$%SQRT^MATH than there would be for SET X=$$SQRT^MYOWN, and the library function will typically be more accurate than a function written in M[UMPS] would be.

Now, in M[UMPS] we could always abbreviate everything to its first letter; can we still do that with those new library functions? No, it wasn't possible to define the library functions in an unambiguous way and retain this option. Both the name of the function and the name of the library will have to be spelled fully.

A first at this MDC meeting was a set of responses by the Interpretations Task Group. This group was created to answer questions about the standard. The notation and language in the standard have evolved to the point where certain parts are no longer easy to understand for everyone. If ever you don't understand what is intended in the standard: "Just Ask!" The Interpretations group is here to answer.

The questions this time are:

- In counting the number of characters in the name of a global variable, do you include the leading ^ sign? No, that character doesn't count toward the maximum number that is portable.
- For values between 1 and +1, should the two-parameter form of $FNUMBER produce a leading zero, just as it does for the three-parameter form? No, that zero only appears when the third parameter is specified.
- What is the sign of 0 (zero)? Is zero negative or positive? Zero does not have a sign. It is neither positive nor negative.
- How is the naked indicator affected in the context of SET $EXTRACT(^D(4,^A(1)),1,^B(2))=^C(3)? The references that affect the naked indicator are, in sequence: ^A(1), ^B(2), ^C(3) and ^D(4,a), unless the value of ^B(2) is less than 1, in which case the reference to ^D never occurs.
- And, finally, what is the meaning of the term *standard* in the definition of library functions? The term *standard* was introduced to simplify (sic!) the description of the values that a function can return. It means that the possible values are all that are typical for the data interpretation (tangent would return a *standard* value, sine is limited: $-1 \leq$ value $\leq +1$). M

If you have questions in need of answers, forward them to the managing editor of *M Computing*. They will be handled by the Interpretations Task Group or in this magazine.—*Editor*

Ed de Moel is the new chair of the MUMPS Development Committee. He is with SAIC and works on the tools for performance monitoring. His e-mail address is demoel@fwva.saic.com.

FOCUS ON FILEMAN

FileMan Version 21: DBS Record Selection and Criteria

by Rick Marshall

In Part 1 of this series we explored the conventions of File Manager's new API (application programming interface), the database server or DBS, designed to make effective use of the language features of the 1990 M standard, and to begin using the 1995 standard. Taken as a whole, the new conventions described in that article—parameter passing, file numbers, passing closed array names by value, IENS (Internal Entry Number String) values, standard arrays and flags, the Language and Dialog files, and the FDA (Filer Data Array) structure—help form a new method of communication between FileMan and applications. This new system of communicating opens the door to multilingual applications, graphical user interface, non-M environments, other new user interfaces, and even other new APIs, any of which can use the DBS calls to access FileMan databases. We continue now with our survey of the DBS by exploring when and how to use the calls it provides, in this Part 2: Record Selection and Criteria.

Accessing the Data Dictionary

In the past, the user interface has monopolized the examination and ma-

nipulation of data dictionaries (DDs), through options such as List File Attributes. Writing code capable of making reliable decisions based on what it finds has been difficult and risky. The DBS begins to address this reliability problem with a set of four calls in version 21. Version 21 barely begins to open this doorway to the data dictionary, with many versions ahead needed to complete the process. Your input will help us prioritize the features you most need to access, but all the future work will build from the foundation laid here in version 21.

The DBS's DD retrieval calls come in two pairs of calls: one oriented toward returning information about whole files, and the other about specific fields. In each pair, one call actually gets the information for the caller, and the other returns a list of pieces of information the first call can get for you. The four calls are `FIELD^DID` and `FIELDLST^DID`, and `FILE^DID` and `FILELST^DID`, all of which as DBS calls accept parameters (as briefly described in the previous article in this series; please see the November 1994 issue).

Programmers usually do not need to inspect the DDs in their applications, but when necessary, they have resorted to integration agreements with the FileMan team to ensure that their direct access did not cause problems later when the team changed the DDs underneath them. The four new calls give developers a much-needed level of abstraction for handling special situations, protecting them better than integration agreements have done.

Finding Records

The developer's first major challenge is selecting the record to display in the application's main window. While the classic FileMan API includes several record-selection calls, each of them will issue `WRITE` commands in some situations. It is especially difficult to control the various programming hooks that contain embedded `READ` and `WRITE` commands: Poor control can result in the classic API properly returning part of the information, with another part of it trailing off to never-never land instead of leading back to the application's window.

The Finder call, `FIND^DIC`, solves much of the problem. As with all DBS calls, it passes back all information through standardized arrays that the application can manipulate into a nice window display. The developer can select nearly any window gadget in order for the user to pick a record, interpret the results of the selection to produce a text value, and pass the result to the Finder for the actual database lookup. Of all the DBS calls, the Finder solves the widest number of record-selection problems for our would-be GUI-application developer. The DBS even has an extrinsic function version of the Finder, `$$FIND1^DIC`, to handle unambiguous lookups.

Listing Sets of Records

One class of window gadgets presents difficulties for the Finder—listing gadgets. For the user to choose from among list boxes, list/entry boxes, and drop boxes, the developer must show the user the available choices in advance. The Finder cannot initially populate these gadgets because it needs a starting value from which to work. Therefore, a different tool, the Lister, supports these kinds of gadgets.

The Lister runs on any regular (or what looks to be regular) FileMan index, and will return part or all of the list of records cross-referenced in that index. The list can be generated in forward or backward order, restricted to unique starting and ending values or to a partial match with a value, and even resumed where it was interrupted from a previous Lister call. So, the listing gadget will show what the user needs to see. Between the Lister and Finder, the programmer has the necessary tools to attach a live FileMan database to a window used to select records.

Space-Bar Recall

Another specialized call completes the record-selection tool kit. FileMan users take advantage of a specialized user-interface feature called space-bar recall. At any record-selection prompt, the user taps the space bar, rather than typing an actual value; FileMan interprets this as "Pick the same record I picked the last time." This valuable feature saves users many keystrokes, and the Finder properly handles the space character as an input value.

The Finder, however, cannot save selected values for later retrieval using space-bar recall because it cannot know if the programmer calls it with a user-entered value or with some other value. Because only user-entered values should be saved for subsequent space-bar retrieval, the programmer should call `RECALL^DILFD` to make FileMan remember which records the user selects. The programmer should call `RECALL^DILFD` as soon as the user picks a record, probably as part of the callback associated with the selection gadget, to keep the valuable space-bar recall feature as the application migrates to a GUI platform.

The Tools to Create New Records

Sometimes the user does not want to work with an existing record, but instead wants to create a new one. The DBS supplies a GUI-compatible call to do just that: the Updater.

The programmer tells the Updater in which file to create the new record, and may pick the record number to use. Because some files lack a single, unique field capable of identifying individual records, the Updater needs to be fed the information for all the new record's Required Identifiers; if the programmer calls the Updater without supplying it with these field values, it will reject the call. If during application development the programmer forgets which fields a file requires, `FILE^DID` will supply the information. This feature demonstrates some of the synergy between the various DBS calls.

Therefore, GUI applications need to prompt the user for all required identifier fields before calling the Updater. The DBS supplies the Data Checker call, `CHK^DIE`, to check the values for fields in a record that does not yet exist, as the programmer needs to do when creating new records. This call cannot provide the sophisticated checking possible when a record exists, but it ensures a match between the basic syntax of each field and the user's entry for the fields in the window.

The Steps to Create New Records

To use the Updater and Data Checker together for GUI-based record creation, then, follow these basic steps.

1. As a programmer, you need to present the user with gadgets to set any required identifier fields whenever the user starts to add a new record.

2. The callback for each field should pass the user's value through the Data Checker, handle bad input appropriately, and load valid data into a Filer Data Array, or FDA, node. (See the previous article in this series or the FileMan documentation for more information on how to do this.)

3. Only after the user has entered valid values for all the new record's required identifiers should you pass the FDA you've built to the Updater, which will then add your new record and pass you back the record number.

4. Armed with this record number you can then proceed with any additional data entry you wish for this record.

Things to Remember about New Records

Remember these important details about the Updater. First, although it can file data for more fields than just the required identifiers, it is not as effective as the Filer, the best tool for editing. The Updater's data-validation abilities are not as strong as those of the Filer: Use the Filer for everything beyond the required identifiers.

Second, the Updater can handle more than one record at a time, but it does not do so efficiently. Version 21 behaves perfectly with individual records, but FileMan 22 probably will include a Batch Updater optimized for creating many new records at a time.

Third, the Updater has limited, built-in lookup capabilities. FileMan's classic API supported record creation through a feature known as "Learn As You Go," or LAYGO, in which the record-selection calls automatically would add a new record with the user's permission if FileMan could not find a match with the user-entered value in the file.

The Updater has a limited version of this popular feature, which the developer can ask to create a new record only if it can't find an existing one to match. The Updater's LAYGO features use a limited subset of FileMan's lookup capabilities to perform the matching, and can handle simple situations in one call. Sophisticated LAYGOs require the programmer to make separate DBS calls, first to the Finder, and then to the Updater if it cannot find a match. Future versions of the DBS will incorporate such LAYGO features in a single call.

Retrieving and Editing Records

Part 3 of this series will explain how to use the DBS to get data from records for the user, to help in filling in those fields, to validate what the user enters, and eventually to file the updated record back into the database. The DBS for FileMan version 21 supports the basic tasks to create a GUI entry-and-editing application built around a FileMan database. As developers of FileMan-based applications around the world continue migrating to GUI and client-server software, the DBS will evolve to solve the next generation of problems. Eventually, the DBS will provide a complete API to the FileMan database, which lets programmers put their own interfaces onto this powerful database management system.

Forward your FileMan questions or topics you would like to see addressed in this column to G.FILEMAN DEVELOPMENT TEAM@FORUM.VA.GOV, or write to VAISC6-San Francisco, Suite 600, 301 Howard Street, San Francisco, CA, 94105.

Rick Marshall works at the Seattle office of VA's San Francisco Information Systems Center. He writes and programs for the FileMan development team, teaches the Kernel class at MTA's annual meeting, and works in MDC as the editor of the 2001 M Standard.