Programming Hooks 105: Intermediate Input Transforms

by Rick Marshall

Programming hook: a significant point at which a programmer can insert M code in the sequence of events that makes up a standard database activity.

Introduction

The previous article in this series introduced this most widely used of FileMan's programming hooks, listed its ten distinct uses, and examined the first. This article will continue that investigation by digging into the techniques you need to successfully use the input transform.

General Issue: Execution

The input transform protects the integrity of the database. When you choose to override FileMan's handling of input transforms, you assume responsibility for preserving that integrity and for ensuring that your field astonishes your users as little as possible.

To meet this responsibility you need to know when FileMan executes input transforms, both to understand what you must reproduce when you go around FileMan's API, and to make educated decisions about when to suppress execution of parts of the input transform.

Creation & Update

a. When creating a new record within a LAYGO lookup, the input transform on the .001 (if one exists) and .01 are executed. However, when you create entries with FILE^DICN, FileMan does not fire the .01's input transform. Also, input transforms on screened pointers and screened sets of codes aren't fired during record creation. FileMan treats any fields updated during record creation, such as required identifiers and fields specified in the DIC("DR") input variable, as updating (see below) rather than creating.

b. When updating field values, whether entering them for the first time or editing them thereafter, FileMan executes the input transform for each field changed. You can override this when updating fields through FileMan's API, or when using an input template, as documented in the FileMan manuals. Also, the input transform does not execute when you delete field values.

c. When FileMan installs filegrams on your system, it executes the input transforms for the affected fields.

d. When FileMan imports data from a foreign environment, it will fire the input transform if you specify the data to be in external format.

e. When FileMan extracts data from one FileMan file into another, it executes the input transform for the changing fields of the destination file.

Validation & Record-Independent Firing

f. The Verify Fields utility option fires the input transform for the chosen field on every record in the file.

g. FileMan's Reader module can prompt the user based on a field's definition. Such DD reads involve firing the base field's input transform on whatever value the user enters.

h. The Data Checker call in FileMan's Database Server (DBS), CHK^DIE, fires the chosen field's input transform to test a value not yet associated with a specific record.

i. The Data Validator, on the other hand, executes the input transform for a value proposed as a new value for a field in a specific record.

Surprises

j. DIFROM, the DIFROM Server, and KIDS do not execute input transforms. Since the data (transported in internal format) already passed its input transforms when FileMan filed it in the source database, FileMan does not fire the input transforms again at the target site.

k. The Transfer/Merge option does not execute the input transforms of the fields in the transferred or merged records, since this option assumes that the target and destination files store data in a similar format.

l. Triggers do not fire input transforms.

m. The input transform of a pointer or set of codes field is sometimes par-

tially fired if a cross-reference on that field is used in a lookup. The conditions and results of this are tricky enough that we will treat it as a separate subject in a future article on input transforms.

n. Bypassing FileMan and setting data directly into the globals means taking the entire responsibility for data integrity upon yourself, not surprisingly.

General Issue: On Which Record Are You Operating?

Many of the input transform's functions beyond syntactical validation require finding out the identity of the executing transform's record. For example, to coordinate the current field's value with another field (no pregnant patients who are male, say), an input transform needs to identify the record for which it is firing so it can fetch the other field belonging to the same record.

Unlike certain other programming hooks, such as the whole file screen, the input transform does not offer you a naked indicator preset to the level of the current record. Instead, you must refer to the current record explicitly by its internal entry number (IEN).

A Review of IENs

You can uniquely identify a record in a relational file, or at the top level of a hierarchical file, by knowing only two numbers: the file number and the record number within that file. However, in the lower levels of hierarchical files (called multiples or subfiles) the tree structure of the hierarchy requires more information for identification.

In tree theory terms, you cannot identify a record without identifying its ancestors all the way back to the file root. Individual record numbers can only distinguish among records that share the same immediate parent. Since records at the top level of a tree all share the same root as their common parent, knowing their record numbers is enough to uniquely identify them.

Below that first level, while knowing the record number does tell you which record you want of those in its immediate group of siblings, it doesn't tell you which group of siblings you're referring to. In a tree of ten records, each of which has five descendent records, knowing the subfile record is number 1 only cuts your choices down from fifty to ten; all ten top level records have a descendent record number 1. Just as in works of heroic fantasy, you must name each record's parents going back to the original ancestor; our record is the number 1 whose parent is number 5. So throughout FileMan, identifying a record takes the file number and a number of record numbers equal to the level of the file within its hierarchy: one for the top level, three for a multiple two levels below it.

The DA Array

For input transforms, FileMan puts the current record's IENs into the local array DA. The IEN of the current record is in DA itself, its parent's record number is in DA(1), grandparent's in DA(2), and so on. When you run out of ancestors, counting back toward the file root, the DA array runs out of array nodes; top level records have only DA.

Or is it that simple?

DA & .01 Fields

Consider the input transform on a .01 field for a new record. Let's imagine the new .01 value is "SIBELIUS,JEAN" (for a COMPOSER file). The user gives FileMan "SIBELIUS,JEAN" as a lookup value for the file, and FileMan confirms that no such record yet exists. It asks the user to confirm that this should be a new record and then passes the value through the input transform of the .01.

At this point, the new record does not yet exist, so it can't have an IEN or a DA value. For that matter, it doesn't have any other field's values yet, so there's nothing for the input transform to compare the .01 with in the current record. The same input transform that handles modification of the .01 in an existing record must be able to deal with this situation as well, so clearly the programmer needs some way to distinguish the two cases.

For historical reasons, the DA array during a new record addition can't be distinguished from the DA array during updating. There are no value or structural differences the programmer can use to make the input transform recognize from DA whether the .01 is being added or updated. DA may or may not be defined and if defined, may have a random value; additionally, extraneous DA array elements may be present. However, except for the top level DA variable, any others that should be present (DA(1) should be present when working with an entry in a subfile under a top level file) will be present and have the correct values.

DA & Other Contexts

When the input transform fires in a record-independent context, such as part of a Reader's DD read or for the Data Checker call, the DA array has no meaning. Any input transforms that normally try to coordinate the field's value with other fields must not do so in these contexts. Knowing that the input transforms may not be tied to a record

should help programmers write them to behave correctly in either case.

Another context that needs the developer's increasing attention is transaction processing. For example, ScreenMan sessions operate as transactions, with the changes proposed by the user saved up until the user saves the data. Many developers assume FileMan always files data sequentially and writes input transforms that assume the presence of other fields, but clearly, with the advent of GUI, such sequential processing will increasingly become the exception. Knowing the potential for transactions should lead developers to code their inter-field relationships flexibly, to run correctly however many fields may happen to have been filed at any given time.

FileMan also frequently parses the input transform instead of executing it. The overloading of the input transform discussed in the previous article means FileMan can often save time executing it by making a decision based on quick scans of its contents. For example, FileMan may check for +X=X in the input transform and take this to mean the field is numeric, or may look for %DT= in a date field to figure out what kind of help to provide a confused user. Knowing that FileMan parses input transforms should lead developers to be conservative in transforming existing input transforms, to retain the kinds of coding constructs FileMan looks for.

As we'll see later, FileMan takes this as far as executing selected pieces of input transforms under certain circumstances, such as when looking up entries with a pointer index. Some of these kinds of activities are private, variable tactics that developers should not rely on, but others are supported and lend themselves to valuable solutions that are not otherwise easy to arrive at. Knowing where FileMan puts the boundaries on such partial execution of input transforms will help programmers choose the sequence of activities within their transforms.

Conclusion

Identifying the record under consideration and staying aware of the context in which input transforms execute are essential to making them do the right thing in all the situations in which they might be called to action. In the next issue we'll get into how to apply this knowledge to the various problems you can solve with input transforms. **M**

Forward your FileMan questions to the mail group FMTEAM on the VA's FORUM system, or write to: VA IRMFO San Francisco, Suite 600, 301 Howard Street, San Francisco, CA 94105.

Rick Marshall works at the Seattle Development Satellite office of VA's San Francisco IRM Field Office. He works on FileMan, the MTA Board of Directors, and the MDC, and is currently writing the 1995 Standard M Programmers' Reference Manual.
