

Fileman 22.2

Author: Sam Habel, VISTA Expertise Network

Update: Frederick D. S. Marshall, VISTA Expertise Network

Introduction

This paper presents a new version of Fileman, MSC Fileman 22.2.

Fileman is a database management system written in Mumps. Astute readers will note that Mumps already provides a storage capabilities consisting of saving data as sparse multi-dimensional arrays. Fileman builds on top of Mumps a full database management system with capabilities including:

- Abstracting Mumps multi-dimensional arrays into user-friendly files
- Providing user-defined data definitions for all data elements
- Providing indexing for data stored in the Mumps arrays for easy retrieval
- Providing interactive data behavior (using triggers, mail messages, or Mumps style cross-references)
- Providing the User Interface for manipulation and reporting of the data
- Providing tools for sending data to other Fileman systems.
- Providing programmer tools for use to manipulate Fileman stored data.
- Providing a security framework for data access.
- Providing a Structured Query Language (SQL) interface to the data.

Fileman is indefinitely extensible. Virtually every aspect of it can be customized by writing your own code and calling it from a Fileman hook. For example, you can write your own search logic for a file and have Fileman execute that rather than the default search logic.

Fileman is a hybrid relational/hierarchical database management system. It can store data in a relational manner (like a traditional SQL based database) or in a hierarchical manner (like an XML database).

Fileman can be installed by itself without the VISTA Kernel, but doing so deprives you of features such as Operating System file level interaction, sending of mail messages, and use of the Remote Procedure broker to access Fileman data from other programming languages via the Transmission Control Protocol (TCP) over Internet Protocol (IP) version 4. It's expected that users who use Standalone Fileman will replace these functions on their own.

Installation Instructions

The software is supplied as a set of routines packaged in Routine Output (RSA) format.

The latest version can be downloaded from OSEHRA's public repository.

<https://github.com/OSEHRA/fileman-22p2>

As of the time of this writing, the RSA file can be obtained by getting the response of this URL:

<https://github.com/OSEHRA/fileman-22p2/blob/master/VA-FILEMAN-22P2T1.RSA?raw=true>

Detailed installation instructions are supplied in a manual accompanying this journal article.

Configuration

Initial configuration of Fileman is done from Mumps by running the routine DINIT. The major configuration item there is choosing your Mumps Operating System (i.e., your Mumps Virtual Machine). You need to answer this correctly in order for Fileman to function properly. The currently supported Operating Systems are:

- CACHE/OpenM
- DSM for OpenVMS
- DTM-PC
- GT.M(UNIX)
- GT.M(VAX)
- MSM
- OTHER

As of this version, only Cache/OpenM and GT.M(Unix) are fully supported and tested. The others are there because they still by and large work, but they are unsupported by the Fileman development team.

More details on running DINIT can be found in the Installation Guide.

Other configuration items are system specific and can be set using Mumps programmer mode:

- `^DD("STRING_LIMIT")` sets the maximum length of a global node. By default, it's 255, following the latest Mumps Standard (MUMPS 95). In order to use Fileman with lengthy vocabularies such as SNOMED or RxNorm, it may be necessary to change the length of the maximum string limit to fit the longest field supplied by the database.
- `^DD("DILOCKTM")` sets the standard lock time. This may be needed in clustered Cache environments where lock times include the time for the connection to the remote machine to be made.
- `^DD("DD")` sets the date/time format to be used by Fileman. This could be changed to produce a different locale appropriate date/time output.

Documentation

Most of the documentation is in the process of being written. The planned documentation will consist of:

- Release Notes
- Value Proposition
- Install Guide

- Getting Started Manual
- Advanced User Manual
- User Reference Manual
- Programmer Reference Manual
- Security and Privacy Manual
- Technical Manual
- Key and Index Tutorial
- Screenman Tutorial

Notable new features in Fileman 22.2

This section includes the new features. Bug fixes are not mentioned. They are documented in the Release Notes.

Screenman enhancements

Screenman is a screen oriented editor which mirrors a Graphical User Interface but still runs on a terminal emulator.

In version 22.2, Screenman has had several major enhancements. We added the capability of the Screenman to have customizable colors for each type of presentable element. In addition, Screenman can now understand DEC/XTERM X11 mouse reporting so that users can use a mouse to point and click. Support for this is not universal among terminal emulators; consult your terminal emulator's documentation to see if this feature is supported. Our experience has been that Putty, gnome-terminal, konsole support it with no or little configuration. On Attachmate Reflection, you need to set your terminal emulation type to "Linux Console" for this to work.

Two more minor changes: Screenman now includes a "Previous Page" command, which was omitted in the original version; and Screenman now shows the presence of text in a Word Processing field by a "+" sign.

Below are two screen shots comparing the before (v. 22.0) and after (v. 22.2) for Screenman.

Name: **PS0*6.0*102**

TYPE: **SINGLE PACKAGE**

Name: **PS0*6.0*102**

Date Distributed: **MAY 29,1996**

Description:

Delete Routine
after install

Environment Check Routine:

Y/N:

Pre-Install Routine: **PSOCLPOS**

Y/N:

Post-Install Routine:

Y/N:

Pre-Transportation Routine:

Exit Save Next Page Refresh

Enter a command or '^' followed by a caption to jump to a specific field.

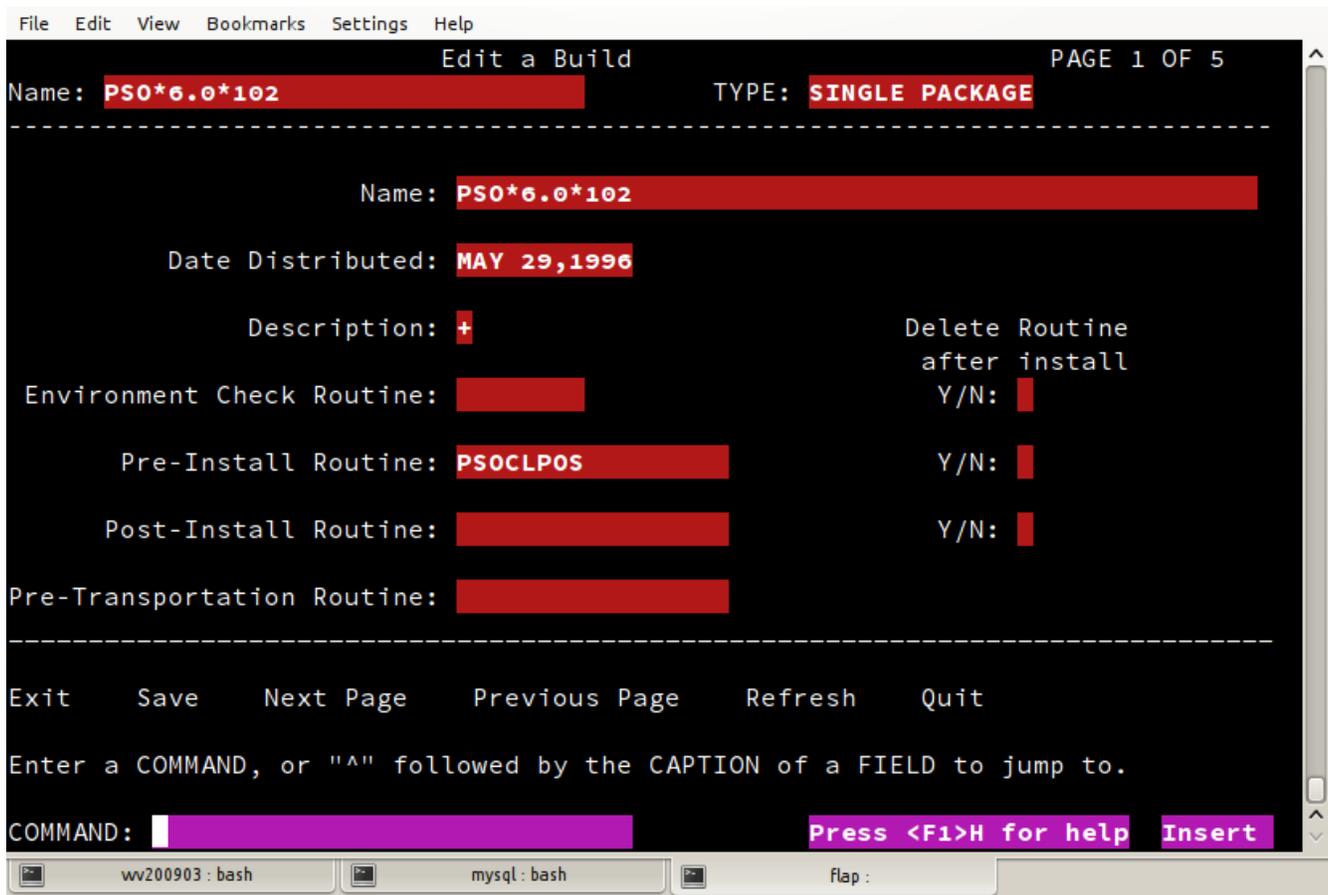
COMMAND:

Press <PF1>H for help

ww200903 : bash

mysql : bash

flap :



Internationalization and Localization

Fileman is more capable than ever in the arena of Internationalization (i18n) and Localization (l10n). The following new features are available:

- Fileman can now have data dictionary elements in any language and will display the correct language depending on the user's DUZ("LANG") value.
- Fileman handles dates consistently based on the value of the ^DD("DD") node.
- Fileman seamlessly handles all forms of non-US Dates using the "I" flag to %DT.
- Fileman contains an optional expanded language file that contains all the ISO-632 languages .
- Fileman uppercasing code is language specific.

Interested users can find how to translate data dictionary elements by running the routine DIALOGZ from the top after setting their DUZ("LANG") to the Language Internal Entry Number (IEN). Convenience entry points also exist for several language, including GERMAN^DIALOGZ and SPANISH^DIALOGZ.

```
dev>D SPANISH^DIALOGZ
Modify what File: VA FILEMAN CHANGE// 2 PATIENT (11 entries)
SPANISH translation of PATIENT:
```

The value of ^DD("DD") without any user change is by default S Y=\$\$FMTE^DILIBF(Y,"5U"). This entry point checks if a language (File .85) has entry for field 10.21, DATE/TIME FORMAT (FMTE). If the language isn't defined in DUZ("LANG") or is English (DUZ("LANG")=1), the default logic is executed.

The expanded language file can be installed by running ^DMLAINIT. This is already noted in the Install Manual.

Users are referred to the Programmer Manual for %DT documentation for instructions on how to use it with the "I" flag.

Uppercasing code for Fileman is located at \$\$SUP^DILIBF.

Improved Data Analysis Tools

Fileman has a better Data Analysis and Data Integrity Check tools. These include.

- Dangling Pointer reporter, currently not invocable from inside Fileman but invocable using ^DIVRPTR.
- Find Pointers Into a File
- Improved Verify Fields Option
- A Data Dictionary comparer for comparison of Data Dictionaries across different Mumps Instances.

Dangling Pointer Reporter

Here's an example of running this:

```
v22p2>D ^DIVRPTR

START WITH What File: BROKEN FILE//      (9 entries)
      GO TO What File: BROKEN FILE//      (9 entries)
DEVICE: HOME// ;;99
DANGLING POINTER REPORT

FILE 1009.801 'POINTER' (Field #.02 in File #1009.801)
BROKEN FILE: `2 BAD POINTER >>No '6666' in LANGUAGE File<<
```

Find Pointers Into a File

This is a new menu option under the DATA DICTIONARY UTILITIES menu.

Select OPTION: DATA DICTIONARY UTILITIES

Select DATA DICTIONARY UTILITY OPTION: ?

Answer with DATA DICTIONARY UTILITY OPTION NUMBER, or NAME

Choose from:

- 1 LIST FILE ATTRIBUTES
- 2 MAP POINTER RELATIONS
- 3 CHECK/FIX DD STRUCTURE
- 4 FIND POINTERS INTO A FILE

Select DATA DICTIONARY UTILITY OPTION: 4 FIND POINTERS INTO A FILE

THIS UTILITY TRIES TO FIND ALL ENTRIES IN ALL FILES POINTING TO A CERTAIN FILE

Select FILE: LANGUAGE

Select one of the following:

- 1 One particular LANGUAGE Entry
- 2 All LANGUAGE Entries
- 3 Non-existent LANGUAGE Entries

Find pointers to: All LANGUAGE Entries//

Verify Fields

There are many subtle improvements to the Verify Fields option. There include:

- Better cross reference checking (including checking whether a 30 character limit for a cross-reference was obeyed)
- Better handling of Date value errors
- Better verification of fields that contain an Output Transform. If a field's contents fail its Input Transform, the contents are sent through the Output Transform and checked again against the Input Transform.

An example report follows below.

VERIFY FIELDS REPORT

KBAN BROKEN FILE FILE (#11310003)

JAN 09, 2013 14:33 PAGE 1

--11310003,.01--FIELD #.01 NAME-- (FREE TEXT)

(CHECKING CROSS-REFERENCE)

ENTRY#	NAME	ERROR
6	BAD B XREF	"BAD B XREF" not properly Cross-reference
10	THIRTY-TWO CHARACTER LIMIT ENTWRONG "B" CROSS-REF 'BAD B XREF'	
99	99	DANGLING "B" CROSS-REF 'BAD B XREF'
9	LOOPY OUTPUT TRANSFORM	WRONG "B" CROSS-REF 'LOOPY OUTPUT TRANS'
10	THIRTY-TWO CHARACTER LIMIT ENTDUPLICATE "B" CROSS-REF 'THIRTY-TWO CHAR	

--11310003,.02--FIELD #.02 POINTER-- (POINTER)

(CHECKING CROSS-REFERENCE)

2	BAD POINTER	No '6666' in pointed-to File
7	BAD C (POINTER) XREF	"4" not properly Cross-referenced
7	BAD C (POINTER) XREF	WRONG "C" CROSS-REF '44'
2	BAD POINTER	WRONG "C" CROSS-REF '7777'

--11310003,.03--FIELD #.03 DATE-- (DATE)

(CHECKING CROSS-REFERENCE)

3	BAD DATE	"ABCDEF" fails Input Transform
8	BAD D (DATE) XREF	"3220801": D index (#1051) not properly set

--11310003,.04--FIELD #.04 SET-- (SET OF CODES)

4	BAD SET	"U" not in Set
---	---------	----------------

--11310003,.05--FIELD #.05 NUMBER-- (NUMERIC)

```
5          BAD NUMBER          "ABCDEF" fails Input Transform
```

```
--11310003,.06--FIELD #.06 LOOPY OUTPUT TRANSFORM-- (DATE)
```

Data Dictionary Comparer

This allows you to compare different instances of VISTA for Data or Data Dictionary changes. This is a new option under the TRANSFER ENTRIES menu.

Note: Under GT.M, you need to make sure that your Global Directory file only contains absolute path references for this to work. Using relative paths and environment variables in your Global Directory file will cause inaccurate behavior.

Note2: In case you are wondering, no, you can't compare a GT.M database against a Cache database.

```
GTM>S DUZ=1 D P^DI
```

```
VA FILEMAN 22.2T1
```

```
Select OPTION: TRANSFER ENTRIES
```

```
Select TRANSFER OPTION: ?
```

```
    Answer with TRANSFER OPTION NUMBER, or NAME
```

```
    Choose from:
```

- 1 TRANSFER FILE ENTRIES
- 2 COMPARE/MERGE FILE ENTRIES
- 3 NAMESPACE COMPARE

```
Select TRANSFER OPTION: 3  NAMESPACE COMPARE
```

```
UCI: /home/sam/emptyENV3/g/mumps.gld
```

```
START WITH What File: RXNCONSO//          (202420 entries)
```

```
GO TO What File: RXNCONSO//          (202420 entries)
```

```
Compare to what UCI: // /home/sam/emptyENV2/g/mumps.gld
```

Select one of the following:

- 1 DATA DICTIONARY ONLY
- 2 FILE ENTRIES ONLY
- 3 DATA DICTIONARY AND FILE ENTRIES

Enter response: 3// 1 DATA DICTIONARY ONLY
DISPLAY COMPARISON ON

JAN 15, 2013 SAM'S RXNORM SITE

UCI: /home/sam/emptyENV3/g/mumps.gld UCI: /home/sam/emptyENV2/g/mumps.gld

DATA DICTIONARY #1009.811 (RXNCONSO)
{Missing}

Enhanced Auditing Capabilities

This new version has the following enhanced auditing capabilities:

- Fileman now allows Word Processing fields to be audited
- Data Dictionary changes are automatically audited
- There is a new menu option to show changes done by a specific user on a specific file

To audit word processing fields, use the option OTHER OPTIONS > AUDITING > TURN DATA AUDIT ON/OFF. You will be asked for the field, and you can choose a word processing field. After modifying an audited field, you can check for the audit trail using the captioned output of the Inquire option.

Select AUDIT OPTION: 5 TURN DATA AUDIT ON/OFF
Audit from what File: TIU DOCUMENT// (3 entries)
Select FIELD: REPORT TEXT (word-processing)
Select REPORT TEXT SUB-FIELD: .01 REPORT TEXT
AUDIT: YES, ALWAYS

To see Data Dictionary changes, use the option OTHER OPTIONS > AUDITING > SHOW PAST CHANGES TO DD'S. You will be asked to select a date range.

Select AUDIT OPTION: SHOW PAST CHANGES TO DD'S

```
Show Data Dictionary changes since: First// T-1 (JAN 14, 2013)
```

```
DEVICE: HOME//
```

To see changes done by a specific user, use the option OTHER OPTIONS > AUDITING > MONITOR A USER.

```
Select AUDIT OPTION: 2 MONITOR A USER
```

```
Select a USER who has signed on to this system: `1 MASTER,USER
```

```
Select AUDITED File: PACKAGE//
```

```
START WITH DATE: FIRST//
```

```
DEVICE: TELNET
```

```
PACKAGE RECORDS ACCESSED BY MASTER,USER (DUZ=1)
```

```
Page 1
```

```
EARLIEST ACCESS
```

```
LATEST ACCESS
```

```
-----  
VA FILEMAN
```

```
JAN 15,2013@13:27:57
```

Enhanced Database Server Finder and Lister calls

Both the database server (DBS) finder (FIND^DIC) and lister (LIST^DIC) calls have been enhanced to provide capabilities available in the Fileman print module. These enhancements are of very significant value to VISTA programmers who had to settle for less when using these modules. These enhancements are:

- The return fields list (3rd argument for both the finder and lister) can now return computed fields, including multiples and forward and backward relational jumps.
- The “E” flag (4th argument for both the finder and lister) will now cause all output to be returned even if the data is invalid (especially with sets of codes and pointers fields).
- The new eighth parameter of the lister will be invoked if there is an “X” flag present (4th argument). This parameter will invoke Fileman's print module sorter to allow you to sort your data in any order the print module allows. For example:
 - You can now sort by an un-indexed field.
 - You can construct a filtering expression to just return certain records.
 - You can sort by a previously defined SORT TEMPLATE
 - You can sort by several fields (e.g. by Date of Birth, then by Name)
 - You can sort by multiples rather than top-level fields.
- To help with the “X” flag, you can now create a SORT TEMPLATE silently using the call BUILDNEW^DIBTED.

Several examples will be presented below. When using ZWRITE, apply your Mumps implementation

preferred format to get the desired output. Alternately, you can use \$QUERY to inspect the returned data. The ZWRITE format presented below is GT.M's.

Returning Computed fields as part of the third argument.

Here's an example using the finder. This example returns the name and abbreviation plus the count of the counties for each state that begins with the letters "AL". You will see that there are three entries that match.

```
v22p2>D FIND^DIC(5,,"@;.01;1;COUNT(COUNTY)","", "AL")
```

```
v22p2>ZWRITE ^TMP("DILIST", $J,*)
```

```
^TMP("DILIST", 16294, 0) = "3^*^0^"
```

```
^TMP("DILIST", 16294, 0, "MAP") = ".01^1^C4"
```

```
^TMP("DILIST", 16294, 2, 1) = 1
```

```
^TMP("DILIST", 16294, 2, 2) = 2
```

```
^TMP("DILIST", 16294, 2, 3) = 100
```

```
^TMP("DILIST", 16294, "ID", 1, .01) = "ALABAMA"
```

```
^TMP("DILIST", 16294, "ID", 1, 1) = "AL"
```

```
^TMP("DILIST", 16294, "ID", 1, "C4", 1) = 67
```

```
^TMP("DILIST", 16294, "ID", 2, .01) = "ALASKA"
```

```
^TMP("DILIST", 16294, "ID", 2, 1) = "AK"
```

```
^TMP("DILIST", 16294, "ID", 2, "C4", 1) = 29
```

```
^TMP("DILIST", 16294, "ID", 3, .01) = "ALBERTA"
```

```
^TMP("DILIST", 16294, "ID", 3, 1) = "AB"
```

```
^TMP("DILIST", 16294, "ID", 3, "C4", 1) = 1
```

The following example gets you all providers associated with a Visit by using a backwards pointer from the VISIT file to the V PROVIDER file via the V PROVIDER's "AD" cross-reference. We asked the lister here to pack the output and only return 10 results. Notice the 3rd entry for the way multiple results from the backward pointer are packed.

```
GTM>D LIST^DIC(9000010,,"@;.01;V PROVIDER:PROVIDER", "P", 10)
```

```
GTM>ZWRITE ^TMP("DILIST",*)
```

```
^TMP("DILIST", 10057, 0) = "10^10^1^"
```

```
^TMP("DILIST", 10057, 0, "MAP") = "IEN^.01^C3"
```

```
^TMP("DILIST", 10057, 1, 0) = "51^FEB 6, 2005^"
```

```
^TMP("DILIST", 10057, 2, 0) = "1^JUL 1, 2005@14:37:23^DOCTOR, THIRTEEN"
```

```
^TMP("DILIST", 10057, 3, 0) = "2^JUL 1, 2005@14:37:23^DOCTOR, TEST~DOCTOR, THIRTEEN"
```

```

^TMP("DILIST",10057,4,0)="144^JUL 1,2005@14:37:23^"
^TMP("DILIST",10057,5,0)="9^JUL 8,2005@13:03^DOCTOR,EIGHT"
^TMP("DILIST",10057,6,0)="14^JUL 15,2005@16:00^DOCTOR,TEST"
^TMP("DILIST",10057,7,0)="3^JUL 18,2005@17:57:49^COORDINATOR,SIX"
^TMP("DILIST",10057,8,0)="4^JUL 19,2005@07:41:49^COORDINATOR,SIX"
^TMP("DILIST",10057,9,0)="5^JUL 19,2005@07:45:55^COORDINATOR,SIX"
^TMP("DILIST",10057,10,0)="6^JUL 19,2005@08:52:13^COORDINATOR,SIX"

```

“E” Flag

Most of the time, using the “E” flag will produce no difference than not using it. However, if you have invalid data in Fileman (as in an invalid code in a set of code field), the “E” flag will cause the lister or finder to continue returning all data, in spite of the error. Previously, and without the “E” flag, the lister/finder will return a partial listing and therefore misleading the user on the actual entries.

To test this, set a Patient's Sex in file PATIENT (#2) to be “U” by forcing the value in. Use the lister to grab the patients, and you will notice the number of patients returned is different with the “E” flag versus without the “E”.

Print Module Integration with the Lister

The new eighth parameter of the lister will be invoked if there is an “X” flag present (4th argument). This parameter will invoke Fileman's print module sorter to allow you to sort your data in any order the print module allows.

A few brief examples (with no output presented) will show how this works:

Example 1

Caller no longer needs to know whether the field he wants to sort by has a cross-reference on it. If it has an index, that index will be used. Otherwise the sort will be done on-the-fly. Here is how to sort users (File 200) by LAST OPTION ACCESSED (Field 202.1):

```
D LIST^DIC(200,,.01,"X",,,,202.1)
```

Example 2

Boolean-valued Computed Expressions are very useful for FileMan sorts. They can simulate the functionality of FileMan's "SEARCH" Option. Here is how to retrieve only DRUGs (File 50) with a MESSAGE (Field 101) containing "HOME MED", and a GENERIC NAME field containing "VITAMIN":

```
D LIST^DIC(50,,".01;101","XP",,,,MESSAGE[""HOME MED""&(#.01[""VITAMIN""])"
```

Examples 3 & 4

"INDEX" can also be the bracketed name of a SORT TEMPLATE. The Template can be of either type, sort or search. The SORT algorithm, however, must not specify "asking the user" for "From" or "To" fields. Here is how to retrieve the first nine LAB TEST values (File 60), sorted by the "LR TEST DICTIONARY" Template:

```
D LIST^DIC(60,,.01,"XP",9,,,"[LR TEST DICTIONARY]")
```

Next, a sort module is created only the fly to return States whose name begin with "NEW" sorted in the order of most number of counties.

```
N DMUST ; Holds Sort Template Text
S DMUST(1)="SORT BY: -COUNT(COUNTY)" ; Sort by the reverse of number of counties
S DMUST(2)="From:"
S DMUST(3)="To:"
S DMUST(4)=" WITHIN COUNT(COUNTY), SORT BY: $E(NAME,1,3)="NEW"" ; Only get the
States whose names start with NEW
N RET ; RP style return reference variable
D BUILDNEW^DIBTED(.RET,5,$NA(DMUST),"DMU NEW STATES W MOST COUNTIES")
D LIST^DIC(5,,".01;COUNT(COUNTY)","X",,,,"[DMU NEW STATES W MOST COUNTIES]")
```

Example 5

The eighth parameter is in fact evaluated as the "BY" parameter of a call to EN1^DIP. Thus, it can contain a string of "answers", separated by commas. Here is how to sort patients (File 2) by year of birth, and within that alphabetically:

```
D LIST^DIC(2,,.01,"XP",,,,"YEAR(DOB),NAME")
```

Example 6

The "BY" answers can navigate downward into multiples. Here is how to sort OPTIONS (File 19) by the Menu Options they call:

```
D LIST^DIC(19,,.01,"XP",,,,"10,.01")
```

Silent Creation of Sort Templates

This is done using BUILDNEW^DIBTED. See example 4 of how that is used. Programmers should note that the sort template creator is very exact on what it will accept as valid input. Programmers should create the sort template by hand and then copy the generated array which can be obtained using the EDIT TEMPLATE option under the UTILITIES menu. This array needs to include all the spaces as specified in the original sort template.

Definition of Canonic Input, Sort or Print Templates

Fileman's Template Edit option has been upgraded to allow a template to be defined as "canonic" for a file, making it the file's default template. Three canonic templates can be defined for each file, one for each of the three main kinds of templates. Canonic templates improve the usability of files and Fileman options by remembering a reasonable selection of fields and formatting for all three operations.

Applying a Field's Output Transform to Totals and Subtotals

When totals and subtotals are generated from fields with defined output transforms, those same transformations are now applied to the totals and subtotals. For example, if a numeric field's output transform inserts commas to make a long number more readable, those commas will now be inserted into totals and subtotals derived from that field.

Computed From and To Values

When a user enters a computation as the From or To value of a sort field, Fileman now offers the option of whether to run the calculation now and store the result as a fixed From/To value, or to save the calculation and rerun it whenever the sort is performed in the future, allowing for more flexible sort specifications.

Data Protection Enhancements

To better protect data from unintentional changes, the following changes have been introduced:

- Word Processing fields can now be made uneditable.
- Set Cross-references to be non-rerunnable

To make a word processing field uneditable, simply go to UTILITY FUNCTIONS > UNEDITABLE DATA. This functionality behaves like the uneditable regular fields.

To set a cross-reference to be non-rerunnable, you need to either EDIT an old style non-regular cross-reference (you won't get the prompt the first time you are creating the field) or CREATE or EDIT a new style cross-reference (Regular and Mumps).

This is ^DIK's behavior for re-runnability:

- IXALL,IXALL2: Non-re-indexable not executed
- IX,IX1,IX2,EN,EN1,EN2: All executed.
- ENALL,ENALL2: Works properly when an index is specified; but doesn't execute old style xrefs when index isn't specified.

Expanded data storage

Fileman can now store an arbitrary amount of data into a Global Node. Setting ^DD("STIRNG_LIMIT") to a number containing the limit of length of global nodes of your Mumps database will allow you to store data with a combined length up to the limit. By default, if not set, Fileman limits any globals to 256 characters, the limit set by the last adopted Mumps standard, M 95 .

This is an important enhancement to Fileman as many data sets today (such as RxNorm and SNOMED) contain very long strings which are inappropriate for word processing fields. In addition, this makes loading delimited data into Fileman very easy for a programmer. Consider the following (GT.M code; use Cache equivalents to the Open, Use and Close commands on Cache or use ^%ZISH calls to open and close files):

```
N F S F="rrf/RXNCONSO.RRF" O F:(readonly:rewind) U F
N I,X F I=1:1 R X:0 Q:$ZEOF S ^DMU(1009.811,I,0)=$TR(X,"|^","^|")
C F U $P
N DIK S DIK="^DMU(1009.811," D IXALL^DIK
```

This code imports all of the public domain RxNorm Concepts into an existing empty Fileman file stored in ^DMU(1009.811) and then indexes the data. The data is delimited using pipes (“|”) and may contain a carets (“^”). The \$TR switches the pipes and carets to make the data safe for Fileman.

Programmers should note that storing long nodes may render your data less portable. As a guideline, as of the time of this writing, the following are the maximum lengths in Mumps Implementations the author is familiar with:

- Intersystems Cache 2013: 32 kilobytes
- GT.M V6.0: 1 Megabyte
- Mumps V1 V1.60 (not supported by Fileman): 32 kilobytes

Standalone Fileman Installation

Version 22.2 restores the ability for Fileman to run without a VISTA Kernel. Details on installing a Standalone Fileman instance can be found in the Install Guide.

Not all functions of Fileman work without a Kernel. For example, files cannot be imported or exported to the underlying OS file system as this is handled by the Kernel %ZISH routine.

DIFROM Enhancements

In version 22.0, Fileman did not have the ability to transport its new first-class data integrity elements, new style indexes and keys. While the code did actually reside in Fileman, Fileman couldn't use it transport its files. This version adds support for transporting new style indexes and keys to Fileman. Programmers do not have to do anything special to this to work; just use the routine ^DIFROM as before.

Unit Tests

This version of Fileman provides some Unit Tests for Fileman functionality. All the Unit Tests are located in the DMU namespace. You need all of routines for them to function. In addition, you need the M Unit package.

Currently, the following Unit Tests entry points are supplied:

```
D ^DMU DT000 ; Check %DT functionality
```

D ^DMUDTC00 ; Check %DTC functionality

D ^DMUDIC00 ; Check improvements to FIND and LIST^DIC introduced in Fileman 22.2

Conclusion

This new version of Fileman brings several important changes that will address the needs of Medical Data in the twenty-first century. These include:

- Expanded storage support for vocabularies with long indexable text entries
- Better support for internationalization and localization
- Better auditing support

Future efforts for Fileman will include support for extensible data types and better support for pseudo-pointers. In the internationalization arena, we hope to add full support for UTF-8. Fileman by and large support UTF-8, but it needs to be fully tested and certified.