Sneak Preview: FileMan Version 21

by Rick Marshall

Part 1 of 3: The Database Server

We interrupt our regularly scheduled investigation of FileMan's programming hooks to bring you this preview of FileMan version 21. The next release, due in early 1995, brings so many improvements we cannot cover everything in this column. This month's *M Computing* carries the first of three columns devoted to this very special topic, and focuses on the database server (DBS), a new programmer interface for FileMan.

Concept

Classic FileMan benefited from the ease with which M handled input/output and database access. FileMan's developers could fold these activities in with general processing easily, assisting in the rapid creation of sophisticated database tools. Now, fifteen years of computer science breakthroughs reveal the many benefits we can reap if we separate our user interface from database access. Client-server, GUI, access through non-M systems, object orientation, and independence from any single database interface all become possible with this conceptual shift, which lies at the heart of FileMan's new DBS.

The DBS consists of a suite of new programming calls that function independently of FileMan's classic API (application program interface). These new calls take advantage of all the advances of the M standard made over the past ten years; for example, using the NEW command, parameter passing, and extrinsic functions have allowed a degree of modularity not previously possible, permitting expanded use of reentrancy and recursion. These calls contain no user interface; silent to the users, they communicate only with the calling application. Finally, these calls wrap up database access at a low level, providing the basic engine that underlies Data Definition Languages, Data Manipulation Languages, and Query Languages in traditional database theory. These essential ingredients to the DBS throw the doors open to people wanting to write a DDL, DML, QL, or user interface of their own, and to the FileMan team itself, which plans to take advantage of the DBS in exploring new database models and features in future versions of FileMan.

Parameter Passing

Silent only to the users, the DBS speaks to the calling application with a broader and deeper vocabulary than the classic API, with parameter passing as the application's basic voice for speaking back. Introduced by MDC (MUMPS Development Committee) in the 1990 M standard, this feature of M wraps every DBS call in a strong modular envelope. Excepting only basic state variables such as DUZ, applications explicitly pass all needed information through parameter lists. To make these calls easier to master, all DBS calls share common conventions about the order and value of parameters.

For example, applications declare files by passing file numbers, not roots. This lets the DBS calls use their understanding of the file's attributes to save the application work later on. Since this means applications will now have file numbers in their symbol table more often than file roots, the FileMan team has provided the new DBS function `$$ROOT^DILFD` to return a file's global root, given the file number and internal entry numbers needed.

As another example of parameter passing, applications will pass arrays by reference only when the DBS call expects short local arrays. Since the M standard does not allow passing globals by reference, to pass arrays of arbitrary size or location the application passes the name of the array by value; the DBS call will then use the name of the array with subscript indirection to access the actual array values. The DBS documentation always will declare how the calls expect each array to be passed.

A final key example, array roots are always passed closed, rather than open. This brings FileMan more into line with the rest of M (which uses closed roots in `$ORDER`, `$QUERY`, etc.), and encourages the use of subscript indirection, a powerful but little used feature of M. Since applications will now be using open roots for the classic FileMan API and closed for the DBS, we have provided the functions `$$OREF^DILF` and `$$CREF^DILF` to convert between the two kinds of roots.

The Internal Entry Number String

The parameter used to identify entries in files deserves special scrutiny. FileMan has always relied on arbitrary (or sometimes not so arbitrary) record numbers to uniquely identify

records in files. Also known as entry numbers, internal entry numbers, or IENs, these numbers are adequate to identify an entry in a top-level file, but not in a subfile. Identifying this kind of entry requires not just the IEN for the subfile, but also the IEN for the file that contains the subfile. Since the subfile's parent file may also be a subfile, FileMan may require several such numbers to identify a unique file entry. Classic FileMan uses two different conventions to represent these sets of IENs. In the first, a series of separate variables holds each IEN: D0, D1, D2, etc. In the second, the various nodes of an array hold each IEN: DA, DA(1), DA(2), etc.

Both of these schemes, while adequate in most contexts, suffer from two related problems in special situations. Because these methods keep the IENs split out as separate values, testing two such sets of numbers for equality is not as trivial as it should be, and using this information in a subscript for sorting purposes results in an arbitrary number of subscripts.

The DBS solves these problems by using a new convention for representing file numbers: the Internal Entry Number String (or IENS). This concatenates together the various IENs into a single string, with each IEN followed by a comma, and the order of the numbers with the deepest level IEN at the left, and the top-level IEN at the right. This scheme always results with a trailing comma as the last character, an important detail since it ensures all IENS values sort as strings, a detail usually forgotten several times by programmers first learning about IENS values. An example of an IENS for a New Person file entry might be "9," while that of an entry in the Menu subfile of the Option file might be "4,67,". The two DBS calls—`$$IEN^DILF` and `DA^DILF`—let the programmer convert between DA arrays and IENS values.

Certain DBS calls use variations of the standard IENS. Calls that deal with the whole file, rather than a single entry, leave off the left-most IEN, so that the IENS starts with a comma. The remaining IENs then uniquely identify the subfile, but not any particular entry within it. The Lister and Finder calls use this variation on IENS values, for example ",67," to identify a specific Menu subfile within the Option file. The Updater, on the other hand, which often deals with records that do not yet exist or have not yet been identified, uses place holders to stand in for specific IENs in the IENS, where the place holders uniquely identify the undetermined entry and also indicate whether to add the entry or locate it. For example, "?1,67," might represent an entry that will be found in a specific Menu subfile of the Option file, whereas "+2,67," would represent a different entry that will be added to that same subfile. The DBS documentation for each call describes these kinds of variations.

Standard Arrays

Although the DBS uses parameter passing to speak to the application wherever possible, database activities often return too much data to pass through the parameter list, forcing the DBS to fall back on communication through named arrays. Although each call documents when this will occur, the DBS always handles three kinds of information this way: errors, help, and text. Any DBS call able to generate this kind of information will include a parameter that lets the application name the array where the call should return the information. If the call needs to return information in this array, it will set a specific local variable that the application should always check for after the DBS call. `$G(DIERR)` means the call returned error messages, `$G(DIHELP)` means it returned help text, and `$G(DIMSG)` means it returned some kind of general message text.

CALENDAR

January 16–20, 1995

USENIX Winter 1995 Technical Conference, Marriott Hotel, New Orleans, Louisiana. Call 714-588-8943 for details.

January 26–29, 1995

MUMPS Development Committee meeting, Sheraton Inn Albuquerque (Old Town), Albuquerque, New Mexico. Call 301-431-4070 for details.

February 13–16, 1995

Austin Information Technology Conference for U.S. Department of Veterans Affairs employees. Register through the Austin HELPDESK at 512-326-6780.

February 14–17, 1995

Client/Server West Conference and Exposition. San Jose Convention Center, San Jose, California. For registration and information, call CMP Trade Show Services at 1-800-808-EXPO.

April 2–5, 1995

7th Annual National Managed Health Care Congress (with MIS sessions), Sheraton Washington/Omni Sheraton Hotels, Washington, D.C. To register, call 617-487-6700.

July 23–27, 1995

International Medical Informatics Association Medinfo '95. Vancouver Trade and Convention Centre, Vancouver, British Columbia, Canada. For information, write Medinfo '95 Administration Office, Suite 216, 10458 Mayfield Road, Edmonton, Alberta, Canada. Phone: 403-489-8100; fax: 403-489-1122,

The DBS returns the text itself in a standard structure within the named array. For example, if the application asked such information to return in TEXT(42), the DBS would put errors under TEXT(42;"DIERR"), help text under TEXT(42,"DIHELP"), and general messages under TEXT(42,"DIMSG"). Each block of text gets its own numeric subscript after that, followed by the documented structure used to return the information. For example, if a DBS call generated three errors, TEXT(42,"DIERR",0) would show the count, and the 1, 2, and 3, nodes would separate off the three errors.

If the application chooses not to reroute this information, the DBS will return it in ^TMP($J,"DIERR"), ^TMP($J,"DIHELP"), and ^TMP($J,"DIMSG"). The DBS call CLEAN^DILF will clear out these three trees, as well as the three local flags, when the application has finished processing this information.

Dialog

The DBS generates the text returned in these standard arrays through the use of two new files: the Language and Dialog files. The local variable DUZ("LANG") specifies the language the user understands, by referencing one of the languages in the Language file. This file also describes how each language displays dates, numbers, and other language-dependent entities. The Dialog file, on the other hand, contains numbered entries, each holding the text of a specific message. Each message may include parameter windows for values to fill in at run time, and each entry describes the significance of the parameters. In addition, the Language file holds a multiple for versions of the same text in other languages, with a pointer to the Language file binding them together. At run time, DUZ("LANG") indicates which version of the message to return. All standard text returned by the DBS—errors, help text, and general messages—originates in the Dialog file, as does the most commonly used text in classic FileMan. All major DBS calls will return any of a broad spectrum of precise error messages that help the programmer or user understand exactly what has gone wrong, and the returned information gives programmers the option of coding to deal with specific problems as they arise.

Because the DBS executes programming hooks, application programmers need to help enforce the user interface silence. Rather than use WRITE commands in these hooks, the application should call the Loader: EN^DDIOL. The Loader will display text passed to it, except within the DBS, when the Loader will instead load the text into the message array and set DIMSG. This sleight of hand happens without any intervention by the application, and ensures that messages to the user travel through whatever route is needed to reach the actual user interface. If after a DBS call the application does need to simply WRITE the messages returned, the MSG^DIALOG call can do that.

The Filer Data Array

Whereas the DBS uses these three standard arrays to send messages to the application, the application and the DBS can use the Filer Data Array (FDA) to communicate back and forth about proposed changes to the database. FDAs can identify fields of entries within files and describe values for each. The meaning of an FDA depends on how the application uses it. An FDA returning from the Data Retriever call describes the current values of those fields. An FDA node returning from the Validator call acknowledges that the value is legal for that field. An FDA passed to the Filer call changes the database based on the contents of the FDA nodes. In general, then, an FDA identifies something about the field values for specific entries in certain files: either that the database looks like that, or should be altered to look like that.

The application can locate an FDA in any array, local, or global, at any subscript depth. The structures that make an array an FDA are the last three subscripts and the values of each node. The last three subscripts are: the file number, the record's IENS, and the field number. The value of each node is the value for that field of that record in that file. For example, the FDA node ^TMP($J,"EDIT",200,"9,",.01)="EINSTEIN,ALBERT" describes the .01 field of entry 9 in the New Person file (# 200) as having the value "EINSTEIN,ALBERT". Returned from the Data Retriever, this would tell the application that user number 9 is Albert Einstein. Passed to the Filer, it would tell the DBS to change user 9's name to Albert Einstein. To set up the FDA nodes for a DBS call, the application can either use the SET command, or call FDA^DILF with the right parameters. An even more convenient method likely to work in most GUI settings involves making a series of DBS calls that first return an FDA describing the DBS, modify the FDA as the user requests changes, and update the database with that FDA, so that the application never actually sets or kills an FDA node.

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 is a member of the VA FileMan development team.