW draws a shadow at the topmost screen's right and bottom edges. The drop shadow is two columns wide and one line deep. The right shadow starts one space below the screen's top edge, while the bottom shadow starts two columns from the screen's left edge. The bottom shadow is indented two spaces from the left edge of the screen. The shadow is formed by graying the underlying text.

4.2 SETTING GRAY ATTRIBUTES

Whether you use graying or drop shadows, you must set the display attributes that JAM uses for the underlying objects. You set these through two video file variables:

- EMPHASIS_KEEPATT specifies the attributes that a grayed object retains. This variable intially has all attributes enabled except HILIGHT.
- EMPHASIS_SETATT specifies the attributes that the grayed object acquires. This variable intially has two attributes enabled: REVERSE and DIM.

You can reset EMPHASIS_KEEPATT and EMPHASIS_SETATT either in the video file, or through the runtime function sm_pset. For example, the following statement sets graying with attributes DIM and WHITE:

```
sm_pset(V_EMPHASIS_SETATT, "DIM WHITE");
```

After you call sm_pset, call sm_rescreen to update the display.

See section 4.6 in the JAM *Configuration Guide* for display attribute names.

5 Remote Scrolling

You can now configure JAM applications to allow or disallow scrolling data inside an array when the cursor is positioned outside that array. This is particularly useful for character-mode applications in which users need to view off-screen data in arrays that are tab-protected.

You enable or disable remote scrolling by assigning one of these values to the setup variable SCR_KEY_OPT:

- SCR_NEAREST, the default, enables remote scrolling. This causes the nearest scrollable array to scroll when the user presses a scrolling key.
- SCR_CURRENT allows users to scroll array data only when the cursor is in that array. Scrolling keys are inactive when the cursor is outside a scrollable array.

INDEX