Just Ask!

Question: Last year at the Annual Meeting I saw several presentations about the new standard. I'm still waiting for my vendor to announce a release date for features we all saw running in Washington. What's taking so long?

Question: Some features in the language have been available for years from the vendors yet *still* are not standardized (e.g., saving a routine). Why is the standardization process so slow?

Editors: While these two questions sound mutually exclusive, either or both can ring true, depending on your point of view. Implementors may appear slow to start work on a new feature, or they may demonstrate a prototype and then take months (or years) to make the product available. By the same token, the M community can recognize the need for a new feature and implementors may introduce it long before the standard includes the feature.

Let us look at the current MUMPS Development Committee (and proposed ANSI) standard. As this is written, no implementor has all the features specified by the new standard available in a commercial product. The perception that the implementors are taking a long time to bring out conforming implementations is correct. They are taking a long time, but with good reason.

The 1984 and 1990 revisions of the ANSI standard were mostly incremental improvements. Features, such as new functions and command constructs, were added. Generally, the changes needed to an implementation were just that: additions that did not require a change in the underlying design of the system.

The current standard is completely different. Several new features are not just add-ons, but may require a fundamental rework of the underlying implementation. Transaction processing affects the heart of every M implementation: long-term data storage. The requirement that changes to several globals be "undo-able" and fault tolerant may call for a complete rewrite of the way a system accesses those globals.

Another new feature, error-processing, may require a rethinking of the "stack" and how the implementor compiles or pseudo-compiles the code. In addition, most implementors try to provide backward compatibility with their existing extensions. How does an implementor provide standardized error-handling and, at the same time, support older code that may be using proprietary Z functions and Z commands? Changing a system this much is orders of magnitude greater than the kind of change needed to add the $GET function, for example.

Another and most timely example is adding a completely new standard: the M Windowing Application Program Interface (MWAPI) (see "The M Windowing API: The Tools," by Gardner Trask III, in this issue). While not part of the basic M language specification (ANSI X11.1), customer needs have driven the implementors to pursue implementation of this additional standard. The specification is larger than X11.1. By some accounts, the implementation requires a level of effort comparable to implementation of a complete new M system without the database engine. At least one implementor has described the work as *more* than the effort required to implement the original language standard in 1977.

All of these efforts, along with many other changes (Open MUMPS Interconnect or OMI, GKS, SQL, internationalization, plus language enhancements), are going on simultaneously. Most implementors have decided to put more effort into one or two areas first and get those working, before moving on to another task. This is why all the new features will be demonstrated at the M Technology Association's Annual Meeting in Reno (June 13-17), but they may not all be running on the same system.

The problem with getting a feature standardized is similar to the problem with getting it implemented. While the feature may have been simple for a particular implementation, that does not carry over to all implementations (a real consideration when writing a standard), nor is it necessarily simple to specify. More than once a developer has been frustrated trying to write the language to specify a feature, *even though its own system already had the feature running!*

Add to this the complications of doing work through a committee. With some implementations, one programmer may add a new feature without any input or knowledge of other developers (often, it seems, in the middle of the night). But when writing a standard, the committee manages the

MUMPS Office Automation and TOOLS

Data Methods Packages feature easy integration with one another and with your MUMPS applications. Immediate links to major packages are also provided including FileMan, MailMan, Kernel and others.

WORD MANAGER ™	A full-featured word processor with spelling, powerful formatting and numerous features for all types of documents.
FORMS MANAGER ™	A complete forms design, data entry, editing and printing package. A front-end to applications packages including FileMan.
SCRIPT MANAGER ™	A total medical transcription solution featuring glossaries, medical dictionary, and sophisticated management functions.
CALC MANAGER ™	A complete spread-sheet package with all the features and functions of popular PC based packages.
REPORT GENIE ™	A flexible, powerful and easy-to-use report generator with three different interfaces to fit every users needs.
GRAPH MANAGER ™	Business and scientific graphical package supporting many printers and plotters.
VIEW MANAGER ™	This package features: Online free-text search,view and print functions, with an intuitive interface combined with powerful features.
MEDICAL DICTIONARY	A complete medical dictionary - compatible with our software or yours.

PROGRAMMERS AND RESELLERS Data Methods products are also available as functional modules for programmers and in quanities for resellers. Special license arrangements and complete technical support provide an easy, low-cost path to full integration with your MUMPS software.

Data Methods

Data Methods Incorporated
63 North Broadway
Nyack, New York 10960-2636
(914) 353-2000
(914) 358-6456 FAX

specification process, and two-thirds of the committee must agree with the specification before it is incorporated into the standard. The resulting specification may bear little resemblance to the original proposal. (Many who have been in on the process refer to it as the Sausage Principle. Ask an MDC member what this means.)

Sometimes the need for a particular feature is questioned. An implementor may provide a feature at the request of a client and then bring the feature forward as a proposed standard. The committee must determine if this is a generally useful feature, or if it has only limited appeal. Is this consistent with the general approach in M? Is it implementable across all M systems, or does the proposal assume or imply a particular environment (hardware, operating system, etc.)? Is there enough interest within the committee to pursue working on the functionality?

This last point is important and mirrors the concerns implementors express about resources. The MDC has a limited number of members who volunteer time and effort. While a particular functionality may be generally acceptable in all other respects, unless at least one person within the committee is willing to develop a proposal (and, consequently, take time and effort *away* from some other proposal or the real job that pays the rent), a proposal will languish and die.

All this effort is just on the functionality, not the actual specification. While the committee may be in complete agreement on the functionality, agreeing on the *syntax* may be a long (and bloody) process.

For all these reasons, the processes of standardization and of implementation may take much longer than those of us not directly involved, waiting for the end result, think it should. Remember that it all works together. Sometimes an implementor will develop a new feature and bring it forward for standardization. Sometimes the MDC will specify a new feature or approach. Sometimes it comes from the user community. Eventually, it all comes together in a product we can use. **M**

Send *Just Ask!* questions or requests to the managing editor at *M Computing*.