FileMan Version 21: The Database Server

by Rick Marshall

Part 3 of 3: Editing an Existing Record

We continue now with our preview of FileMan version 21, released to the field earlier this year. This article concludes our examination of how and when to use the individual calls that make up the DBS (Database Server) by explaining how to use the DBS to display records to the user, provide help in filling in the fields, validate what the user enters, and eventually file the updated record back into the database. The DBS for FileMan version 21 was designed to emphasize these basic tasks involved in the creation of a GUI (Graphical User Interface) data entry and editing application built around a FileMan database.

Because the DBS focuses on the noninteractive parts of the process, the application must provide both the user interface and any networking or channeling capabilities. Although the DBS can provide database access for several distinct kinds of applications (such as generic client-server, linkage with non-M systems, and synchronization between parallel databases in different formats), this article will use the GUI example to demonstrate the DBS's use.

Retrieving Current Data

After the user selects a record using the DBS lookup calls explored in part 2 of this series (see *M Computing*, April 1995), the application needs to offer a window with data entry and editing gadgets prepopulated with the current data. Two calls in the DBS will help: the Data Retriever, GETS^DIQ and the Single Data Retriever, $$GET1^DIQ.

The two data retrieval calls are similar. They can return data in either internal (storage) or external (display) format, the former being most useful for computation and the latter for user interaction. Like most DBS calls, these use a full set of error messages to report problems with the call, and the application bears responsibility for detecting and responding to these error conditions when they arise. Each returns data from a single record.

Apart from those similarities, the structural differences between the calls result in distinct usage. Where the Single Data Retriever uses M's extrinsic function calling syntax to return the value of a single field, the Data Retriever returns any number of fields in an FDA (Filer Data Array) structure (see *M Computing*, November 1994). This makes the Single Data Retriever ideal for use within programming hooks, where the current convention of direct global access can make database conversions a delicate process requiring coordination among many different software packages. The Data Retriever proves more useful in full- scale database interface work such as the sample GUI editing session.

The 1995 (draft) M Standard brings transaction processing capabilities to the language. The FileMan team plans to use these capabilities after ANSI approves the standard, so the DBS calls assume a transaction-like interaction with the calling application. To fit this model, an efficient data entry and editing application will use two FDA structures: one to describe the data prior to editing, and one to collect and store the changes until the user decides to update the database.

The Data Retriever populates the first FDA for the application. To accommodate transactions involving a single logical record that spans multiple physical records, the Data Retriever adds each record's data to the FDA without first clearing the FDA. Because this lets data accumulate, calling the Data Retriever once per physical record will result in an FDA that describes the logical record. Once the Data Retriever has returned the complete logical record's current data, the application bears responsibility for presenting the data to the user and engaging in the dialog in which the user will edit the data.

Providing Help

FileMan's rich help capabilities have proven a key element in VA's (Veterans Affairs) software strategy, so FileMan extends those help capabilities to all applications built around the DBS. The Helper, HELP^DIE, lets the application fetch the help text as-

sociated with any data field. An assortment of flags lets the caller specify any level of help available from FileMan.

As with all user interface issues, the application bears responsibility for detecting a user's request for help. The VA FileMan team recommends that fields interfaced by text gadgets should permit the user to use the standard "?" and "??" input convention to request simple and more elaborate help. The Helper will return the help text in a standard array, which the application should then display to the user.

Note that the Helper does not provide the lists of entries usually displayed for .01, pointer, and variable pointer field help. The application should use the Lister, described in part 2, to generate these lists.

Validating User Input

An effective user interface should perform as much data validation as possible while the user edits the data values, rather than saving up all data validation for the end of the transaction. This takes advantage of the user's attention and helps correct misunderstandings and errors before they are propagated to other fields.

The Validator, VAL^DIE, takes a field value and either accepts or rejects it. If the caller uses the F flag, the Validator puts the accepted value into an FDA. This usage creates the second FDA used in transaction-like editing sessions. The Validator rejects values by returning error #701 (The Value is not Valid).

Aside from executing the usual screens and transforms in the course of data validation, VAL^DIE properly interprets the values "" and "@" as field value deletion and populates the FDA accordingly. It also rejects changes to uneditable fields that already contain data, and if the R flag is present, confirms that the record already exists. If the E flag is present, it also returns the external format of the validated value in a parameter passed by reference. The application can use this last feature to echo back the standard representation of a value that has multiple user input values.

The Validator does not handle input values beginning with "?", conventionally used to request help. The application must check for such values and handle them with calls to the Helper, calling the Validator only for the remaining input values.

In the course of an editing session, this second FDA, the one passed to the Validator, will accumulate data for the user's edits, resulting in an FDA typically shorter than the one used to populate the window initially.

Filing Edited Data

Should the user choose to cancel the transaction, the application has little work to do; the database has not been updated yet. To commit the transaction, simply pass the second FDA to the Filer, FILE^DIE, and it will file the prevalidated changes to the database.

With the E flag, the Filer can both validate and file data, but the technique described in this article results in a more efficient call. Should the Filer encounter error conditions, it will file what it can and return error messages for what it cannot. It will fire off all appropriate cross-references, will audit as appropriate, and will properly delete field values.

Use the Filer specifically for editing the data of an existing record. The Filer will not delete .01 fields, since this would result in record deletion. Neither does the Filer permit the creation of new records, either at the top level or in a subfile. Use the Updater to add records. FileMan 21's DBS does not include a record deletion call. The classic call ^DIK has been made reentrant for version 21, so for now, applications should use it for record deletion. Because this operation has not yet been integrated into the DBS, the application will need to use some application-specific data structures to track record deletion requests prior to committing the transaction. The Word Processing Filer, WP^DIE, permits more control over how word processing fields are filed, giving the application the option of appending new data rather than replacing it. The application needs to handle the various cases with the right call for the task involved.

This step, filing the changes, concludes and completes a data entry transaction, but the application needs to properly handle two other details of the process: serialization and invocation.

Locking the Record

The application must ensure serial access of its records. Because this kind of editing session spans multiple FileMan calls, only the application is guaranteed to have control at both the beginning and end of the process, so the application must bear responsibility for using the LOCK command to reserve the record during editing.

Use incremental locks in order by file number to lock the physical records that make up the logical record in use and decremental locks to release them. Lock the record, or a specific node within the record, not the whole file. Lock the records after selection, and release them after filing the results of the editing session.

This step prevents the possibility of having multiple users simultaneously

modifying the same record. Some developers skip this step in an attempt to hurry their applications to the field, but doing so ensures eventual database degrade. Give this application development step the priority it deserves.

Invoking the DBS

MWAPI interfaces can place calls to the DBS directly in the callback code, ensuring a fast, simple linkage between the GUI interface and the DBS. GUI interfaces built outside M, on the other hand, must provide some kind of linkage that allows invocation of the DBS calls by the external GUI system. In either case the GUI system must begin the invocation, but external systems need software to complete it.

VA has built a tool called the Data Broker, which manages and completes the linkage between an external system and the DBS. It converts requests into calls to the DBS and then takes the results of the DBS calls and converts them into formatted data for the external system. Although VA currently uses Delphi as its GUI system, the Broker can be used as the linkage to any external system.

Finally, efficient use of development time demands the creation of gadgets or data controls that package up the various details of starting and processing a DBS invocation. For example, the technology of a listbox will usually be combined with calls to the Lister and processing of the results, so applications should develop gadgets that bind up the standard code involved into a sharable, reusable gadget. Reuse of such gadgets will help standardize GUI interface conventions and speed up development time.

Conclusion

As developers of FileMan-based applications around the world continue migrating to GUI and client-server software, the DBS will continue to evolve to solve the next generation of problems developers face. Eventually, the DBS will provide a complete API (application programming interface) to the FileMan database, letting developers write their own interfaces to this powerful database management system. M

Forward your FileMan questions or topics 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 IRM Field Office. He writes and programs for the FileMan development team, serves on the Board of Directors of the MTA, edits the M Standard for MDC, and teaches at MTA's annual meeting.
