Programming Hooks 104: Introduction to the Input Transform

by Rick Marshall

Introduction

This is the first in a series of articles that examines programming hooks available in VA FileMan. Most standard database activities within FileMan contain programming hooks, points at which a programmer can insert M code to change the outcome. Knowing their proper use distinguishes the FileMan expert from the novice.

Ironically, the best known and most frequently used programming hook, the input transform, is also one of the hardest to master. The name suggests a single function for a programming hook that has been overloaded to accommodate at least ten distinct, related capabilities including:

1. Validation of the syntax of a data type.
2. Screening out of certain syntactically valid choices.
3. Validation of relationship to other field values.
4. Pointer lookup control.
5. Transformation from external syntax to internal syntax.
6. Information about the length of the field.
7. Record numbering (01 field only).
8. Warnings to the user.
9. Side effects--M code entered by the developer, code unrelated to input transform per se, but that the developer wants to execute at this point in processing.
10. Definition of computed field.

Changing the Input Transform

The basic features of the input transform are constant across its functions. FileMan stores each field's input transform in ^-pieces 5 and up in the 0-node of the field definition in the ^DD global. FileMan automatically creates and adjusts each field's input transform as you define the data type and other characteristics of the field. However, FileMan lets you use and manipulate the input transform yourself as well, using the Input Transform (Syntax) option on the Utility Functions menu.

If you change the input transform, FileMan disavows all knowledge of the field's data type and will no longer let you modify the data type with the standard Modify File Attributes option. As documented in the Programmer Manual, this is controlled by an X flag that FileMan puts in ^-piece 2 of the field definition 0-node. This flag is essential. Without it, FileMan would incorrectly treat your handcrafted field definition as the base data type you originally assigned it. Therefore, you should not tamper with it casually.

However, if during your initial development of the field you change the input transform, but then later change your mind and return the input transform to its original definition, you should remove the X flag manually. In return, FileMan will once again let you use Modify File Attributes to manipulate the field definition.

General Usage Issues

The input transform's interface is simple. FileMan sends in X as a potential field value, and the input transform sends out X in one of two ways: undefined, to signify the input value was bad, or defined. For most data types, FileMan sends in X in external format (such as "JUL 7, 1996"), but for variable pointers X will come in internal format (such as "42;DIC(19,"). In either case, if the input transform accepts X it should send it back out to FileMan in internal format (such as 2970708, for our date example, or unchanged for the variable pointer). While there are a

few commonly used undocumented input and output variables, X is the only documented, reliable one. The others, which we will discuss in future columns, have side effects or are difficult to use.

That's it. Unlike similar programming hooks, the input transform can not rely upon the naked indicator being set to any particular location beforehand and is not expected to adjust $TEST.

Unlike output transforms, input transforms do not need to worry about preserving the flow of control through the hook. This is a common misunderstanding based on how closely the two hooks are related. You are free to use the IF, ELSE, FOR, NEW, and other commands that alter the flow of control. In fact, consider yourselves particularly encouraged to use the NEW command to clean up any temporary variables you create within the input transform; this will improve your programming hook's reentrancy if it is called in recursive situations, even aside from the housekeeping advantages for your symbol table.

Aside from these guidelines, the usual rules for programming hooks apply. Think carefully before introducing any I/O into your input transform. Where possible, move it into your user interface instead. When you must give the user output, use the EN^DDIOL utility to ensure it will work not just with a scrolling-mode user interface, but with screen-oriented or GUI interfaces as well. The details to using the input transform beyond these kinds of general guidelines are specific to the problems you are trying to solve.

Data Type Syntax

The input transform's original function was to validate the syntax of a field's data type. When you assign a data type to a field, FileMan builds an input transform to validate its syntax. FileMan's base data types each use the input transform in a slightly different way:

1. DATE/TIME: builds a call to the %DT data validation tool.

2. NUMERIC: builds a postconditional KILL command that ensures that the value is numeric, falls within the right range, and has the right number of fractional digits.

3. SET OF CODES: doesn't use the input transform; just inserts a QUIT command. Instead, the set of codes is defined in ^-piece 3 of the field definition's 0-node.

4. FREE TEXT: builds a postconditional KILL command that enforces the length of the value and may include an optional pattern match (see below).

5. WORD-PROCESSING: the stub field definition has no input transform, the .01 of the word processing subfile has only a QUIT command.

6. COMPUTED: contains the M code to implement the computed expression. We'll cover input transforms on computed fields in detail in a future article.

7. POINTER TO A FILE: builds a call to the DIC record selection tool that includes an optional screen on the choices. This too, is fodder for a future article.

8. VARIABLE POINTER: doesn't use the input transform, just inserts a QUIT command.

9. M: builds a call to the DIM code validation tool.

For your purposes though, the great value of the input transform is in creating new data types out of existing ones. For now, new data types must be redefined for each new field you want to use them with, but in the near future you will be able to reuse your definitions. The input transform lets you define the syntax of a new data type by accepting only correctly formatted strings. You should probably follow up this syntactic definition by adjusting the field's description and help, and adding functions to the Function file.

When planning a new data type for your field, be sure to start from the base data type most similar to it. Changing the input transform will be enough to alert FileMan to be careful about assuming how your field behaves, but it will still expect it to follow the general rules of collation length, etc. For example, if a social security number data type is stored without its hyphens (e.g., 123456789), it can be based on the numeric data type, but with them intact (e.g, "123-45-6789") it had better be based on free text. Think about the impact of those hyphens on its collation sequence, let alone arithmetic operations.

When in doubt, base data types on free text, always a safe bet. FileMan makes few assumptions about the behavior of free text, which has the added advantage of giving you access to pattern match capabilities without the need to modify the input transform directly.

All of this assumes your new data type is based solely on standalone syntactic considerations. Definitions based on relationships with other parts of the database or other forms of decision making are more involved and will be covered later. M

(Next issue, *Intermediate Input Transforms*)

Forward your FileMan questions or topics you would like to see addressed in this column to the mail group FMTEAM on the VA's FORUM System, or write to: VAISC6 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 is a member of the FileMan development team, the MTA Board of Directors, and is currently writing the 1995 Standard M Programmers' Reference Manual and editing the next draft M Language Standard.

MTA NOTEBOOK

M Technology Association
1738 Elton Rd.
Suite 205
Silver Spring, MD 20903

MTA Annual Conference Set for '97

The 1997 MTA Annual Conference will be held in Boston at the Hynes Convention Center the week of May 18th. MTA will once again join forces with Database & Client/Server World.

Conference Program Committee Volunteers Needed

MTA is looking for volunteers to participate in the planning of the 1997 annual conference. Anyone interested should contact MTA at 301-431-4070 or send e-mail to:

MTA1994@aol.com

Netscape: A New Avenue for M?

Netscape Navigator™ beta version 2.02 has a new scripting language called "LiveScript." It is used to control the way the browser displays pages on websites.

Rumor has it that M is far better suited to this purpose. Is there anyone out there who would care to comment? For more information on LiveScript, see: http://www.netscape.com