# Where Is the M in POSIX?

*by Chris Richardson*

## POSIX API Efforts

In the efforts to define standards for programming languages, operating environments have been allowed to remain proprietary. The POSIX (Portable Operating System Interface) standard attempts to standardize the operating-system environment. POSIX is derived from the common portions of the dialects of UNIX by the Technical Committee on Operating Systems and Application Environments of the IEEE Computer Society. Frequently, an application needs to request the operating system to perform system services. To ease the interface between different programming languages and the operating system, a series of subroutine and function calls needs to be defined. These calls are referred to collectively as an Application Program Interface (API).

The MUMPS Development Committee (MDC) is acting as a consultant to the IEEE Computer Society, which is working toward a standard for the POSIX M[UMPS] API. This article briefly describes many features that MUMPS programmers can look forward to when this particular standard is approved.

Most POSIX APIs (FORTRAN, COBOL, and Ada) are derived from a series of calls derived from the POSIX C API. The calls are modified as little as possible to enable easy implementation by the vendors. The POSIX M[UMPS] API specifies a series of calls that are derived from the same POSIX C functions.

M, in some older implementations, was not only the language but also the operating system. It already provides many of the system services to the application environment just as many other operating systems. A portion of the functionality requested to comply with the POSIX standard is already ensconced in the M language. This is both a blessing and a curse. There are additional functionalities that have no current analog in the M language (or in some of the other language APIs). These API functionality topics are described briefly in this article.

## Issues and Services

Producing a POSIX M[UMPS] API encompasses a variety of issues, which need to be addressed. While many of these issues are services, they are still issues with implementing a POSIX API. Some issues, such as data structures, are difficult for other environments (FORTRAN and COBOL) to apply in a standard way; the M environment has less of a problem. The issue becomes one of mapping the dynamics of an M data structure to the more static C structures to provide the services described.

## Data Typing Definitions and Conventions

Many of the definitions and mechanisms are applicable to the POSIX M[UMPS] API. The major difference is found in the definition of data types. While languages with defined APIs (C, Ada, and FORTRAN) or in review (COBOL and Modula-2) do have data types, M only has the single data type, strings, and extremely dynamic memory allocation. The calling interfaces of the POSIX functions involve the passing of data via arguments. The C functions require different variable types, integers, characters, pointers, and floating-point scalars and arrays of various magnitudes (8-, 16-, 32-, or 64-bit sizes) as well as C structures. Different platforms conform to either big endian or small endian (byte order of the word) constraints. These constraints are not a problem within the M environment. The standard M code will seldom have a problem with these constraints. The conversion of arguments and constraints is defined within the POSIX M[UMPS] API. This external API library is available to the M application, and maps the values to the external data type and arrangement of the arguments.

## Conformance Requirements

The conformance of an implementation to the API standards, as identified by the first POSIX C API, should make a statement to describe the deviations. The M[UMPS] API will have a few of these deviations because of the M operational environment, which identifies only a single data type, strings. The POSIX M[UMPS] API defines the conversion of the M string value to the more platform-specific representation required by the POSIX side of the interface. As mentioned, the word size, endian configuration issue is a difficult barrier to general portability. Applications in compiled languages need to be recompiled in order to run in a new environment.

### *Mapping Rules*

The mapping issues involve the naming conventions as specified by the POSIX C API. The POSIX M[UMPS] API defines

the conversions that the calls will make to the external environment. Once these interfaces are well defined in the POSIX M[UMPS] API, the M implementor's job becomes easier to provide the POSIX library or POSIX library's conformance. On different platforms the word sizes differ: An integer or floating-point word can be 16 to 64 bits, depending on the system. In M, the size is never defined and is converted to that size required by the underlying POSIX M[UMPS] API. The M applications developer can still build platform-transportable code.

## Configuration Services

Essential to a flexible operating environment is run-time knowledge of the operational environment in which an application runs. The more information an application "knows" about its environment, the better it can react to the situation within the system. Furthermore, self-awareness enhances fault tolerance and improves applications that can adapt to the current state of the environment in which they are working.

## Process-Related Services

The M environment has performed well in this set of services. The $JOB identifier is specific to the M process, but may not be the same identifier that the operating system assigns. These services provide the process identification, as well as the user and parent-process identifiers. The system name and the concept of a group identifier are introduced along with environment variables, terminal identifiers, and configurable system variables.

### *Timing Services*

One characteristic of a POSIX system is the ability to respond to real-time activities. M systems have not been used in this capacity because of the inherent database-centered applications environment. M is usually more concerned with mediating access to data than with waiting for asynchronous external activity. With the POSIX M[UMPS] API, it will become easier for applications to provide many of these external activity-monitoring services without impacting the database environment.

### *Directory Services*

What files are currently available? How big are they? Where is a specific file and in what directory? Some M environments have had these directory services in one form or another. This has meant that applications need to be tuned to the particular environment to access the information in the same manner in which a particular vendor provides it. The POSIX M[UMPS] API will make that interface much easier to generalize and implement these types of services.

### *File Services*

One challenge of most M systems has been the availability of a set of consistent sequential file services. Some older M systems provided a reserve area or tape for the generation of spooled reports or transport. This new interface provides the capability consistently to create, read, write, spool, submit, or delete files in a more transportable fashion.

### *Input-Output Services*

The ability to read and write information with files and processes is critical to most data-processing operations. The POSIX M[UMPS] API provides a standardized method of performing interprocess channel creation (pipes), file opening and closing, file input, output, and file positioning.

### *Synchronization Services*

Traditionally, complex M applications have employed a number of mechanisms to synchronize the operations of separate applications. These mechanisms include locks, globals, and file or device availability. The POSIX synchronization will enable M applications to be synchronized with other language applications. It should be possible to provide external synchronization for other language applications to call into M applications and return data in a near real-time manner.

### *Shared Memory and Message Services*

Physical memory allocation issues have been avoided by the M community. All POSIX language APIs provide some form of shared memory and message services. This is a characteristic that enables the user to lock and unlock memory, and communicate via this shared memory with other entities. This is valuable for real-time operation, object-oriented processes, messaging, and message queuing.

### *Scheduler Services*

Process scheduling has been implemented by many M applications. File Manager uses Task Manager to perform these background tasks. The ability to establish scheduled services with the surrounding operating system enables reaching beyond the M environment and thus carrying out additional services. An M environment actually may set up a backup procedure that shuts down the M environment, performs the backup, and then restarts the M environment without operator intervention.

*continued on page 31*

*continued from page 29*

### *Execution Thread Services*

The set of execution services makes it possible to extend task-forking beyond the M environment. Rather than just JOBbing off other M applications, M applications could initiate other language processes to perform specific tasks separate from the current task.

### *Security Services*

The POSIX environment provides Access Control Lists (ACL) to ensure system security. The ACLs offer a level of service access-security and come in two components: one is the list of identities of accessors to the system (individual user, user group, other networked systems, and so on) and the second is what types of access are allowed (read, write, query, execution), that will be applied to a file entry (permitted actions). Additional security services are usually provided by other means outside the operating system, such as network firewalls and gateways.

## Database

Most languages have this section in their APIs, but few have much to say concerning their database interface. M has a database intrinsic to the language. The M definition for this section will be a strong bridge for the interoperability between other database systems. M

R. Chris Richardson is a software engineer with Science Applications International Corp. in San Diego, California. He heads MDC's task group on data structures. In addition, he wrote for *MUG Quarterly* (a forerunner of *M Computing*) on the hypertext model and on vector implementation within the MUMPS environment.

## Endnotes

The reader may want to refer to the following selected standards and technical materials for a more detailed understanding of the material presented in this article.

IOS/IEC 9945-1:1990 (E) - Information Technology - Portable Operating System Interface (POSIX) - Part 1: System Application Program Interface (API) [C Language].

IEEE Std. 1003.9-1992 - POSIX FORTRAN 77 Language Interfaces, Part 1: Binding for System Application Program Interface [API].

ISO/IET JTC1/SC22 - Working Draft on: Information Technology - Programming Languages - Modula-2 Binding to POSIX.

British Standard Implementation of ISO/IEC DIS 14519-1 -Information Technology - POSIX Ada Language Interfaces - Part 1: Binding for System Application Program Interface (API).

ISO 7498-2, Information Processing Systems - Open Systems Interconnection - Basic Reference Model.