Peeking at the New M Windowing API

by Guy Gardner

It's a Different World

With the introduction of the Apple Lisa computer in 1983 and the Macintosh in 1984, the windowing desktop metaphor has steadily gained acceptance among microcomputer users. As an example, by the end of 1992, dollar sales of MS Windows applications probably exceeded MS DOS application sales for the first time.

Users demand more consistency in user interface, ease of use, and user-driven applications along with multipurpose workstations. Leading windowing systems running on modern bitmap workstations together with windowing applications answer these user needs.

M Technology is now at a crossroads, after having gone through steady market growth since its inception. The roll-and-scroll or forms-based user interfaces are no longer acceptable to many users. Without the ability to generate native windowing-system-based applications, the M market will recede as users look elsewhere for windowing applications.

This API approach will make it easier and faster for M programmers to develop windowing applications than will all of the third and fourth generation languages and tools available on these windowing platforms.

The MDC or MUMPS Development Committee (the ANSI X11 committee responsible for the MUMPS standard), is finishing up the standards process for the M windowing application program interface (API) specification. When complete, this will enable programmers to create native windowing-system-based applications in M. These applications will be portable across MS Windows, Macintosh, OS/2 PM, and the X Window systems. This API approach will make it easier and faster for M programmers to develop windowing applications than will all of the third and fourth generation languages and tools available on these windowing platforms. New M tools, layered on top of the API, will increase this productivity.

Architectural Features of the Windowing API

Initially in the Macintosh Hypercard, more recently the Windows-based Toolbook, Visual Basic, Powerbuilder, and other tools have led to high-productivity GUI (graphical user interface) programming capabilities without the need to know the host windowing system's API or low-level details. Instead of the hundreds of low-level function calls that C and Pascal programmers need to know, these environments present the programmer with a simplified API. This is done by a series of new paradigms or ways of programming.

Let's look at some of these new programming features.

Separation of Form from Function—a feature that allows the visual aspects of the windows to be set up in a WYSIWYG ("what you see is what you get") window-painter environment. Programming code is specified in a pop-up window-based text editor and "attached" directly to the part of the window that the code acts on or for. Experience indicates that this type of programming results in dramatically improved productivity and enables fairly inexperienced programmers to create windowing-based applications quickly and effectively.

Attributes with Side Effects—this feature greatly reduces the complexity of the code by replacing function calls with a series of attributes, which, if modified, change the system as a side effect.

The advantages of this feature over traditional coding appear in the following example demonstrating two ways to move a window to a new position relative to its current position.

First, the traditional approach (in Pascal-like pseudo-code):

```
VAR XPosition AS INTEGER
VAR YPosition AS INTEGER
VAR bad AS INTEGER
XPosition:=GetWindowXPosition(MyWindowID)
YPosition:=GetWindowYPosition(MyWindowID)
bad:=MoveWindow(MyWindow,XPosition+20,YPosition)
IF bad THEN HandleError(bad)
```

Now, the attribute-based version:

```
MyWindow.XPosition:=MyWindow.XPosition+20
```

There are significant areas in which our API outshines other windowing-system application tools by playing up the advantages of unique M features.

It is easier to remember attribute names for windows and have side effects handle the low-level details than to memorize hundreds of functions with their respective parameters and return values. The system commonly ignores any attempt to modify an attribute with an invalid value.

Attributes make coding closer to a more declarative programming model (desirable) and move away from the complexities of the procedural model (detailing how to do exactly what you want to do). Think of how much complexity is hidden behind a single Global SET in M and you'll get the idea.

The attributes feature also easily handles the hierarchies that are normally present in windowing-system visual components. For example, in Visual Basic, to modify the text in a text field on another window, the following code could be used:

```
WindowName.ChildWindowName.FieldName.Text="Some text"
```

Event-Oriented Processing—enables the programmer to generate an application controlled or driven by the user instead of the software driving the user. The user decides what and when something is to be done, instead of the software forcing a series of prompts in a predetermined order. Many event-oriented systems use a callback metaphor in which the system software automatically performs callbacks to application subroutines based on specific user and system events. These callbacks are usually synchronized rather than asynchronous, which means that a given process cannot initiate a second callback until the event-handler has completed the first. The system has processing control most of the time and the application code runs as subroutines, usually taking small amounts of processing time and reacting only to the event at hand.

Other aspects of these application tools that further enhance development are: tight integration of the windowing API into the application language instead of C's separate include files and M's external bindings, extensible application environment features (i.e., XCMDs in Hypercard and Custom Controls in Visual Basic), and an extensible operating system environment (i.e., DLLs under Windows and Extensions and INITs on the Mac).

The M windowing API incorporates the attribute side effect and event-oriented processing features along with other aspects of the aforementioned application tools. All that is missing in the separation-of-form-from-function feature is the window painter. This should not present a problem since M-based windowing/forms tools will be adapted to fill this need. Vendors may even include window painters as part of their API offerings.

M's Particular Windowing Advantages

There are significant areas in which our API outshines other windowing-system application tools by playing up the advantages of unique M features. This API uses structured system variables (ssvns) to store all attributes representing the various aspects of the host windowing system, windows, and gadgets (the smaller objects/controls that appear on windows). These ssvns work and look much like M global arrays, but are actually local to the job and they have side effects. Window and gadget attributes are contained in the ^$WINDOW ssvn and the workstation display information is in the ^$DISPLAY ssvn while information regarding the current event being processed is in the ^$EVENT ssvn.

New to the M standard is MERGE, the command that copies an entire M array substructure onto another without requiring the programmer to write $ORDER loops to transverse the subscript levels. This is similar to the SET * capability that was in MIIS, an early variant of the M language. By using the MERGE command with globals and the ^$WINDOW ssvn, the programmer can easily copy any visible window into a global or copy a window definition that is in a global into ^$WINDOW, thereby enabling the window display with a single M command. For example,

```
MERGE ^$WINDOW("myWindow")=^GLOBAL
```

This is a great improvement over Hypercard-like tools wherein only a single attribute can be modified or examined at a time.

This is a great improvement over Hypercard-like tools wherein only a single attribute can be modified or examined at a time. Being able to copy a window's attributes along with its gadget attributes as represented in a hierarchy of a global or ^$WINDOW further shows M's advantage. Variants of window and/or gadget attribute definitions can be combined with standard definitions from M globals into a single ^$WINDOW entry using a single MERGE command. For example,

```
MERGE^$WINDOW("myWindow")=^STANDARD,^$WINDOW("myWindow")=
^VARIANT(1)
```

Some of the attributes defined for windows and gadgets tell the API what M subroutines to call when certain events occur in the system. The API automatically will call these subroutines, also known as event-handlers or callbacks, when specified events occur. In fact, the API handles most aspects of most events automatically, relieving the programmer of most details of the windowing system's operations. For instance, the API and host windowing system automatically make push buttons look pushed when they are clicked on and text entry boxes provide editing features that comply with the host windowing system.

As in the Hypercard-like tools, most details about windows and gadgets are done in the API with attributes. A few functions and new commands are incorporated for things that are not appropriate for attributes. An example is the new ESTART command, which puts M into event processing mode when invoked.

The API also allows a single MUMPS job to address more than one display or workstation at the same time. That single job can put windows up on multiple displays that are networked, and get event callbacks from several users simultaneously. This is essential for complete support of the X Window networked operating system. Similar capabilities across Apple's Appletalk and Microsoft's Windows for Workgroups networks are certainly possible. Assuming a vendor provided the ability, the API running would not preclude an M job on a Macintosh from putting up and running a window on an MS Windows-based personal computer using a network connection (and possibly a nice feature to add to Open MUMPS Interconnect?).

General Structure

The following is general structure information on the ssvns used in the API. Most, but not all, of the M windowing API ssvn nodes fit into the model shown in figure 1.

In this model, windowName is a programmer-defined unique string identifying a window. For example, myWindow eventName is one of the following: CHANGE, CLICK, CLOSE, DBLCLICK, FKEYDOWN, FKEYUP, FOCUS, GOBOTTOM, GODOWN, GODOWNBIG, GOTOP, GOUP, GOUPBIG, HELP, KEYDOWN, KEYUP, MAX, MIN, MOVE, PDOWN, PDRAG, PMOVE, PUP, RESIZE, RESTORE, SELECT, TIMER or UNFOCUS. Not all event names are valid for all gadgets and windows.

Implementors can extend this list with names starting with a Z.

doArgument is a list of MUMPS entry references that will be called when an event occurs. For instance, `A^PGM,B^PGM2(1)` gadgetName is a programmer-defined unique string identifying a gadget on a window. For example, `OK-Button` displayID is a vendor-specified unique string identifying a specific display.

attributeName is one of the following:

- For `^$DISPLAY`: BCOLOR, CLIPBOARD, COLOR, COLORTYPE, FCOLOR, FOCUS, KEYBOARD, PEN, PLATFORM, PTR, SIZE, SPECTRUM, TYPEFACE, or UNITS.
- For `^$EVENT`: CHOICE, CLASS, ELEMENT, KEY, NEXTFOCUS, OK, PBUTTON, PPOS, PRIORFOCUS, PSTATE, SEQUENCE, TYPE, or WINDOW.

```
^$WINDOW(windowName,attributeName) = value
^$WINDOW(windowName,attributeName,pos) = value
^$WINDOW(windowName,"EVENT",eventName) = doArguments
^$WINDOW(windowName,"G",gadgetName,attributeName) = value
^$WINDOW(windowName,"G",gadgetName,attributeName,pos) = value
^$WINDOW(windowName,"G",gadgetName,"EVENT",eventName) = doArguments
^$WINDOW(windowName,"M",menuName,"CHOICE",pos) = menuDisplayName
^$WINDOW(windowName,"M",menuName,"CHOICE",pos,attributeName) = value
^$WINDOW(windowName,"M",menuName,"CHOICE",pos,"EVENT",eventName) = doArguments

^$EVENT(attributeName) = value
^$EVENT(attributeName,pos) = value^$DISPLAY(displayID,attributeName) = value
^$DISPLAY(displayID,attributeName,pos) = value
^$DISPLAY(displayID,"EVENT",eventName) = value
```

Figure 1. General structure model.

- For windows in ^$WINDOW: ACTIVE, BCOLOR, COLOR, DEFBUTTON, DISPLAY, EVENT, FCOLOR, FFACE, FSIZE, FSTYLE, ICON, ID, ITITLE, MENUBAR, MIN, MODAL, NEXTG, PARENT, POS, RESIZE, SCROLL, SIZE, SIZEMIN, SIZEWIN, TIED, TITLE, TYPE, UNITS, or VISIBLE.
- For gadgets in ^$WINDOW: ACTIVE, BCOLOR, CANCEL, CANCHANGE, CHANGED, CHARMAX, CHOICE, DRAW, DRAWTYPE, EVENT, FCOLOR, FFACE, FRAMED, FSIZE, FSTYLE, ID, INSELECT, INTERVAL, NEXTG, POS, RESOURCE, ROWCOL, SCROLL, SCROLLBY, SCROLLDIR, SCROLLPOS, SCROLLRANGE, SELECTMAX, SELECTVAL, SIZE, TBCOLOR, TFCOLOR, TFFACE, TFSTYLE, TITLE, TOPSHOW, TPOS, TYPE, UNITS, VALUE, or VISIBLE.

Not all attribute names are valid for all windows, gadgets, events, and displays. Implementors can extend this list with names starting with a Z.

- The term pos is a number or string that indicates the position of a value in a repeating attribute list. For example, 1 or 12-23-C34-44-55A.
- menuName is a programmer-defined unique string identifying a menu associated with a window.
- menuDisplayName is the menu name as displayed to the user.
- value is the actual value of a specified attribute, such as `1` or `"Window A."`

The gadget level TYPE attribute values supported are: BUTTON, CHECK, DOCUMENT, FRAME, GENERIC, LABEL, LIST, LISTBUTTON, LISTENTRY, LONGLIST, RADIO, SCROLL, SYMBOL, TEXT, and any implementor-specific Z types.

Keep in mind that most application programmers will only be concerned with a small subset of the attribute names and

event names presented above. The attribute and event names are short yet fairly descriptive and many readers will be able to guess the functionality provided by many of them.

M Windowing API Coding Examples

Now we take a look at coding samples from a component of medical record-keeping. These simplified examples preview some aspects of programming in the proposed MDC MUMPS Windowing API. Keep in mind that even though most syntactic and semantic details of the final M windowing API are known as of this writing, there may be some minor aspects of the API that will change before it becomes an MDC Type A.

. . . many other attributes exist for both windows and gadgets but we only have to specify a few because the proposed M windowing API defaults them for us.

Coding for a simple patient add window may look as it does in figure 2. Keep in mind that many other attributes exist for both windows and gadgets but we only have to specify a few because the proposed M windowing API defaults them for us. We can reference any attribute in ^$WINDOW, including the ones defaulted by the API, for any gadget or window known to it.

```
Routine pat:
pat             ; Routine to run 'Patient Add' Application@glg@
                ;
                ; Create the 'patadd' window definition, if not already done.
                DO ^pati:'$DATA(^windows("patadd"))
                ; Put the 'patient add' window up. (defines and display it)
                MERGE ^$WINDOW("patadd")=^windows("patadd")
                ; Start event processing.
                ; The API automatically does event callbacks while program
                ; control at this level stays within the ESTART command.
                ESTART
                ; We get here after an event callback executes an ESTOP command.
                ;
                ; Take the window down. (stops displaying it and makes it undefined)
                KILL ^$WINDOW("patadd")
                ; All done.
                QUIT
                ;
                ;
                ; Event handlers for window 'patadd'
                ; The MWAPI does an implied DO to these handlers while control
                ; is within the ESTART command above.
                ;
pnmv            ; CHANGE event handler for the patient name (text changed).
                NEW a
                ; Get the just edited value from the text gadget 'pname'
                SET a=$GET(^$WINDOW("patadd","G","pname","VALUE"))
                ; We are doing required field checks at filing time (in the OK
                ; button) so don't bother now.
                IF a="" QUIT
                ;
                ; If it doesn't look like a name.
                IF a'?1.U1","1.U DO
                . ; Make focus stay on the pname gadget by refusing this event.
                . ; This KILL command will also suppress UNFOCUS and FOCUS
                . ; events that are about to be processed by the MWAPI.
                . KILL ^$EVENT("OK")
                . ; Put up a modal error window (error message box).
                . ; See later detaxs on routine%win.
                . DO msg^%win("Validation Error","Patient Name format is incorrect")
                ;
                ; Exit this event handler and return control to the MWAPI.
```

Figure 2 continues

```
                QUIT
                ;
                ;
pnov            ; CHANGE event handler for patient number (text changed).
                NEW a
                ; Get the new patient number.
                SET a=$GET(^$W("patadd","G","pnumber","VALUE"))
                ; We are doing required field checks later.
                IF a="" QUIT
                ;
                ; If it isn't a number.
                IF a'?.N DO QUIT
                . ; Refuse the CHANGE event.
                . KILL ^$EVENT("OK")
                . ; Inform the user of the error.
                . DO msg^%win("Validation Error","Patient Number must be numeric")
                ;
                ; We are only adding new patient therefore we must refuse any
                ; number already on file.
                ; NOTE: This method is not bullet proof on a multiuser system.
                IF$DATA(^pat(a)) DO QUIT
                . ; Refuse the CHANGE event.
                . KILL ^$EVENT("OK")
                . ; Tell the user.
                . DO msg^%win("Validation Error","Patient Number is not unique")
                ;
                ; Quit this event handler.
                QUIT
                ;
                ;
ok              ; SELECT event handler for'OK' button (button pushed).
                NEW a
                ; Get the patient number.
                SET a=$GET(^$W("patadd","G","pnumber","V"))
                ;
                ; If no patient number entered the do a required message and exit.
                IF a="" DO reqd("Patient Number","pnumber") QUIT
                ; If no patient name entered do a required message and exit.
                IF$GET(^$W("patadd","G","pname","V"))="" DO QUIT
                . DO reqd("Patient Name","pname")
                ; Patient Sex is optional.
                ;
                ; File data into ^pat under patient number.
                FOR b="psex","pname" DO
                . SET ^pat(a,b)=$GET(^$W("patadd","G",b,"V"))
                ; Signal the MWAPI to stop event processing mode.
                ESTOP
                ; Quit this event handler.
                QUIT
                ;
exit            ; SELECT event handler for the Exit menu item.
                ; Ignore work in progress and stop event processing mode.
                ESTOP
                ; Quit this event handler.
                QUIT
                ;
halt            ; SELECT event handler for the Halt menu item.
                ; HALT will also take down all windows and stop event processing.
                HALT
                ;
reqd(field,gadget) ; Do require message box and set FOCUS to a gadget.
                ; Set focus to gadget that is missing data.
```

Figure 2 continues

```
                ; Window "patadd" will get focus back at this gadget when the user
                ; has dismissed the message box window. (see below)
                ; $PDISPLAY is the'principal display' for this MUMPS job.
                SET ^$DISPLAY($PDISPLAY,"FOCUS")="patadd,"_gadget
                ; Put up a modal message box.
                DO msg^%win("Missing Value",field_" is required")
                ; Return to caller.
                QUIT
                ;
can             ; SELECT event for the'Cancel' button.
                ; No need to file anything if user canceled.
                ; Signal the system to stop event processing mode.
                ESTOP
                ; Quit this event handler.
                QUIT
Routine pati:
pati            ; Setup ^window("patadd") with the window definition @glg@
                ;
                ; Zap old definition of the window. For recompiles only.
                KILL ^windows("patadd")
                ;
                ; We only need to set up attributes that do not have defaults or
                ; where we don't want the MWAPI defaults.
                ;
                ;
                ; Window level attributes:
                ;
                ; Window Title.
                SET ^windows("patadd","TITLE")="Add a Patient"
                ; Window Units.
                ; We can use CHAR, PIXEL, POINT, REL or an implementor
                ; specific Z unit here. All gadgets on this window use CHAR
                ; since UNITS are not specified at the gadget level.
                SET ^("UNITS")="CHAR"
                ; Indicate that the OK button should be the default button.
                SET ^("DEFBUTTON")="ok"
                ; Mark the'main' menu as the window's menubar.
                SET ^("MENUBAR")="main"
                ; Make window resizable by the user.
                SET ^("RESIZE")=1
                ; Allow the user to scroll the window when needed.
                SET ^("SCROLL")=1
                ; POS and SIZE attributes will be calculated by the MWAPI for us!
                ; Among other defaulted attributes, VISIBLE will default to 1 (true).
                ;
                ;
                ; Menus and their attributes:
                ;
                ; Set up the'main' menu (menubar level).
                ; First menubar level menu is File. The keyboard accelerator is F.
                SET ^windows("patadd","M","main","CHOICE",1)="&File"
                ; The drop down menu is'file'.
                SET ^(1,"SUBMENU")="file"
                ; Second menubar level menu is Wow. Accelerator is W.
                SET ^windows("patadd","M","main","CHOICE",2)="&Wow"
                ; The drop down menu is'wow'.
                SET ^(2,"SUBMENU")="wow
                ;
                ; Set up the'file' menu drop down with one item.
                SET ^windows("patadd","M","file","CHOICE",1)="&Exit"
                ; DO exit^pat when selected.
```

Figure 2 continues

```
SET ^(1,"EVENT","SELECT")="exit^pat"
; Set up the'wow' menu drop down with one item.
SET ^windows("patadd","M","wow","CHOICE",1)="&Halt"
; DO halt^pat when selected.
SET ^(1,"EVENT","SELECT")="halt^pat"
;
;
; Gadgets and their attributes:
;
; Patient Name prompt:
; This is a TEXT (entry box) gadget.
SET ^windows("patadd","G","pname","TYPE")="TEXT"
; Position at H,V (in CHAR units as defined at window level).
SET ^("POS")="20,1"
; Size of text display area (one line of 30 characters).
SET ^("SIZE")="30,1"
; Allow 40 characters to be entered. Scrolling is automatic if needed.
SET ^("CHARMAX")=40
; Title or label. Note:'N' is the Accelerator.
SET ^("TITLE")="Patient &Name:"
; Make the Title appear to the left of the text entry area.
SET ^("TPOS")="LEFT"
; Event handler for change validation.
SET ^("EVENT","CHANGE")="pnmv^patadd"
;
; Patient Sex prompt:
; This is a LISTBUTTON (dropdown list box) gadget.
SET ^windows("patadd","G","psex","TYPE")="LISTBUTTON"
SET ^("POS")="20,3"
; Make 8 characters wide and 3 items shown in the popup list.
SET ^("SIZE")="8,3"
SET ^("TITLE")="Patient &Sex:"
SET ^("TPOS")="LEFT"
; Make three choices available.
SET ^("CHOICE",1)="Female"
SET ^(2)="Male"
SET ^(3)="Unknown"
; No CHANGE event handler needed.
;
; Patient Number prompt:
SET ^windows("patadd","G","pnumber","TYPE")="TEXT"
SET ^("POS")="20,5",^("SIZE")="15,1",^("CHARMAX")=15
SET ^("TITLE")="Patient &Number:",^("TPOS")="LEFT"
SET ^("EVENT","CHANGE")="pnmv^patadd"
;
; OK button:
SET ^windows("patadd","G","ok","TYPE")="BUTTON"
SET ^("POS")="30,7",^("TITLE")="&OK"
; SELECT is when users pushes the button.
SET ^("EVENT","SELECT")="ok^patadd"
;
; Cancel Button:
SET ^windows("patadd","G","cancel","TYPE")="BUTTON"
SET ^("POS")="40,7",^("TITLE")="&Cancel"
SET ^("EVENT","SELECT")="can^patadd"
;
; All done setting up the'patadd' window definition
; Note: The API uses defaults to fill in the other
; attributes that we have not specified.
QUIT
```

Figure 2. Coding for 'Patient Add' window.

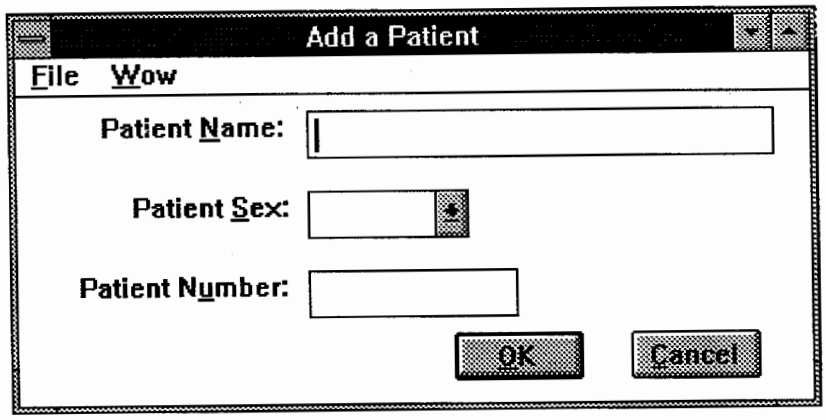

Figure 3. 'Patient Add' window.

After executing DO ^pati on an MS Windows-based computer, the window pictured in figure 3 will appear on the monitor.

The code fragments in figure 4 illustrate some of the centralized utility subroutines that could be used by application programmers.

```
%win; MWAPI utilities @glg@
          ;
msg(title,text)  ; Put up a modal message box.
          NEW wname
          ; Create a basic box.
          DO box
          ; Add the OK button and SELECT event handler.
          DO but("&OK","stop^%win")
          ; Run the modal window.
          DO run
          ; Zap the message box.
          ; NOTE: FOCUS moves back to the callers window at this point.
          KILL ^$W(wname)
          ; Return to caller.
          QUIT
          ;
bbox(title,text,but1,but2,but3,but4,but5) ; Put up a modal button box.
          NEW wname,answer
          ; Create a basic box.
          DO box
          ; Add the buttons that were specified, the first two are required.
```

Figure 4 continues

```
           DO but(but1,"butdown^%win")
           DO but(but2,"butdown^%win")
           IF$DATA(but3) DO but(but3,"butdown^%win")
           IF$DATA(but4) DO but(but4,"butdown^%win")
           IF$DATA(but5) DO but(but5,"butdown^%win")
           ; Run the window.
           DO run
           ; Zap the window.
           KILL ^$W(wname)
           ; Return the button name that was pushed.
           QUIT answer
           ;
prompt(title,text,default); Put up a modal prompt box.
NEW wname,answer
           ; Create a basic box.
           DO box
           ; Add a TEXT gadget to the window.
           SET ^$W(wname,"G","answer","TYPE")="TEXT"
           SET ^$W(wname,"G","answer","POS")="20,1"
           SET ^$W(wname,"G","answer","SIZE")="30,1"
           SET ^$W(wname,"G","answer","CHARMAX")=50
           ; Set up in the title.
           SET ^$W(wname,"G","answer","TITLE")=title
           SET ^$W(wname,"G","answer","TPOS")="LEFT"
           ; Set up the default answer.
           SET ^$W(wname,"G","answer","VALUE")=default
           ; Run the window.
           DO run
           ; Get the text that was entered.
           SET answer=^$W(wname,"G","answer","VALUE")
           ; Zap the window.
           KILL ^$W(wname)
           ; Return the text.
           QUIT answer
           ;
ask(title,text)  ; Put up modal ask box.
           NEW wname,answer
           ; Create a basic box.
           DO box
           ; Add OK button.
           DO but("&OK","butdown^%win")
           ; Add Cancel button.
           DO but("&Cancel","butdown^%win")
           ; Run the window.
           DO run
           ; Zap the window.
           KILL ^$W(wname)
           ; Return to caller with 'OK' or 'Cancel'.
           QUIT answer
           ;
           ;
box        ; Subroutine to create a basic box (invisible)
           ; Generate a unique window name. NOTE: wname is NEWed by the caller.
           wname="%win-"_$H_"-"_$R(500)
           ; Copy window template in a global into ^$WINDOW.
           ; Note: VISIBLE is false and ^%win("box") has been
           ; previously setup for our use.
           MERGE ^$W(wname)=^%win("box")
           ; Override default window title and title of LABEL gadget.
           SET ^$W(wname,"TITLE")=title
           SET ^$W(wname,"G","message","TITLE")=text
           ; Box is invisible but ready to use.
           QUIT
           ;
```

Figure 4 continues

```
but(txt,hand); Add a new button to the box.
          ; Set gadget type
          SET ^$W(wname,"G",txt,"TYPE")="PUSH BUTTON"
          ; Set text of button.
          SET ^$W(wname,"G",txt,"TITLE")=txt
          ; Set event handler for a push of the button.
          SET ^$W(wname,"G",txt,"EVENT","SELECT")=hand
          ; Normally, some code would appear here to position the button on the window. This is left as
          ; an exercise to the reader.
          QUIT
          ;
butdown   ; Generic handler for a button press that needs handling.
          ; Get the name of the button pushed.
          SET answer=^$EVENT("GADGET")
          ; Take out '&' if used as accelerator.
          SET answer=$TR(answer,"&")
          ; We are done.
          GOTO stop
          ;
run       ; Make box visible (modal).
          SET ^$W(wname,"VISIBLE")=1
          ; Move focus to this window.
          SET ^$DISPLAY($PDISPLAY,"FOCUS")=wname
          ; Do event processing for the modal window.
          ;NOTE: Only this 'modal' window gets events now.
          ESTART
          ; Window has completed, return to caller for further processing.
          ;processing.
          QUIT
          ;
stop      ; Generic handler to stop the window from running.
          ESTOP
          ; Return to the MWAPI.
QUIT
```

Figure 4. Coding examples for window utilities.

The M windowing API also supports the following features not covered in figure 4: menu items with checked items, a GENERIC box gadget that allows the user to draw and support a particular style of gadget entirely in M code, dropdown list box and list box gadgets, most other standard gadgets that one would expect for M applications, and MTerm, a type of window that supports older style dumb terminal-based M applications. There are many other features and functionalities.

An MDC working group has discussed adding two new commands or functions to M that give the programmer the ability to generate and use native window resources as provided by the host windowing system or the vendor. The first command would "snapshot" a window that is currently in ^$WINDOW into a specified resource file. A window design utility could use this functionality. The second would restore a window definition, in a resource file, back into ^$WINDOW. The running application program could use this side of the functionality.

These commands would allow M-based window painters to "save" prevalidated, compiled window definitions that could be "restored" very efficiently at runtime. This functionality would probably be introduced as implementor-specific extensions to the M windowing API since it will not be included in the initial standard.

A window painter could be created in the API by a main painter window by using menu items to allow window creation and editing and the gadgets on them. A window will appear separately but will have special event-handlers installed to select a gadget, drag one to a new location on the window, and perform other window-painter functions.

The programmer would enter M code or subroutine entry points to use at runtime by painting a window. When the window definition is saved, it would be merged out of ^$WINDOW into a global and have the event-handlers reset to remove the window-painter handlers and to install the actual runtime event-handlers that the programmers specified.

Fitting the API with the Standard

The MDC will be introducing enhancements to the standard M that may include features such as object-oriented programming, general synchronous and asynchronous event-handling, static M variables, and others. Designing M to enjoy maximum compatibility, the API model should be expanded or even layered under a different scheme. It seems likely that windows and gadgets will be able to be tucked into a class hierarchy wherein attributes and their side effects become instance variables and methods of a class.

Down the API Road

The MDC will not be producing an initial standard that fills the needs of all application developers. That will be left for the future. Meanwhile, M vendors have an approved way of providing additional functionality without interfering with future additions to the standard. The ability to call machine-level code that bypasses the API and a "Z" namespace for additional attributes and events for windows and gadgets are just two such features. Eventually the API will be enhanced to support these new features. The ssvn approach should make this easier when compared with a function-based API in that it is easier to add new attributes to ssvns with reasonable defaults than it is to add new parameters to existing functions.

M Prospects

The Gartner Report commissioned by the M Technology Association indicates that the worldwide M market will double in size in the next four years. This should tell us that M Technology is indeed healthy. In fact, it may be that M is on the verge of vast market expansions as technologically superior features are added to the language over the next year or two. Where else are ANSI standard open systems supporting portable applications across all major windowing systems? Add in M's high productivity along with the ease of use built into the windowing API and you have a winning combination.

More Information on the Windowing API

For additional information contact the M Technology Association office for copies of MDC MUMPS windowing API documents. The following documents may be of special interest: X11/SC11/TG4/93-3 and X11/SC11/TG4/WG6/93-12. Some M vendors already are working to support the API and also may be a source of information; some already have alpha and early beta copies of M to support the API available to developers.

Guy Gardner is chief research engineer and tools product manager at Collaborative Medical Systems in Waltham, Massachusetts.

Invitation to Authors from the Editors

The editors of *M Computing* (formerly *MUMPS Computing*) welcome articles and news items submitted for publication. Topics include but are not limited to M application areas, new M systems and installations, transfer of M applications, interfacing systems, needs of computing, programming techniques, challenges of technology, and related areas. Also, send your book reviews, information about meetings (past and future), programming tips and tricks, product announcements, industry news, and news of MUMPS users' groups worldwide.

We'd like to hear your article ideas. Your article will be reviewed and, if accepted for publication, will appear in one of the following issues. Mail or fax a brief description of your proposed article to Marsha Ogden, managing editor. You will receive a copy of author guidelines and procedures for submitting articles.

Final determination about any copy submitted for publication in the magazine rests with the editors. All material is subject to editing. Contact Marsha Ogden, Managing Editor, M Technology Association, Suite 205, 1738 Elton Road, Silver Spring, MD 20903. Phone 301-431-4070, fax 301-431-0017, or by FORUM.

Coming in Future Issues of *M Computing* 1993 and 1994!

Publication	*Focus*	*Deadline*
September 1993	Multilingual Applications	July 2, 1993
November 1993	Database Research	August 20, 1993
February 1994	Education	November 18, 1993
April 1994	Business and Commerce	January 14, 1994