FileMan 22 and Indexes: Part 1

by Rick Marshall

Overview

The last column introduced FileMan version 22, but due to the volume of material to cover could not do so in depth. This column continues exploring this new version in more detail, starting with a focus on FileMan's index capabilities, old and new.

What is an index?

A file's index operates much like a book's. Just as without an index (or table of contents) you must search a book from page to page looking for the information you need, so without an index you must search a file from record to record. In both cases, the index lets you start with the value you're looking for and quickly identify the page or record number where that data is found.

Indexes are created and maintained by cross-references attached to specific data fields, so one of the most important decisions a file designer must make is which indexes to build. Like all good file design decisions, this should be based on 1) what the data represents, and 2) how it will be used. This article touches only briefly on these design questions, and focuses mainly on the kinds of indexes you the file designer can create.

Index Uses

Indexes can be used to sort entries by the values in one or more of their fields, and to select entries with certain data values. Since indexes store the field value in a subscript, M built-in collation rules perform the sorting for you. Selection, though, is more complex. Although you frequently want to let your users pick any entry in a file based on its field values, you may also choose to create indexes that include only a subset of the file's entries, to effectively hide the entries that do
not meet your criteria.

In either case, what the index does for you is store CPU time. That is, it spends the CPU time required to perform the computation (sorting or screening) as the data is created and modified, instead of when the user is ready to sort or select. An index does nothing for you that could not be done more slowly at the time the user is ready to sort or select entries.

Therefore, some of the crucial questions to ask when choosing which indexes to create are:

1) What reports run regularly enough or are for users powerful and impatient enough to warrant having index support?

2) How do you want your users to be able to pick entries in this file?

3) What kind of index support does your software need to make decisions on the fly as it executes? And

4) How much disk space can you spare to speed up these activities?

The answers to these questions are usually a pretty good guide to which indexes you should create. As you select each index to build, think through the answers to the question that you'll have to answer when you actually use FileMan to define the associated cross-references.

Index Structure

Not all cross-references build indexes. Some send bulletins, trigger values into other fields, or perform other activities, but this article shows you how to define cross-references that build indexes. Indexes share a common structural pattern.

First, FileMan stores them in the same global array as the data file they're derived from (but see Whole File Indexes below for a variation on this theme).

After the subscripts shared in common with the file's global root, the very next subscript is always the index's name, stored as a string subscript.

Next comes one or more subscripts used to store data values; these may be field values from the entries in the file, or they may be based on those values, or they may come from somewhere else entirely, but the general case is a field value. More precisely, the subscripts stored up to the first 30 characters of a field value (an older M standard restricted subscript length in earlier FileMan versions). With version 22 you can set this limit where you like for each of your data subscripts in each of your indexes.

After the data subscripts, the last subscripts in an index node store record numbers (or IENs), pointers to the entries in the file above that correspond to the data subscripts; except for Whole File Indexes (see below), most indexes have a single IEN subscript.

Finally, except for Mnemonic indexes (described below) FileMan index nodes always equal the empty string.

Here's an example familiar to those who work with the VA Kernel, the B index on the Name field of the New Person file. I've shown the file header, the relevant node of the entry, and its corresponding index node:

```
^VA(200,0) = NEW PERSON^200I^42^22
^VA(200,9,0) =
MARSHALL,RICK^TOAD^xxxxxxxxxxxxxxxxxx^@^^^
1^^1
^VA(200,"B","MARSHALL,RICK",9) =
```

Indexes differ by the number of data and IEN subscripts, by the contents of the data subscripts, and by the node value (Mnemonic indexes equal 1). These differences are due to the different types of indexes you can create, and each type of index follows a single pattern.

Regular Type Indexes

Most FileMan indexes are type Regular, which up until version 22 has meant the index has a single data subscript, (except for Whole File indexes on subfiles) a single IEN subscript, a value of the empty string, and an index node for every entry in the file that has a value for the cross-referenced field.

Version 22 extends this type to allow screening of entries, so Regular indexes need not include all file entries with values for the cross-referenced field. Version 22 also extends it to let you store a value derived from the field value in the index, instead of the exact field value itself. Version 22 also extends the Regular type to let you define subscripts not necessarily derived from any field in the cross-referenced file, to use any M code you like to create a data subscript value. These three changes will probably increase the percentage of indexes that are Regular, since they remove the need for some of the indexes out there created by M cross-references.

The B index of the New Person file shown above is a Regular index.

Index Names

Every cross-reference must have a name, and every index must have a name, and for all but M cross-references the names are the same (M cross-references can build indexes with any name they like, or need not build indexes at all. See below). By convention, the B index of every file is a Regular index on the .01 field, and is automatically created by FileMan when the file is created. Beyond that, two general rules apply based on how you use the index and who you are.

Index names encode their use. Indexes that will be used for lookup must have a name that collates after the letter "B", because that is where FileMan looks for lookup indexes. Those that will only be used for sorting must have names that start with the letter "A" followed by whatever you like.

Index names encode their origin. Indexes added to files developed and maintained by someone else must be namespaced. The reason for this is simple: the developers can put indexes wherever they like on their own files, so unless you're namespaced nothing prevents a new index in their next release from overwriting yours. This applies to 1) local indexes added to national files, 2) local indexes added to files developed and maintained at some other site, and 3) national indexes added to national files developed elsewhere.

Index Numbers

This is the famous prompt that when you entered a ? for help always responded NEVER MIND, JUST HIT RETURN." Actually, this is worth minding, because exactly the same guidelines for namespacing apply to numberspacing. Index numbers are significant, affecting both exactly where the index is stored in the DD and the order in which it is fired when the field changes. In your own files, use whichever numbers you like (usually, never mind and just hit return), but when adding indexes to someone else's files be sure to use your own numberspace.

Simple and Compound Indexes

Simple indexes have a single data subscript, which makes them, well, simple to understand and use. Compound indexes are those with two or more data subscripts. Up until version 22, none of FileMan's built-in index types allowed compound indexes, only simple ones, which led developers to use MUMPS cross-references to build their compound indexes. In fact, before this new version FileMan was unable to use a compound index for selecting file entries, which meant you could cause hard errors unless you stored your compound indexes among the sort-only indexes. Version 22, however, directly supports the creation of compound indexes for Regular and M types, so we expect this too to increase how often you can use a Regular type cross-reference to create your index.

Compound indexes require more thought on your part. For one thing, they will have nodes only for file entries that have nonempty values for ALL of the data subscripts, which means you must think twice to be sure you know whether your index will have a node for every entry. Another issue that deserves your consideration is which order to store your data subscripts, since that affects sorting, the order in which you must retrieve data to match against the index, and how quickly each data subscript cuts down your list of candidates when being used for selection. As a rule of thumb, if you don't care about the data subscript order, put the most unique value in the first data subscript. Name is a good choice; SSN even better; and Sex is a really bad choice (only two choices, usually).

The first file to have one of FileMan version 22's new compound indexes was the new Index file itself, because identifying an index requires knowing both the file and the index name. This is stored in the Index file's BB index as follows (I've abbreviated the short description):

```
^DD("IX",0) = INDEX^.11I^167^69
^DD("IX",.1101,0) = .11^BB^The uniqueness
index for ...

^R^^R^IR^I^.11^^^^^LS
^DD("IX","BB",.11,"BB",.1101) =
```

Whole File Indexes

Cross-references on fields in a subfile can be used to create indexes either just within the subfile under the parent entry, or to create larger shared conglomerate indexes containing subfile values from under all the parent entries. A typical example of the latter is to create an index on the top-level file, though if you're subfile is down more than one level you can create a shared index at any of the levels in between.

These shared indexes are called whole file indexes (for obvious reasons), and they are usually Regular type indexes. They are the only kind of index that can have more than one IEN subscript; in fact they must have one IEN subscript for each level of the hierarchical file spanned by the index. For example, a whole file index on a field in a subfile one level down will have first an IEN for the top level parent of the subfile containing the indexed entry, and then the IEN of the subfile entry itself. A whole file index two levels down will have three IEN subscripts in the same order, and so on.

An excellent example of a whole file index is the New Person file's BB index,, which is actually cross-references the Alias field (.01) of the Alias multiple (field 10, DD 200.04). For comparison I've included the normal B index on the multiple, which cross-references only entries in the multiple, not the whole file:

```
^VA(200,0) = NEW PERSON^200I^42^22
^VA(200,9,0) =
MARSHALL,RICK^TOAD^xxxxxxxxxxxxxxxxxx^@^^^
1^^1
^VA(200,9,3,0) = ^200.04^9^8
^VA(200,9,3,1,0) = TOAD
^VA(200,9,3,2,0) = SHADOW
^VA(200,9,3,"B","SHADOW",2) =
^VA(200,9,3,"B","TOAD",1) =
^VA(200,"BB","SHADOW",9,2) =
^VA(200,"BB","TOAD",9,1) =
```

As you should be able to see from this example, that B index is only useful for picking entries in the Alias subfile if you already know which New Person file entry to look in, whereas the BB index lets us actually pick the New Person file entry based on the values from the Alias multiple.

Index Ownership: Two Files

An important feature of whole file indexes is that they "belong" to two different files. The data comes from the Alias subfile, but the index is located up on the New Person file, and it cross-references both of them; this shared ownership is reflected by the fact that even though the cross-reference definition lives under the .01 field of the Alias subfile's DD entry, it shows up under the list of available lookup indexes in the New Person file's DD. Of the indexes FileMan supports, only whole file indexes have this kind of shared ownership.

Traditional or New Cross-References

As a side note, note that this BB cross-reference on the Alias field is a traditional one that predates version 22 and is therefore defined where cross-references have always been defined: attached to the definition of the field they cross-reference. Starting with version 22, to take advantage of the many new cross-referencing features you may create a "new" cross-reference, which will be defined in the new Index file (# .11) instead.

KWIC, Mnemonic, and Soundex Indexes

Three other types of cross-references, used less frequently and probably therefore less well understood, create indexes but have not been upgraded with version 22 to include the new capabilities. They must be created as traditional cross-references.

The KWIC, or Key Word In Context, cross-reference builds an index that parses out all the words of the field value and stores each one in the KWIC index. This lets users pick file entries based on an incomplete recollection of all the words, and is especially useful for files that store book or movie titles and the like.

Mnemonic cross-references add entries to the file's primary B index based on the values of some field other than the .01. These mnemonic entries are distinguished as being the only index nodes equal to anything other than the empty string, the value 1. The intent is to let other fields behave like extensions of the .01 and be used for most lookups, but given FileMan's many flexible lookup features their use is being discouraged.

Soundex cross-references create indexes that store a code that approximates the sound or structure of the field value rather than the actual field value itself. Similar values should encode to the same code value, which lets a user who can't quite remember how to spell a name but can come close succeed with the selection.

Indexes Created by MUMPS Cross-References

One of FileMan's strongest features is that through the use of programming hooks you can usually extend its capabilities when its built-in features do not solve your problem. One of the most powerful and widely used of these is the M type cross-reference, which fires under the same conditions as a Regular cross-reference but whose logic is anything you choose to have happen at that point.

Although M cross-references can do anything, in practice, well over half of them are used to build indexes not directly supported by FileMan. Often, the resulting indexes conform to the structure of a regular index, so the designers put them among the standard sort or lookup indexes and FileMan detects and uses them like built-in ones, but sometimes the resulting index has a structure not supported by FileMan. The clearest past example of this was compound indexes, which FileMan could be made to use for sorting through the use of a special input parameter to the sort module, but which FileMan could not use at all for lookups.

Now that FileMan 22 has so extended the capabilities of Regular indexes, many existing M cross-references are candidates for replacement. This is particularly true of M cross-references that were used to build Regular compound indexes.

Because FileMan's lookup couldn't use these in the past, they're all up in the sort namespace; if they're redefined in the Index file and renamed to the lookup namespace, FileMan will let users look up file entries with them.

Also, since FileMan had no way of knowing what these cross-references did, it could not optimize the creation of the resulting indexes. With the new support, developers can define Regular compound indexes that are built far more efficiently, firing logic once per record update rather than once per change to a field used to build that index.

Many other kinds of M cross-references are also up for replacement, from those that built inverse date indexes (you can now store the real date and tell FileMan to traverse the index backwards) to those that screened out entries (you can now give FileMan screening code for Regular indexes) to many more.

You should take some time after installing FileMan version 22 to look back over the indexes you've built over the years and decide which ones are worth the investment to transfer over to the new Index file.

Conclusion

Because of the leap in capabilities version 22 represents in cross-referencing, this would be a good time to retrain yourself and your coworkers not just in the new features but also those obscure but useful older ones you may have overlooked. I hope this article has helped give you a framework for exploration. Next column we will walk through the creation of some indexes using FileMan's new ScreenMan options to point out all the choices and pitfalls. **M**

If you have questions or comments about FileMan 22 or topics you would like to see addressed in this column, send email to .FMTEAM@FORUM.VA.GOV,
or write to: Infrastructure Maintenance Team, VACIOFO-San Francisco, Suite 600, 301 Howard Street, San Francisco, CA 94105.

The ubiquitous Rick Marshall (toad@eskimo.com) is a hardhat (www.hardhats.org) who works at the VA's Puget Sound Health Care System. He knows that he is overcommitted, and he even knows why, but the more tasks he removes from his plate the busier his life gets. Weird.

KB_SQL Version 4.0!

The upcoming release of KB_SQL contains several exciting new features, including:

- The proven SQL/ODBC solution for all M types, including Caché
- Windows Query and Reporting Environment
- Support for long TEXT data type
- Improved query optimization for better performance
- Online documentation
- And more!

Want to know more?

www.kbsystems.com

KB Systems, Inc.

Voice (703) 318-0405

KB Systems, Inc.

Time is running out . . .

Every day brings you closer to Y2K.

How long are you going to wait to have your system checked out?

FREE SAMPLE RUN!

24-HOUR TURN AROUND!

M-Tools can help you start your Y2K Compliance Project or make sure you've done it right. Send the routines and you'll recieve a report within 24 hours! It's quick and easy - call today!

- *Checks both routines and global formats*
- *More complete than line parsers*
- *Creates on-line documentation*
- *$3 per routine, quantity discount*
- *FREE sample run (20 routines)*

M-Tools
High Productivity Programming Systems

CALL TODAY! **617.547.1459**
Visit us at: www.m-tools.com