MacApp 2.0 Globals

The MacApp unit defines a number of global constants, variables, types, procedures, and functions. The globals documented here are described for your reference, but you will probably not use many of them directly. MacApp methods use the globals, while the application code very rarely, if ever.

Constants

This section documents the constants defined as part of the MacApp package. Although the values of the constants are given here for your information, those values are subject to change. (In some cases, when the values are very likely to change, they are not given here.) Normally, you should simply use the constant identifier and not concern yourself with its value.

The constants are categorized according to purpose.

Copyright constant

```
kCopyright = 'Copyright
  1984, 1985, 1986,
  1987, 1988
  Apple Computer Inc.';
```

Used to store the copyright notice for MacApp.

Menu constants

`kMBarDisplayed = 128;` Identifies the menu bar resource that holds the menus that are initially displayed.

`kMBarNotDisplayed = 129;` Identifies the menu bar resource that holds menus that are not initially displayed. These menus include buzzword menus and menus that may be displayed later.

`kMBarHierarchical` Identifies menu bar resource that holds menus that are submenus or pop-up menus. These menus will be installed when the application is initialized.

The following constants identify the standard menus shared by all Macintosh applications.

`mApple = 1;` Identifies the Apple menu, the leftmost menu in the menu bar.

`mFile = 2;` Identifies the File menu.

`mEdit = 3;` Identifies the Edit menu.

`mLastMenu = 63;` Identifies the last menu managed by MacApp's DoSetupMenus methods. This commands in menus above this number are never unchecked or disabled by MacApp.

`mDebug = 900;` Identifies the Debug menu.

Command numbers

The command numbers listed here are passed to your methods, generally to gTarget.DoMenuCommand. Most are passed as a result of the user picking a menu command; command numbers are also used for other types of commands, such as typing or mouse commands. When the description says "MacApp catches this," that means that the MacApp DoMenuCommand methods will handle those command numbers, often by calling application methods.

`cNoCommand` Command number representing no command. MacApp catches this.

`cAboutApp` Identifies the About <AppNme>... command. MacApp catches this.

File menu commands

`cNew = 10;` Identifies the New command. MacApp catches this. (See also `cNewLast`.)

`cNewLast = 19;` Identifies the last New command. MacApp provides a range of New commands for applications that have different document types, and cNewLast identifies the end of the range. If you enable these commands, MacApp handles them.

`cSave = 30;` Identifies the Save command. MacApp catches this.

`cClose = 31;` Identifies the Close command. MacApp catches this.

`cSaveAs = 32;` Identifies the Save As command. MacApp catches this.

`cSaveCopy = 33;` Identifies the Save a Copy In command. MacApp catches this.

`cRevert = 34;` Identifies the Revert command. MacApp catches this.

`cOpen = 20;` Identifies the Open command. MacApp catches this. See also cOpenLast.

`cOpenLast = 29;` Identifies the last Open command. MacApp provides a range of Open commands for applications that have different document types, and cOpenLast identifies the end of the range. If you enable these commands, MacApp handles them.

`cPageSetup = 176;` Identifies the Page Setup command. MacApp catches this.

`cPrintOne = 177;` Identifies the Print One command. MacApp catches this.

`cPrint = 178;` Identifies the Print command. MacApp catches this.

`cPrintToFile = 179;` Identifies the Print to File command. MacApp catches this.

`cPrFileBase = 176;`
`cPrFileMax = 195;` Command numbers between these two bounds are sent to a document's fDocPrintHandler even if it is not in the target chain.

`cPrViewBase = 201;`
`cPrViewMax = 250;` Command numbers in this range are printing commands applied to a displayed view that is in the target chain.

`cQuit = 36;` Identifies the Quit command. MacApp catches this.

Edit menu commands

With the following commands, `cEditBase` is subtracted from the command number to arrive at the appropriate number to pass to SystemEdit. This relationship is enforced in TApplication.IApplication.

`cEditBase = 101;`	The number that is the start of standard editing commands.
`cUndo = 101;`	Identifies the Undo and Redo commands. (Those are a single menu command; which one is displayed depends on the phase of the current command.) MacApp catches this.
`cEditSep = 102;`	Identifies the line separating the Undo (or Redo) command from the Cut command in the menu.
`cCut = 103;`	Identifies the Cut command. UTEView catches this. Applications may also.
`cCopy = 104;`	Identifies the Copy command. UTEView catches this. Applications may also.
`cPaste = 105;`	Identifies the Paste command. UTEView catches this. Applications may also.
`cClear = 106;`	Identifies the Clear command. UTEView catches this. Applications may also.
`cEditLast = cClear;`	Marks the last command in the Edit menu.
`cShowClipboard = 35;`	Identifies the Show/Hide Clipboard command. MacApp catches this.

Finder pseudocommands

`cFinderNew = 40;`	Given when the user creates a new document from the Finder, generally by opening the application icon. MacApp catches this.
`cFinderPrint = 41;`	Given when the user prints a document from the Finder. MacApp catches this.
`cFinderOpen = 42;`	Given when the user opens an existing document from the Finder, generally by opening the document's icon. MacApp catches this.

Other command numbers

`cTyping = 120;`	For use in a typing command. Note that this is not a menu command, but is used in a buzzword menu to refer to a string for Undo or Redo.
`kNoItemNumber = -1;`	A UDialog constant representing "no item" in contexts where a dialog item number parameter is possible but none is present. Examples of its use are as the value of fItemNumber for a TRadioCluster object, which does not have a corresponding item in the actual dialog's item list, and as the value of a dialog view's fDfltButton if the dialog has no default button.

Alert constants

These constants are resource ID's that refer to items in the application's resource file. The messages given are the ones these constants are intended for; if you include the standard MacApp resources, you will get these messages. Nothing is done to enforce that correspondence, however.

The following standard should be observed:

- Alerts 0 through 249 are reserved for UMacApp.
- Alerts 250 through 299 are reserved for UPrinting.
- Alerts 300 through 349 are reserved for UDialog.
- Alerts 350 through 399 are reserved for UTEView.
- Alerts 400 through 449 are reserved for UAppleTalk.
- Alerts 450 through 499 are reserved for future use.
- Alerts 1000 and up are available for applications.

See the "Failure Handling" recipe in the Cookbook for details on how these alerts are used.

`phGenError = 100`	"Could not ^2, because ^0. ^1"
`phCmdErr = 101;`	"Could not complete the '^2' command because ^0. ^1."
`phUnknownErr = 102;`	"Could not complete your request because ^0. ^1." Used if no operation string is specified.
`phSaveChanges = 110;`	"Save changes before closing?"
`phRevert = 111;`	"Revert to last version saved?"
`phFileChanged = 112;`	"File changed since last save."
`phPurgeOld = 113;`	"OK to purge old version before saving new one?"
`phReopenDoc = 114;`	"Document is already open."
`phSpaceIsLow = 115;`	"Memory space is low."
`phUnsupportedROMs = 116;`	"Can't start application because ROMs are too old."
`phTooManyChars = 150;`	"Too Many Characters." Used to reject a Paste command or keystrokes.
`phAboutApp = 201`	"About <AppName>..."
`errReasonID = 128`	Identifies the table of strings that describe error messages.
`errRecoveryID = 129`	Identifies the table of strings giving recovery information corresponding to the error messages.
`errOperationsID = 130`	Identifies the table of operation strings for use in building error messages.
`errAppTable = 1000;`	Added to the MacApp resource id to get the id of an application error table.
`msgCmdErr = 0;`	Used in a Failure message parameter to indicate that the low word of the message is a command number. You use this by adding it to the command number (which must be greater than zero) to get the message value.
`msgAlert = $FFFF0000;`	Used in a Failure message parameter to indicate that the low word of the message is an alert resource number. You use this by adding it to the alert resource number (which must be greater than zero) to get the message value.

`msgLookup = $FFFE0000;` Used in a Failure message parameter to indicate that the low word of the message is an index number for the table errOperationsID. You use this by adding it to the index number (which must be greater than zero) to get the message value.

`msgStrList = 200 * $10000;` Identifies the standard list of operation strings used by MacApp. See UMacApp.p for details.

Resource ID's

These constants are used to identify resources in the application's resource file.

`kIDMNTBbyCmdNumber = 0;` Resource ID of the menu command number table.

`kIDClipWindow = 200;` Resource ID of the window displaying the Clipboard.

Highlighting constants

`hlOff = 1;` Indicates the selection is to have highlighting removed.

`hlDim = 2;` Indicates the selection is to have dim highlighting.

`hlOn = 4;` Indicates the selection is to have full highlighting.

```
hlOffDim=hlOff+hlDim;
hlDimOff = hlOffDim;
hlDimOn = hlDim + hlOn;
hlOnDim = hlDimOn;
hlOffOn = hlOff + hlOn;
hlOnOff = hlOffOn;
```

These constants can be used to test for combinations of fromHl + toHL, when you do not care which is the old state and which is the new state.

Miscellaneous constants

`kMakingCopy = TRUE;` Used in a number of calls to TDocument saving methods to indicate that a new copy of the file is being made.

`kDataOpen = TRUE;` Used in call to TDocument.IDocument to indicate that the data fork of the document file should be kept open all the time.

`kPrintInfoSize = 120;` Size, in bytes, of the PrintInfo record.

`kRsrcOpen = TRUE;` Used in call to TDocument.IDocument to indicate that the resource fork of the document file should be kept open all the time.

`kStdScrLimit = 300;` Defines the default value of fScrollLimit, used in TFrame.IFrame.

`kStdScroll = 16;` Defines the default value of fScrollUnit, used in TFrame.IFrame.

`kStdSzSBar = 16;` Defines the width and height of standard vertical and horizontal scroll bars.

```
kStdSzMinus1SBar =
  kStdSzSBar - 1;
```

One pixel less than the size of a standard scroll bar.

`kUsesDataFork = TRUE;` Used in a call to TDocument.IDocument to indicate that the application uses the data fork of the document file.

`kUsesRsrcFork = TRUE;` Used in a call to TDocument.IDocument to indicate that the application uses the resource fork of the document file.

`kWatchDelay = 3 * 60;`	Defines the default number of clock ticks before cursor changes to a watch (approximately three seconds).
`kNotInFrame = inDesk;`	Returned by FindFrame if you pass it a point that is not in any frame.
`kForDisplay = FALSE;`	Tells DoMakeViews whether the views are being created for the display as well as (possibly) for printing.
`kForPrinting = NOT kForDisplay;`	Used to tell DoMakeViews whether views are being created only for printing (usually for Finder printing).
`kWantHScrollBar = TRUE;`	Used with the TFrame.IFrame method to display a horizontal scroll bar.
`kWantVScrollBar = TRUE;`	Used with the TFrame.IFrame method to display a vertical scroll bar.
`kHFrResize = TRUE;`	Used with the TFrame.IFrame method to allow horizontal resizing of the frame.
`kVFrResize = TRUE;`	Used with the TFrame.IFrame method to allow vertical resizing of the frame.
`kDialogWindow = TRUE;`	Passed to a number of window methods and routines to indicate that the window being defined is a dialog box.
`kMakingCopy = TRUE;`	Used with TDocument.Save.
`kAskForFilename = TRUE;`	Used with TDocument.Save so the user will be asked for a filename (usually used with the Save As command).
`kLeftPalette = h;`	Used with NewPaletteWindow to place the palette or status frame on the left of the window.
`kTopPalette = v;`	Used with NewPaletteWindow to place the palette or status frame at the top of the window.
`kVisible = TRUE;`	Used by some methods to indicate that an operation is visible.
`kInvisible = FALSE;`	Used by some methods to indicate that an operation is invisible.
`kRedraw = TRUE;`	Used by some methods to indicate that an operation should redraw a view or the affected part of the view
`kDontRedraw = FALSE;`	Used by some methods to indicate that an operation should not redraw a view or the affected part of the view
`gExperimenting = FALSE;`	Redeclared as global variables when qDebug is TRUE.
`gDebugPrinting = FALSE;`	
`gReportMenuChoices = FALSE;`	
`gReportEvt = FALSE;`	
`gIntenseDebugging = FALSE;`	
`gUnloadAllSegs = TRUE;`	
`gMemMgtReport = FALSE;`	

Types

This section documents the data and object types defined as part of MacApp and the units shipped along with MacApp: UTEView, UPrinting, and UDialog. These types are used for some of the global variables described in the following section, to declare parameters of many methods and global routines, and occasionally to declare variables in applications.

The types are categorized according to purpose.

Event types

There are two types defined in MacApp for events. The first, PEventRecord, is a pointer to an EventRecord, the type defined for the Event Manager that is used to pass events to programs. The second type, EventInfo, is an expanded event record which, as well as including a pointer to the original event record, has fields that make the information contained in the event record more accessible.

```
PEventRecord = ^EventRecord;

EventInfo = RECORD
  thePEvent: PEventRecord;          Pointer to the event used to derive the rest of
                                    the fields.
  theBtnState: BOOLEAN;             The state of the mouse button.
  theCmdKey: BOOLEAN;               The state of the Command key.
  theShiftKey: BOOLEAN;             The state of the Shift key.
  theAlphaLock: BOOLEAN;            The state of the Caps Lock key.
  theOptionKey: BOOLEAN;            The state of the Option key.
  theControlKey: BOOLEAN;           The state of the Control key.
  theAutoKey: BOOLEAN;              TRUE if this was an auto-key event, issued as
                                    the result of a repeating key.
  theClickCount: INTEGER;           Indicates the number of mouse clicks. Zero
                                    indicates the event was not a mouse down;
                                    values greater than zero indicate a number of clicks.
END;
```

Phase types

Several operations in MacApp have distinct phases: when the operation begins, as it continues, and when it is ending. The types described here are used to indicate the phases for the idle part of the event loop and for tracking the mouse.

```
IdlePhase = (idleBegin, idleContinue, idleEnd);
```

The IdlePhase type defines the phase of the idle loop. IdleBegin is the phase when the idle loop first begins. IdleContinue is in effect until something happens to end the idle state. At that point, the idle phase becomes idleEnd.

```
TrackPhase = (trackPress, trackMove, trackRelease);
```

The TrackPhase type defines phase of mouse movement when the mouse button is down. TrackPress is the phase when the mouse button first goes down. TrackMove is when the button continues to be down. TrackRelease is the phase when the button comes up.

Command types

```
CmdNumber = INTEGER;
```
Holds MacApp command numbers.

View coordinate types

The SizeDeterminer type tells how a view's size is to be determined. The size is specified separately in each dimension. The possible values of each are explained below. The ImageSpace and PageAreas types are used by TPrintHandler. PageAreas holds the parameters of a page.

```
SizeDeterminer =
```

`(sizeSuperView,`	View's width or height is exactly the same as its superview. When its superview changes size, the view's size changes as well.
`sizeRelSuperView`	View's width or height changes relative to the size of its superview's size. When the superview's size changes, the view's size changes an equal amount.
`sizePage,`	View to be the size of one page.
`sizeFillPages,`	View to grow upward to fill an exact number of pages.
`sizeVariable,`	View size fluctuates according to application-specific criteria.
`sizeFixed)`	No special default handling of size issues.

```
ImageSpace = (viewSpace, padSpace);
```

```
PageAreas = RECORD
```

`thePaper: Rect;`	The size of the physical page.
`theInk: Rect;`	The size of the printable page.
`theMargins: Rect;`	The margins of the page. The top and left are positive values; the bottom and right are negative values.
`theInterior: Rect;`	The rectangle into which the view subset will be projected.

```
END;
```

Highlighting type

`HLState = hlOff..hlOn;`	The hlState type defines the possible highlighting states.

Miscellaneous types

```
PAppFile = ^AppFile;
```

This defines the pointer type AppFile, which is discussed in the Package Manager chapter of Inside Macintosh. It is used as a parameter to TApplication.KindOfDocument.

```
SIPChoice = (sipNever, sipAlways, sipAskUser);
```

SIP stands for Save In Place. This type is used for a value that determines what happens when there isn't room on the disk to save a document in a new file, rather than writing over the old version of the document. (When the old version is overwritten, the file is "saved in place.")

Global variables

The global variables defined as part of MacApp and the other units can be used like fields of the application object. They hold information pertinent to the application as a whole and not specific to a particular document, view, or window.

In general, you do not change the values of these variables directly, although you may check some values, and MacApp methods you call may result in a change to a global variable.

The variables you may change directly are so indicated; do not alter the values of any other global variables yourself.

`gAppDone: BOOLEAN;`	Indicates whether your application wants to terminate. Initialized to FALSE. MacApp sets this to TRUE when the user issues a Quit command. You may change this value if you want your application to terminate in other circumstances.
`gApplication: TApplication;`	The application object.
`gChooserOK: BOOLEAN;`	Controls whether the user is allowed to change the printer with the Chooser desk accessory. Ordinarily set to TRUE in InitToolBox. Change it to FALSE if you don't want the user to be able to change the printer while your application is running. The right time to set this to FALSE, if you want it FALSE, is after calling InitToolBox and before calling InitPrinting (where, if it's found to be FALSE, the low-memory location governing this option is poked).
`gClipOrphanage: TView;`	A view to represent the Clipboard when it can't otherwise be displayed.
`gClipWindow: TWindow;`	The window holding the Clipboard display.
`gClipShowing: BOOLEAN;`	Indicates whether the Clipboard window is currently showing.
`gClipView: TView;`	The view currently installed in the Clipboard.
`gClipUndoView: TView;`	The view previously installed in the Clipboard.
`gClipWrittenToDeskScrap: BOOLEAN`	Indicates whether the current clipboard has been written to the desk scrap, in which case it is set to TRUE.
`gCmdTable: CmdTable;`	Maps command numbers to the menu and item ID's used by the Menu Manager.
`gCopyright: StringHandle;`	The copyright notice.
`gCouldPrint: BQOLEAN;`	Indicates whether printer code is accessible to the application.
`gCursorInfo: CursorInfo;`	Information about the state of the cursor. This is defined in UBusyCurosr.
`gDocument: TDocument;`	The current document. If the application has multiple documents, this is the document belonging to the last window that was activated.

`gDocList: TList;`	The application's list of documents.
`gDrawingPictScrap: BOOLEAN;`	Indicates whether or not a view's Draw method is being called in order to create PICT data for the desk scrap. Your view's Draw method can check this value and insert picture comments as appropriate if you want to have them in the picture stored in the desk scrap.
`gFileCount: INTEGER;`	The number of files to open or print from the Finder. This is set in Init2.
`gFinderPrinting: BOOLEAN;`	TRUE if the Finder started the application just for printing documents.
`gFocusedView: TView;`	The view that is currently focused.
`gFreeWindowList: TList;`	The list of windows that do not have documents.
`gFrontWindow: TWindow;`	Contains NIL or the window object of the frontmost window.
`gHeadCohandler: TEvtHandler;`	The head of the linked list of global cohandlers.
`gHFSInstalled: BOOLEAN;`	Indicates whether or not the Hierarchical File System (HFS) is installed. When this is TRUE and gROM128K is FALSE, HFS is in RAM. MacApp sets this value.
`gGotClipType: BOOLEAN;`	Indicates whether or not the Clipboard has data of a type that the current target can paste.
`gIdlePhase: IdlePhase;`	Stores the current idle phase. The value is idleBegin, idleContinue, or idleEnd.
`gInBackground: BOOLEAN`	TRUE if the application is in the background on a system with MultiFinder™.
`gLastCommand: TCommand;`	The command object for the last command done or undone by the user. May be NIL if there were no commands or if the last command cannot be undone.
`gLastDeskAcc: LONGINT;`	The time of the most recent possible invocation of a desk accessory.
`gLastMsePt: Point;`	The coordinates of the mouse pointer in the last event passed to TApplication.CountClicks.
`gLastUpTime: LONGINT;`	The time of the last mouse-up event passed to TApplication.ObeyEvent.
`gMBarDisplayed: INTEGER;`	Identifies the menu bar resource ('MBAR') that holds the menus that are initially displayed. This is initialized in InitToolbox to kMBarDisplayed but can be changed to a different value before calling IApplication.
`gMBarNotDisplayed: INTEGER;`	Identifies the MBAR resource ('MBAR') that holds the menus that are not initially displayed. These menus include buzzword menus that will be displayed later. It is initialized to kMBarNotDisplayed but can be changed before calling IApplication.
`gMBarHierarchical: INTEGER;`	Identifies the menu bar resource ('MBAR') that holds the hierarchical submenus. It is initialized in InitToolbox to kMBarHierarchical but can be changed before calling IApplication.

`gMainFileType: OSType;`	The principal file type used by documents of the application, set in TApplication.IApplication. By default, TApplication.SFGetParms returns a list that contains only this.
`gMenusAreSetup: BOOLEAN;`	Indicates whether or not the menus are properly set up. It is set to FALSE after every event and to TRUE by SetupTheMenus, which is called at idleBegin.
`gNoChanges: TCommand;`	A special TCommand object created by MacApp that you can return if the command does not change the document or no command results from an operation. (Note that this is not a real object; you cannot refer to its fields or make method calls using gNoChanges.) You should carry out the command in the DoMenuCommand procedure.
`gNullPrintHandler: TPrintHandler;`	A print handler to handle printing-related messages for views that don't print.
`gOrthogonal: ARRAY[VHSelect] OF VHSelect;`	Used to convert VHSelect values. gOrthogonal[v] = h; gOrthogonal[h] = v.
`gRedrawMenuBar: BOOLEAN;`	If true, then the menu bar will be redrawn by TApplication.SetupTheMenus. If you have menus that are not handled by MacApp then you may need to set this variable to TRUE to force the menu bar to be redrawn.
`gStdHysteresis: Point;`	The standard hysteresis value passed to TView.DoMouseCommand.
`gStdWMoveBounds: Rect;`	The standard boundsRect (of the screen) to pass to the DragWindow Toolbox routine.
`gStdWSizeRect: Rect;`	The standard sizeRect to pass to the GrowWindow Toolbox routine.
`gSysWindowActive: BOOLEAN;`	Indicates whether or not the front window is a system window.
`gTarget: TEvtHandler;`	The event handler that gets the first chance at DoCommand, DoSetupMenus, DoKeyCommand, and DoIdle. You can set this although few applications do set it directly. (See the discussion of the target chain under "The Target Chain and the Cohandler Chain" in Chaper 2.) This is never set to NIL.
`gVarClipPicSize: BOOLEAN;`	Indicates whether or not pictures in the Clipboard should be treated as variable size, depending on the window size. If FALSE (the default), then pictures in the Clipboard are drawn actual size.
`gWorkPort: GrafPtr;`	A grafPort that MacApp uses for temporary purposes. You can use it as well, but only for short periods of time and not across calls to MacApp's routines or methods. Don't make any assumptions about its state.
`gZeroRect: Rect;`	A rectangle all of whose coordinates are 0, obtained with `SetRect(gZeroRect, 0, 0, 0, 0).`
`gZeroVPt: VPoint;`	A VPoint whose coordinates are zero.
`gZeroVREct: VRect;`	A VRect whose coordinates are zero.

Global routines

Most of these routines can be called by your application. A few are called by many or most applications. Those that cannot be called are so indicated.

Window-creation routines

```
FUNCTION NewSimpleWindow (itsRsrcID: INTEGER; itsDocument: TDocument; wantHScrollBar,
                    wantVScrollBar: BOOLEAN;itsView: TView): TWindow;
```

A utility for creating simple windows that contain one view and may or may not scroll, depending on the values of wantHScrollBar and wantVScrollBar. Signals Failure if the window could not be created. This is often called by applications. Note that you usually do not call this if you call NewPaletteWindow.

```
FUNCTION NewPaletteWindow(itsRsrcID: INTEGER; itsDocument: TDocument; wantHScrollBar,
                    wantVScrollBar: BOOLEAN; itsMainView: TView; itsPaletteView:
                    TView;sizePalette: INTEGER; whichWay: VHSelect): TWindow;
```

A utility for creating MacDraw-like windows with a nonscrolling palette along the left edge (if whichWay is kLeftPallete) or a nonscrolling status area at the top of the window (if whichWay is kTopPalette) and a main view that may or may not scroll, depending on the values of wantHScrollBar (scrolls if wantHScollBar is kWantHScrollBar) and wantVScrollBar (scrolls if wantVScrollBar is kWantVScrollBar). (Precede those constants with NOT if you don't want scrolling.) Signals Failure if the window could not be created.

```
FUNCTION NewTemplateWindow(viewRsrcId: INTEGER; itsDocument: TDocument): TWindow;
```

The preferred utility for creating a window and its views from a 'view' resource. See the Cookbook for examples of its use.

Command-related and menu-related routines

The following routines all recognize normal command numbers and command numbers of the form –(256 * menu + item) and –(256 * menu). The former is used when there is no command number; the latter to enable or disable a whole menu (which is rarely done).

```
PROCEDURE CmdToMenuItem(aCmd: CmdNumber; VAR menu, item: INTEGER);
```

Given a command number, finds the menu ID and item number of the command. If aCmd is not in the command table, this returns 0 as the menu number and –aCmd as the item number.

```
FUNCTION CmdFromMenuItem(menu, item: INTEGER): CmdNumber;
```

Given a menu ID and item number, returns the command number of the command. If item < 0, this returns –item. If the item is not in the command table, this returns –(256 * menu + item).

```
PROCEDURE CmdToName(aCmd: CmdNumber; VAR menuText: STR255);
```

Given a command number, returns the text of the command.

```
PROCEDURE Enable(aCmd: CmdNumber; canDo: BOOLEAN);
```

Enables or disables a menu item, depending on the value of canDo. This is called by almost every application, because it must be called from DoSetupMenus for every application-specific menu item that should be enabled.

```
PROCEDURE EnableCheck(aCmd: CmdNumber; canDo: BOOLEAN; checkIt:BOOLEAN);
```

Enables or disables a menu item and places or removes a check mark next to the item. Many applications call this routine, always from DoSetupMenus.

```
PROCEDURE SetStyle(aCmd: CmdNumber; aStyle: Style);
```

Sets the type style for a menu item. This is called from DoSetupMenus, usually only for the font style items.

```
PROCEDURE GetResMenu(menuResID: INTEGER);
```

Calls the Resource Manager routine GetResource('MENU', menuResID). You should use this routine when you are not sure whether the menu is actually loaded in the menu bar. (You cannot call the Menu Manager more than once for a given menu.)

```
PROCEDURE SetCmdIcon(aCmd: CmdNumber; menuIcon: Byte);
```

Alters the icon shown in the menu for the menu item with command number aCmd. You should call this routine to change the command icon rather than calling Menu Manager routines directly.

```
PROCEDURE SetCmdName(aCmd: CmdNumber; menuText: Str255);
```

Alters the text of the menu item with command number aCmd to menuText. You should call this routine to change the command text rather than calling Menu Manager routines directly.

Segment-manipulation routines

* *Debugging note*: You cannot set a MacApp breakpoint at any of these routines, because they must not call anything (such as `%_BP`) that may require a segment load.

```
FUNCTION GetSegNumber(aProc: ProcPtr): INTEGER;
```

Given a pointer to a procedure, returns the number of the segment containing the procedure.

```
FUNCTION PreloadSegment(segNum: INTEGER): BOOLEAN;
```

For programmers who want to lock a segment at the top of the heap without having to call a dummy procedure in that segment, returns TRUE if the segment could be loaded.

```
PROCEDURE SetResidentSegment(segNum: INTEGER; makeResident: BOOLEAN);
```

Makes a segment resident or no longer resident. Resident segments will not be unloaded by UnloadAllSegments. If a segment is made resident, it is also preloaded. MacApp automatically marks its resident segment as resident; you should probably call UnloadAllSegments before making a segment resident, to ensure that the newly resident segment is locked at the top of the heap.

```
PROCEDURE UnloadAllSegments;
```

Unloads all segments except the Main segment and segments marked resident. Never call this from a non-resident segment.

Utility routines

```
FUNCTION RectIsVisible(r: Rect): BOOLEAN;
```

Returns TRUE if the given Rect is visible in the current grafPort's visRgn. If this is called during the update phase, that is, from one of your view.Draw methods or a method called by view.Draw (as it normally is), the visRgn is set to the region that is visible and needs to be updated. If printing, returns TRUE if the rectangle is on the current page.

```
FUNCTION RectsNest(outer, inner: Rect): BOOLEAN;
```

Returns TRUE if the Rect given by inner nests within that given by outer. This returns TRUE even if the borders are the same.

```
PROCEDURE StdAlert(alertId: INTEGER);
```

Displays a standard alert. It calls Alert with NIL filterProc and throws away the result.

Failure-handling routines

This section describes the global routines provided as part of MacApp's failure-handling mechanism. These routines are intended to provide a generalized mechanism for failure recovery. See the "Failure Handling" recipe in the Cookbook for information on how to use these routines.

```
PROCEDURE CatchFailures(VAR fi: FailInfo; PROCEDURE Handler(e: INTEGER; m: LONGINT));
```

Sets up an exception handler. This pushes your handler onto a stack of exception handlers. If MacApp has already pushed a handler onto the stack, yours is above it, so a call to Failure results in a call to your handler. Your handler generally returns, which calls Failure again to invoke the MacApp exception handler. You may call FailNewMessage instead. (That results in a call to Failure, but chooses between two possible messages first.)

```
PROCEDURE Failure(error: INTEGER; message: LONGINT);
```

Signals a failure. This pops the most recently posted exception handler off the handler stack and calls it. FailNIL, FailOSError, FailMemError, and FailResError check for failures and then call this routine. You generally call Failure when you detect a failure condition not detected by FailNIL, FailOSError, FailMemErr, or FailResError.

```
PROCEDURE FailNewMessage(error: INTEGER; oldMessage, newMessage:
LONGINT);
```

This procedure calls Failure and passes the error and newMessage or oldMessage. FailNewMessage passes the oldMessage parameter to Failure unless it is 0, in which case newMessage is passed. This is used in an error handler so that the error handler can provide a message (newMessage) only if a message was not provided already. You would call this if you wanted to set the message value to override a message value established by a lower-level handler.

```
PROCEDURE FailNIL(p: UNIV Ptr);
```

Called with a pointer or handle, this signals Failure(memFullErr, 0) if the pointer or handle is NIL.

```
PROCEDURE FailOSErr(error: INTEGER);
```

Called with an OSError, signals Failure(error, 0) when error is not noErr.

```
PROCEDURE FailMemError;
```

Call this when you suspect there may have been a memory error (generally because you just attempted to allocate a new object). Tests the value of MemError. If MemError <> noErr, this signals Failure(MemErr, 0). If you are using assembly language, then you should just test the return code from the Memory Manager in D0 by calling FailOSErr.

```
PROCEDURE FailResErr;
```

Call this when you suspect there may have been a resource error. If ResError is not equal to noErr, this calls Failure(ResError, 0).

```
PROCEDURE Success(VAR fi: FailInfo);
```

Call this when you want to remove the most recently installed handler from the exception handler stack. Pops one element off the handler stack, but doesn't call the handler.

Miscellaneous routines

```
PROCEDURE BusyDelay(newDelay: INTEGER; forceBusy: BOOLEAN);
```

Changes the busy cursor delay. The newDelay value should be in sixtieths of a second; a value less than or equal to zero means don't change the delay. If forceBusy is TRUE, then the watch is put up immedately, otherwise it doesn't go up until the required time has passed.

```
PROCEDURE ResetBusyCursor;
```

Changes the cursor back to an arrow and resets the time counted for the cursor delay before the pointer changes back to a watch. This is called automatically if you call GetNextEvent or EventAvail.

```
PROCEDURE BusyActivate(entering: BOOLEAN);
```

Activates or deactivates the busy cursor mechanism.

```
PROCEDURE CanPaste(aClipType: ResType);
```

Call this in your DoSetupMenus code to register an ability to paste a particular type of Clipboard data.

```
PROCEDURE EachHandler(aFirstHandler: TEvtHandler; PROCEDURE
DoToEvtHandler(anEvtHandler:
                    TEvtHandler; VAR stopNow: BOOLEAN));
```

Performs DoToEvtHandler to aFirstHandler, then to its fNextHandler, and so on until the fNextHandler chain ends at NIL or stopNow is set to TRUE.

```
PROCEDURE InitUDialog;
```

Registers an instance of each view class defined in UDialog so that objects of those classes can be created from 'view' resources. Note that if you are not using all of the view classes defined in UDialog you can reduce the size of your application by registering on;y the classes you use *instead* of calling InitUDialog.

```
PROCEDURE InitToolbox(callsToMoreMasters: INTEGER);
```

Does essential Toolbox initialization. Every MacApp application should call this as the very first action in its main program. If you also use the printing unit UPrinting, call InitPrinting just after you call InitToolbox. The value callsToMoreMasters multiplied by 32 is the number of master pointers initially allocated. New master pointers are automatically allocated if they are needed later, but that does not use memory as efficiently (because that means allocating nonrelocatable blocks). You should probably start with a value of 3 or 4 initially; when you are fine-tuning your application, you can use the MacApp Interactive Debugger Report Memory Management Information flag to find out how many master pointers you typically need.

```
PROCEDURE InitUTEView;
```

Initializes UTEView. Calling this routine is necessary only if you are using UTEView and you are creating a TTEView object from a 'view' resource.

```
FUNCTION PutDeskScrapData(aResType: ResType; aDataHandle: Handle):
OSErr;
```

Writes data to the desk scrap. Call this from your TView method WriteToDeskScrap. The return code from the Scrap Manager is returned as the function value. It will be noErr unless something went wrong. This procedure leaves aDataHandle unlocked. Rather than calling PutDeskScrapData, you can call the Toolbox routine PutScrap yourself.

```
PROCEDURE FreeObject(obj: TObject);
```

If obj is not NIL, calls the Free method for that object. This is useful for freeing an object that might sometimes be NIL.

```
FUNCTION LengthRect(r: Rect; vhs: VHSelect): INTEGER;
```

Returns the length of the rectangle in direction vhs.

```
FUNCTION Max(a, b: LONGINT): LONGINT;
```

Returns the larger of the two given numbers.

```
FUNCTION Min(a, b: LONGINT): LONGINT;
```

Returns the smaller of the two given numbers.