

# The M Windowing API: Expansive Tool or Expensive Toy?

---

by Gardner S. Trask III

---

**T**he M windowing application program interface (MWAPI) has been touted as the most amazing advancement to M since machines got powerful enough for *eight* character variable names.[1] But is the MWAPI worthy of all this hype? Is the MWAPI an expansive tool or just an expensive toy? Is the MWAPI a panacea or a placebo?

It seems that around every corner there is a user or a manager screaming "GUsize! WINDOWize! Use OOPS! Use Client/Server!" simply because of the latest techno-buzz word in *Newsweek* or *USA Today*. New paradigms and development requests encroach on everyone's limited development resources. Now is the time to start asking some important questions: What does windowing really entail? Will it improve sales? Can I retrofit my application easily? And what are the real costs?

What we need to know is, what is the dark side of windowing?

This article, the first in a three-part series to offer in-depth windows information, will discuss the windows environment and its relationship to existing programming interfaces and practices. It will discuss in general terms how windowing will affect your programming methodology, style, and code. The second installment will discuss the vernacular of windows, i.e., "What is a radio button or list box?", "What's an event driver?", "How does the mouse interact with my application?" The third installment will build an actual application using the MDC Type-A MWAPI.

## Should You Really Open the Window?

Some think "windows" are what they see when using Microsoft Windows or Macintoshes.[2] Windowing does not simply entail a visual desktop or mean creating an icon to represent complex commands for the user to "click" in order to save keystrokes. In the context of this article, windowing is a new method of entering, modifying, and displaying data that is intuitive and visually appealing for the end user.

We who work in M have become accustomed to the "(D)roll and scroll" interface, where the user answers the first question,

has the response run through some validation and error-checking code, and is then prompted for the next question. Roll and scroll went hand-in-hand with the "cafe menu" interface, which center-filled the screen with all available menu alternatives and then invited the user to "Select an Option - 1 thru 10."

The next generation of interface, the panel painter, placed all the questions it could fit up onto a screen, and forced users to answer prompts in a predefined sequence, error checking as it went along. And most recently, we have had M-powered "intelligent" panels and "pseudo-windows" that allow the user to arrow-key from question to question and even to hit some buttons that call for help text or other processes. But all of these interfaces are ancient and outdated in the mainstream computer world. A recent *Database* magazine review of several text retrieval software packages (written in various languages) definitively illustrates the shortcomings of interfaces written in M. When describing the user interfaces of specific products, the authors wrote:

"[Product name] uses a character based line-oriented interface. This is probably due to its MUMPS multi-user, terminal-oriented foundation. This is the weakest part of the package. In these days of windows and graphical user interfaces, having to deal with line-oriented editors is a real frustration. Fifteen years ago, when we would have taken the line-oriented interface as standard, this would have been a dynamite package. Now, were it not for all the power and flexibility that the package offers, we would consider it too outdated. . . . A screen-oriented editor and more point-and-shoot menus would go a long way toward making this an easy-to-use package."[3]

Developers and vendors may ask, Could someone insert our product's name in this quote? If the answer is yes, perhaps it is time to crack open the window.

So you think you are ready "to window"? First you will need the beta version of ACME Corporation's latest implementation of the Type-A MWAPI standard. And you probably have the 120-page MTA standards tome X11/SC11/TG4/WG6/NCC-1701-D (fraught with help text and easy-to-use examples) by your side.[4] Now you may think you are ready to put windows in your products.

But wait. Before beginning, consider some important concepts and paradigms that may be new to users, programmers, or developers. Your view of data entry, manipulation, and presentation is about to change forever. Windows precludes a linear, single-thread, top-down, fall-through mind-set. Users will no longer conform the sequence of questions and answers. They will want to manipulate not only the sequence of questions, but also question appearance and location on menus and panels. Previous error-checking structures, based on question sequencing, are no longer valid; e.g., prompting for the "pregnancy" question can no longer depend on a previous response to the "gender" prompt. Windows developers recognize that a user might employ a combination of a keyboard, a mouse, a light-pen, or possibly a touch screen to input data. Windows programmers realize that users may want to answer questions starting from the bottom of the screen or reconfigure the screen in a specific order, or in specific colors. The applications can no longer expect the individual user to conform; now it is developers and programmers who must be flexible.

## Emulate the Real World

The first, most important concept to incorporate is *consistency*!

Now that this great new tool is available, there may be an inclination to add every possible bell and whistle all at once, but beware, there is danger in this frivolity. It's like getting a deluxe Mr. Potato-Head set: after you have piled on every set of eyes, arms, lips, hats, and glasses (and don't forget the pipe), poor Mr. Potato-Head looks overloaded, cluttered, and nothing at all like a real Potato-Head. It is strongly recommended that developers resist the urge to add one of every available gadget type, font size, and color into each window.

Screen development can be difficult, but it is unreasonable to expect people to be master screen engineers right off. Just as fancy desktop publishing software did not automatically create slick, professionally developed newsletters, the new MWAPI does not automatically lay out the screen in rich, snappy, great-looking formats. This is up to the windows developer. But don't fear, help is available.

Originally developed by IBM, the common user access (CUA) standard tries to define a user interface structure and navigation standard for all "windows-style" applications. The CUA defines such things as how to move between fields, how a menu should be structured, how help text should look, how windows should be layered on top of each other, etc.

The CUA provided the base for the Microsoft Windows-style interface. As a matter of consistency, most windows developers are using a hybrid-CUA. And by examining most of the sustaining, major software packages and vendors, commonalities in architecture, form, and function can be discerned. While this sounds like a lot of predefined structure for open-structure programmers to adapt to, the reality is that this is what the market expects. Think about it, who hates roll and scroll the most? Not the person who only works on the mainframe and has never seen a window. No, it's the person who has to switch back and forth between third-generation roll-and-scroll windows and those clean, intuitive, easily read windows with a common architecture. Because users see how easy and intuitive WordPerfect menus are, or how Lotus works, or how other main-stream windows-based products look and feel, when they are forced to use roll and scroll screens or windows that look homemade and unprofessional, they lament, "Why can't all my software look and work the same?"

It is therefore strongly suggested that new windows developers look to the CUA and other windows products for guidance in structure, look-and-feel, and form. If you want clean, crisp screens that are generally accepted by "power users," look to what the "big boys" are doing. If most windows applications have menu bars placed horizontally at the top of the window, then don't develop menu bars vertically on the left margin. If everybody layers windows from the top left-hand side diagonally to the bottom right, then so should you. The key to acceptability is compatibility.

Of course, when you first open this Pandora's box, play around with it at will. Make all the gadgets different sizes, pop open windows all over the screen, and make every button a different checkerboard color! But when the learning and exploring is done, put away the abstract impressionist artistic tendencies. Picasso could never have developed a professional window. Look and see what the world perceives as a sharp and professional look and feel, and emulate it. Otherwise, you'll be the only one in neon-orange at the fancy dress ball.

## The User Did WHAT??

Two other important paradigm shifts are the concepts of *event processing* and *user control*.

*Event processing* takes the flow of logic and error checking from the developer's hands and puts it in the control of the user. To borrow a popular analogy: Let's suppose a developer is programming a car to drive from point A to point B. The old top-down logic would include steps to insert key, check to see that the transmission was in "park," depress gas, turn key, check to see if the engine started, etc. Event processing visualizes the dashboard as a set of separate objects or gadgets that can be manipu-

lated independently, and in no specific order. An event-control process is sitting as a background job, and when it perceives a "change" or event, it passes this information to the appropriate gadget. Perhaps the user wishes to change the radio station, or turn on the defroster, or flash the high beams at an oncoming car. All these gadgets are controlled by users in an order that they choose. So, the transmission gadget, as an independent process, is simply sitting in a wait state until an "event" happens to it. And when the user chooses to shift to reverse, the transmission gadget "awakes" and takes it upon itself to do some error checking and act accordingly. Encapsulating an event and its parameters, error checking, and processing into the proper gadget can be a formidable change to existing programming architecture.

New windows developers must also recognize the concept of *user control*. Users should be able to modify the display environment as much as is logically and technically feasible. Changes in background and foreground colors, time-out rates, movement of fields, the ability to drag-and-drop help windows around the screen—all these are important to user interaction and for the "market acceptance" of a product.

Speed also plays a role here. If a process or method is going to take a while, either start it as a background process, or at the very least, give the user feedback on the progress. Perhaps change the cursor to an hourglass symbol, or add a "% Complete" window in the foreground. Nothing is more frustrating than waiting for a background task to finish when you have no idea how long it will take.

## The Profit Motive

With all the changes that have been described, should an application vendor even bother? After all, some users still have 40-column punch-card sorters and paper-tape readers, and microcomputers and workstations are a long way off, so why go through the expense and trouble? Well, the answer, in a word, is *money*! If applications and tools don't utilize windows, they can pretty much be guaranteed to cost you money in either lost opportunities or lost customers. Customers won't hesitate to flip to a product that is easier to learn, easier to use, and more friendly.

The fact is, studies have proven that people prefer color to black and white, they prefer graphics to text, and they prefer a standard, familiar interface to several different styles. Studies have also proven that a window interface speeds training, reduces data-entry edits and errors, and increases productivity. Thus, it makes business sense to go to windows. To paraphrase another analogy: At one time there must have been dozens of companies manufacturing buggy whips, and the last company to make buggy whips probably made the best

buggy whip of all. Do you really want to keep making buggy whips? How long will customers support applications and tools with third-generation interfaces?

## Now for the Upside


After listening to all the work, cost, and time it is going to take to bring interfaces into the 1990s, it is time for you to hear a little secret: *You are sitting on a gold mine*. M developers have unique opportunities that no other windows developers have. Without preaching to the choir about M, it must be noted that the M language will be the first ever to have windowing inherent as part of an ANSI standard. What does that mean to developers? It means that the code written now can be ported to *all* vendors conforming to the standard. Developers won't have to care whose M the customer has: As long as it is ANSI standard, it'll run.

What else is unique about windowizing using M Technology? A major, and very distinct, benefit is that the windowing environment is not platform specific. Programmers who develop specifically in Microsoft windows (or any other windows language/tool) are tied exclusively to that environment, forcing all customers into the same environment to run the application. But, under the MWAPI, code written in X Window/Motif can, without modification, run on Microsoft Windows, or Macintosh, or Windows NT, etc. (and vice versa). No other windows language can claim this powerful ability. The MWAPI has been designed to conform to no specific windows vendor, and to actually run on any of them. (Obviously, mapping the MWAPI to a windows environment is the responsibility of the M vendor, and developers will have to lobby hard to get as many platform mappings as possible.)

Are there other advantages to using M? Yes, indeed. The use of the MWAPI in conjunction with other M features and bindings makes this one of the most powerful environments available. For example, programmers can create dynamic, late-binding windows. That is, in all other environments, all possible windows have to be developed, compiled, and saved prior to use. In M, a window that has never existed can be developed on the fly. For example, the medical "patient" window might ask demographic questions such as age and gender; based on the answers, a brand new, unique "symptoms/diagnosis" window could be developed to ask questions specific to that patient's gender and age group. This window may never have existed until run time. This capability is all but impossible with the other windows environments.

Also, this is a most opportune time to incorporate object-oriented programming techniques into your new system. The basis of the MWAPI is to use structured system variables

## What is a Gadget? What kind of Values can I assign?

Answers to these questions and more are just a few  clicks away.

**New CBI now available . . .**

### **MWAPI** Reference & Tutorial CBI

An On-Line Interactive Tool for learning the M Windowing API! The **Tutorial** guides the programmer through a Comprehensive Tutorial that covers all important features of the MWAPI. Exercises are presented for all gadgets. A completely functional windows application is built. The **Reference** contains the full MWAPI specification in hypertext format. It can reside in a Window for quick visual access to gadgets at all times.

Esi also has other CBI's and Lecture/Workshops available in M Programming, File Manager, MSM, DSM, DTM, Object-Oriented Programming, VMS Concepts and EsiAuthor.

**Call Esi for more details!**



**EDUCATIONAL SYSTEMS, INC.**

5 Commonwealth Road • Natick, MA 01760  
Tel: (508) 651-1400 • Fax: (508) 651-0708

and merge commands to take existing code, processes, and/or methods and create brand new windows. This fits well with the OOPS model of classes, methods, objects, and the "inheritance" of common attributes into new objects.

This is a brief overview of some of the benefits of a windows environment written in M. For a more complete picture, I recommend you read the article "Peeking at the New M Windowing API" in the April 1993 *M Computing*. In this article Guy Gardner describes, among other things, architectural features of the windowing API and M's particular windowing advantages.[3]

M technology is standing on the threshold of a brave new world. M developers now have the tools to make or break M as a language and technology. For those of us ready to step across the threshold and embrace this technology, one which experts say mainstream applications will have to utilize to be successful, great benefits await. *Carpe diem*—seize the day. **M**

Guy Gardner and Thomas Salander gave invaluable input and direction for this article. Also, thanks to the MWAPI subcommittee and others on the MUMPS Development Committee (MDC) who made MWAPI possible through their tireless work. And special thanks to Micronetics Design Corporation, Digital Equipment Corporation, and InterSystems Corporation for use and review of beta MWAPI documentation and/or software.

## REPRINTS ARE AVAILABLE!!

Please call the MTA office at  
301-431-4070 for details.

## Advertiser Index

We appreciate these sponsors of the February issue, and all the companies who support the M community with experience, ability, and talent.

Amet .....	3
CoMed .....	37
Cue Data Services .....	6
CyberTools, Inc. ....	17
Data Methods, Inc. ....	55
Digital Equipment Corporation .....	cover 4
Educational Systems, Inc. ....	48
Globalware .....	55
Greystone Technology Corporation .....	7
Henry Elliott & Company .....	27, 39
InterSystems Corporation .....	53
J.J. Althouse & Associates .....	40
KB Systems, Inc. ....	29
Micronetics Design Corporation .....	cover 3,30-31
MUMPS AudioFax .....	25
Polylogics Consulting .....	12
SciCor, Inc. ....	17
Sentient Systems .....	5
Software Technology Services .....	15
Systems Automation Technology Ltd. ....	8
Vista Hill Foundation .....	52

*This index appears as a service to our readers. The publisher does not assume any liability for errors or omissions.*

Gardner S. Trask III is a senior programmer/analyst for Collaborative Medical Systems in Waltham, Massachusetts, and owner of MPower Computer Consultants in Beverly, Massachusetts. He is active with the New England M Users' Group and board member of the City of Beverly Computer Commission. He has a master's degree from Cambridge College. He is also the system operator of the NEMUG Bulletin Board System.

## Endnotes

1. MWAPI was produced by the MUMPS Development Committee (MDC), the ANSI X11 committee responsible for the MUMPS standard, and is currently in the ANSI approval process.
2. Microsoft Programming Series, *The Windows Interface: An Application Design Guide* (Penguin Books, 1992).
3. G. Lundeen and C. Tenopir, "Text Retrieval Software for Microcomputers and Beyond: An Overview and Review of Four Packages," *Database* (August 1992): 51-62.
4. MDC Subcommittee 11, M Windowing API, X11/SC11/TG4/WG6/93-17 29-June-1993.
5. G. Gardner, "Peeking at the New M Windowing API," *M Computing* 1:2 (April 1993): 13-25.