VA FILEMAN V. 22
KEY AND INDEX TUTORIAL

September 2001

Revised December 2006

Revision History

Documentation History

The following table displays the revision history for this document. Revisions to the documentation are based on continuous dialog with Infrastructure and Security Services (ISS) Technical Writers and evolving industry standards and styles.

Date	Description	Author
Sept 2001	Initial release of the VA FileMan Version 22 Key and Index Tutoria.	Susan Strack, Oakland, CA OIFO; Michael Ogi, Oakland, CA OIFO;
Nov 2005	Reformatted document to follow ISS Style Guide. Reviewed document and edited for the "Data Scrubbing" and the "PDF 508 Compliance" projects. **Data Scrubbing**—Changed all patient/user TEST data to conform to HSD&D standards and conventions as indicated below: • The first three digits (prefix) of any Social Security Numbers (SSN) start with "000" or "666." • Patient or user names are formatted as follows: KRNPATIENT,[N] or KRNUSER,[N] respectively, where the N is a number written out and incremented with each new entry (e.g., KRNPATIENT, ONE, KRNPATIENT, TWO, etc.). • Other personal demographic-related data (e.g., addresses, phones, IP addresses, etc.) were also changed to be generic. **PDF 508 Compliance**—The final PDF document was recreated and now supports the minimum requirements to be 508 compliant (i.e., accessibility tags, language selection, alternate text for all images/icons, fully functional Web links, successfully passed Adobe Acrobat Quick Check).	Susan Strack, Oakland, CA OIFO
Dec 2006	Final edits and formatting. No content updates.	Susan Strack, Oakland, CA OIFO

Table i: Documentation History

Patch History

For the current patch history related to this software, please refer to the Patch Module on FORUM.

Contents

Contents

Orientation

All test data used in this tutorial (e.g., names, Social Security Numbers [SSN], dates of birth [DOB], etc.) is fictitious and is used solely for the purpose of illustrating lesson topics.

Formatting Conventions

Several formatting methods are used to highlight different aspects of this tutorial:

- "Snapshots" of computer online displays (i.e., roll-and-scroll screen captures/dialogues) and computer source code are shown in a non-proportional font.

    - User's responses to online prompts will be boldface.

    - Boldface type is also used to highlight a descriptive word or sentence.

    - The Enter (or Return) key is illustrated by the symbol **<Enter>** when displayed in computer dialogue and is included in examples only when it may be unclear to the reader that such a keystroke must be entered.

- All uppercase is reserved for the representation of M code, and field and file names (e.g., the SSN field).

This manual uses several methods to highlight different aspects of the material:

- Various terms are used throughout the documentation to alert the reader to special information. The following table gives a description of each:

Symbol	Description
	Used to inform the reader of general information including references to additional reading material
	Used to caution the reader to take special notice of critical information

Table ix: Documentation Symbol Descriptions

Presentation Structure

Environment Setup

The exercises in this tutorial make use of a simple test file referred to as *ZZINDIVIDUAL* that contains some sample datA. The A6AKIT set of routines can be used to install this test file on an M (MUMPS) system. Environment Setup contains information on how to obtain the A6AKIT routines. Lesson 1, Exercise 1.1. Create Your Test File shows you how to run the A6AKIT routine to install the test file.

Introduction

The tutorial introduction contains an overview of some of the basic key and index concepts.

Tutorial Lessons

Each section of this tutorial (Parts 1 through 3, shown below) contains topics and subsequent lessons. The information is presented in the following topic categories:

1. **Part 1** - Regular Indexes includes the following five lessons. A brief description is given for each.

   - Lesson 1 - You will create a New-style compound index based on the date of birth (DOB) and Social Security Number (SSN) fields in your *ZZINDIVIDUAL* test file.

   - Lesson 2 - You will print the definition of your New-style compound index to view the data.

   - Lesson 3 - You will see how your lookup index affects the behavior of the lookup utility IX^DIC. You will also change the collation and lookup prompt for the DOB field in your "C" index.

   - Lesson 4 - You will use the transform properties located in ScreenMan to perform transforms on subscripts.

   - Lesson 5 - You will create a whole-file index based on the fields in the EMAIL multiple of the *ZZINDIVIDUAL* test file. You will also illustrate a use for computed cross-reference values.

2. **Part 2** - MUMPS Indexes includes the following three lessons. A brief description is given for each.

   - Lesson 6 - You will learn about New-style MUMPS cross-references. It covers two topics: (1) the X, X1, and X2 arrays and (2) the set and kill logic and conditions of cross-references. You will create a New-style MUMPS cross-reference. Afterwards, you will check the behavior of your MUMPS cross-reference.

   - Lesson 7 - You will learn when VA FileMan executes the set and kill logic of New-style cross-references. You will also create a MUMPS cross-reference that makes use of the VA FileMan X, X1, and X2 arrays and test your new MUMPS cross-reference.

   - Lesson 8 - You will use the ACTIVITY property to suppress cross-reference execution.

3. **Part 3** - Keys includes the following three lessons. A brief description is given for each.

   - Lesson 9 - This lesson covers (1) Key Integrity, (2) the Uniqueness Index, and (3) Primary and Secondary Keys. You will create a key on the *ZZINDIVIDUAL* file.

   - Lesson 10 - You will print the definition of the key you created in Lesson 9 on the *ZZINDIVIDUAL* file with key fields NAME field (#.01) and SSN field (#.02). You will look at the Uniqueness Index automatically created and explore different methods for defining the fields in the key.

- Lesson 11 - You will see examples of how VA FileMan enforces the integrity of the key you created in Lesson 9, and worked with in Lesson 10.

Conventions for Displaying TEST Data in this Document are as Follows:

- The first three digits (prefix) of any Social Security Numbers (SSN) will begin with either "666" or "666".

- Patient and user names will be formatted as follows: [Application Name]PATIENT,[N] and [Application Name]USER,[N] respectively, where "Application Name" is defined in the Approved Application Abbreviations document, located on the [web site] and where "N" represents the first name as a number spelled out and incremented with each new entry. For example, in FileMan, test patient and user names would be documented as follows: FMPATIENT,ONE; FMPATIENT,TWO; FMPATIENT,10; etC. and FMUSER,ONE; FMUSER,TWO; FMUSER,10; etC.

The list of Approved Application Abbreviations can be found at the following Web site:

    http://vista.med.va.gov/iss/strategic_docs.asp#sop

Tutorial Quizzes

Each lesson in this tutorial is followed by a quiz. Each quiz is based on the lesson preceding it. They have been designed not only for you to use as a tool to test yourself on what you learned, but to offer a summary of the lesson. The quiz scores are not being recorded anywhere. They are for your benefit only.

Standard Data Dictionary Listing of the Test File

This section contains a standard data dictionary listing of the test file. In this listing, the name of the test file is ZZINDIVIDUAL, with file number 662nnn, stored in global root ^DIZ(662nnn).

Entries in the Tutorial Test File

This section contains a captioned printout, including record numbers, of the data in the ZZINDIVIDUAL test file installed by routine A6AKIT.

Environment Setup

M (MUMPS) Test Account and Tutorial Test File

An M test account is required, which runs VA FileMan Version 22.0, to take the lessons in this tutorial. You will also need to install a test file referred to as *ZZINDIVIDUAL* that contains some sample data. You will use the test file in the exercises in this tutorial to create and experiment with your own keys and indexes.

A set of A6AKIT routines is provided to install the tutorial test file on your M system. You can use one of the following methods to retrieve the A6AKIT routines.

Method 1: Accessing the Tutorial Test File from within the VA Intranet. Use the appropriate FTP utility to retrieve the text host file A6AKIT.TXT from the San Francisco anonymous directory:

OI Field Office	FTP Address	Directory	Host File Name
San Francisco	A1.ISC-SF.MED.VA.GOV	anonymous	A6AKIT.TXT

Method 2: Accessing the Tutorial Test File from both the Internet and VA Intranet. Method 2 is accessible directly from the Internet. People who have access to the VA Intranet can also access the tutorial test file via this method. Download the A6AKIT.TXT from the VistA Documentation Library (VDL) at the following address:

   http://www.va.gov/vdl/application.asp?appid=5 .

Once retrieved, you can use your M operating system's routine restore utility to load the routines A6AKIT, A6AKIT1, and A6AKIT2 from the A6AKIT.TXT file into your M account.

See Lesson 1, Exercise 1.1. Create Your Test File for instructions on how to run the A6AKIT routine to create your personal copy of the tutorial test file.

See the Appendix section of this tutorial to see the data dictionary for the tutorial test file and the file entries that it contains.

Introduction to Keys and Indexes

VA FileMan Version 22.0 introduced Key and Index functionality, which provides **V**_IST_**A** application developers the tools to design and develop more robust, efficient, and maintainable code and files. Native support for keys and complex indexes means developers no longer need to simulate these structures with M code.

With this hands-on tutorial, you will learn how you can take advantage of these powerful new tools. The instructions in the tutorial will guide you in creating your own keys and indexes on a test file set up via the A6AKIT set of routines. (See the Environment Setup page for information on how to obtain the A6AKIT routines.)

First, some definitions…

What is a Key?

A key is a set of one or more fields in a file that together uniquely identifies a record in that file. If you define a key, VA FileMan V. 22.0 automatically enforces the integrity of that key.

Key integrity means:

1. No key field is null.
2. The key (that is, the combination of fields in a key) is unique for all records in the file.

What is an Index?

An index is a sorted list of field values or associated data that "point" to a particular record in a file. It can be used to sort data and to look up a record via the indexed value, instead of searching the file sequentially.

In an index, VA FileMan stores the index name, the data value or values, and the internal entry number (IEN) of the record, each in its own subscript. To create an index in VA FileMan, you define a cross-reference of a given type.

What is a Cross-Reference?

A cross-reference identifies an action that should take place when the value of a field or fields defined in that cross-reference are changed. Often, the action is the placement of the value(s) into an index, in this case, the terms index and cross-reference are often used interchangeably. Technically speaking though, an index is a sorted list of records, and is built and maintained by defining a cross-reference(s).

Cross-references in Version 22 of VA FileMan can be divided into two broad categories: **Traditional** and **New-style**. Traditional cross-references are those cross-references you were able to create in versions of VA FileMan prior to Version 22, and can still create in Version 22. The new kind of cross-references you can create in Version 22 are called New-style cross-references.

What are Traditional Cross-References?

There are seven types of Traditional cross-references:

1. Regular (index)
2. Soundex (index)
3. KWIC (index)
4. Mnemonic (index)
5. MUMPS (index or action)
6. Trigger (action)
7. Bulletin (action)

The first four define indexes, while the last two define actions that should be performed when the field on which the cross-reference is defined is edited. A MUMPS cross-reference may set an index or perform an action, depending on how it's defined.

A Traditional cross-reference is defined on a single field, and its definition is stored under ^DD(file#,field#,1). In general, the logic for a Traditional cross-reference is executed when the field on which it's defined is edited.

What is a New-Style Cross-Reference?

There are two types of New-style cross-references:

1. Regular (index)
2. MUMPS (index or action)

A New-style cross-reference can be composed of one or more fields. Its definition is stored in the INDEX file (#.11), which has the global root ^DD("IX").

New-style cross-references that are composed of a single field are called **simple cross-references**, and by default have **field-level execution**, which means that the cross-reference logic is executed immediately after the field is edited.

New-style cross-references that are composed of more than one field are called **compound cross-references**, and by default have **record-level execution**. Record-level execution means that the cross-reference logic is executed only after an entire record is edited, after all the fields in the cross-reference have been edited.

Tutorial Introduction Quiz

This is the quiz for the VA FileMan V. 22.0 Key and Index Tutorial Introduction . Test yourself on what you've learned in the Introduction. The correct answers can be found at the bottom of the page, or by selecting the link at each question.

Question 1: VA FileMan Version 22.0 introduced native support for keys. All fields in a key for a given record must, when taken together, be unique for all records in the file, but individual key field values may be null.

A. True
B. False

Click here for answer

Question 2: What term does FileMan use to refer to the new kinds of cross-references you can create in Version 22.0?

A. New-style.
B. Modern.
C. Neither of the above.

Click here for answer

Question 3: What function does an index perform?

A. It provides a sorted list of field values.
B. It allows looking up entries via indexed values.
C. Both of the above.

Click here for answer

Question 4: A Regular cross-reference can perform some action other than setting and killing an index entry.

A. True
B. False

Click here for answer

Question 5: There are seven types of Traditional cross-references. How many types of New-style cross-references are there?

A. Two (Regular and MUMPS).
B. Seven (the same as Traditional).
C. Neither of the above.

Click here for answer

Question 6: The definition of a New-style cross-reference is stored in what file:

        A. INDEX file (#.11).
        B. KEY AND INDEX file (#.13).
        C. Neither of the above.

Click here for answer

Correct Answers – Tutorial Introduction Quiz

Question 1: VA FileMan Version 22.0 introduced native support for keys. All fields in a key for a given record must, when taken together, be unique for all records in the file, but individual key field values may be null.

Answer: B. "False." (Key integrity also implies that Key field values are not null.)

Go Back

Question 2: What term does FileMan use to refer to the new kinds of cross-references you can create in Version 22.0?

Answer: A. "New-style."

Go Back

Question 3: What function does an index perform?

Answer: C. "Both of the above." (An index provides both a sorted list of field values and allows looking up entries via indexed values.)

Go Back

Question 4: A Regular cross-reference can perform some action other than setting and killing an index entry.

Answer: B. "False." (A Regular cross-reference specifically only maintains an index. Hence, the terms Regular cross-reference and Regular index are often used interchangeably.)

Go Back

Question 5: There are seven types of Traditional cross-references. How many types of New-style cross-references are there?

Answer: A. "Two (Regular and MUMPS)."

Go Back

Question 6: The definition of a New-style cross-reference is stored in what file:

Answer: A. "INDEX file (#.11)."

Go Back

Part 1. Regular Indexes

Lesson 1. Create a Compound Index

In this lesson you will learn how to create a New-style Regular index. . The index you will create is a compound index, one that contains as subscripts data from two different fields.

    The only way you could have created a two-field compound index prior to Version 22.0 was to create two MUMPS cross-references, one on each field in the index. With Version 22.0, you can define the compound index as a single New-style Regular index.

There are only two types of New-style cross-references, Regular and MUMPS, but they are much more powerful than their Traditional cross-reference counterparts. In fact, most of the indexes you previously had to define as Traditional MUMPS cross-references can, with Version 22.0, be defined as New-style Regular indexes.

Exercise 1.1. Create Your Test File

The exercises in this tutorial make use of a simple test file that contains some sample data. In this exercise, you will use the A6AKIT routine to install your own personal copy of the test file. (See the Environment Setup page for information on how to obtain the A6AKIT set of routines.)

    In this tutorial, all references to the tutorial test file will be of file name *ZZINDIVIDUAL*, file number #*662nnn*, and global root *^DIZ*(*662nnn*,; however, you should always work with your own personal copy of the test file.

Steps to Install the Test File

Step 1.   Enter the following in programmer mode:

```
>D ^A6AKIT

I am going to set up a test file for the FileMan V. 22.0 Key and
Index Tutorial.

The tutorial assumes the name of the test file is ZZINDIVIDUAL and
the number of the test file is 662nnn, but you can choose any name
and number you wish.
```

Step 2. Select a name for the test file. The default is *ZZINDIVIDUAL*, but you can choose a different name, if you wish. Then choose a file number and a global root (internal global reference) for the file.

> **i** If the *ZZINDIVIDUAL* file already exists in the account in which you are taking this tutorial, don't assume you can use it. Someone else may be using that file. **You should use your own personal copy of the test file, with its own unique file name and number.**

```
Name of test file: ZZINDIVIDUAL// MY ZZINDIVIDUAL
  Are you adding 'MY ZZINDIVIDUAL' as a new FILE? No// Y  <Enter>
(Yes)
    FILE NUMBER: nnnn// nnnn

INTERNAL GLOBAL REFERENCE: ^DIZ(nnnn,// <Enter>

...HMMM, THIS MAY TAKE A FEW MOMENTS...
     A FreeText NAME Field (#.01) has been created.

File created: MY ZZINDIVIDUAL (#nnnn)
 Global root: ^DIZ(nnnn,

DONE!!
```

> **i** If at the "Name of test file:" prompt you choose an existing file that looks like a Key and Index Tutorial test file, A6AKIT gives you the opportunity of deleting that file, and replacing it with the original version of the test file.

End of Exercise 1.1.

Exercise 1.2. Create Your First Compound Index

In this exercise, you will create a New-style compound index based on the DOB and SSN fields in your *ZZINDIVIDUAL* test file.

Step 1. First, follow the menu path to the VA FileMan option Cross Reference a Field or File:

```
VA FileMan
   Utility Functions
      Cross Reference a Field or File
```

Step 2. Enter N for New at the prompt:

```
What type of cross-reference (Traditional or New)? Traditional//N
```

Step 3. Next, you are asked for the name of the file on which to create your cross-reference. Select the *ZZINDIVIDUAL* test file. Press Enter when prompted to select a subfile. Answer YES when asked if you want to create a new index on this file.

```
MODIFY WHAT FILE: ZZINDIVIDUAL// <Enter>
Select Subfile: <Enter>
```

> Because you are creating a cross-reference at the top level of the file, not at the subfile level, press the Enter key at this prompt to bypass any further subfile entries.

```
There are no INDEX file cross-references defined on file #662nnn.
Want to create a new Index for this file? No// YES
```

Step 4. Next, you are asked for the type of index you want to create. Press Enter to select the default index type REGULAR:

```
Type of index: REGULAR// <Enter>
```

> **i** You can create two types of New-style cross-references: Regular and MUMPS. MUMPS cross-references are discussed in Part 2 of this tutorial.

Step 5. You are now asked if you want your index to be used for Lookup and Sorting, or Sorting Only. Press Enter to select Lookup and Sorting:

```
Want index to be used for Lookup & Sorting
or Sorting Only: LOOKUP & SORTING// <Enter>
```

Step 6. Press Enter at the prompt to accept "C" as the index name:

```
Index Name: C// <Enter>
```

> VA FileMan determines a default name for your index from your response to the question in Step 5 (the type of index).

Step 7. On the two-page ScreenMan form you can edit the properties of the index you are creating. The first page shows you the responses to the questions you have already answered (i.e., the results of Steps 1 through 6).

The only field you must fill in here is the Short Description. Tab to the Short Description field and enter the following:

`This is a regular index on the DOB and SSN fields.`

```
Number: nnn                    EDIT AN INDEX                 Page 1 of 2
_____

            File: 662nnn                      Root File: 662nnn
      Index Name: C                           Root Type: INDEX FILE

 Short Description: This is a regular index on the DOB and SSN fields.
  Description (wp):  (empty)

            Type: REGULAR
                                              ┌─────────────────┐
        Activity: IR                          │ Here is your Short │
       Execution: FIELD                       │ Description.      │
                                              └─────────────────┘
             Use: LOOKUP & SORTING

_____

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      Insert
```
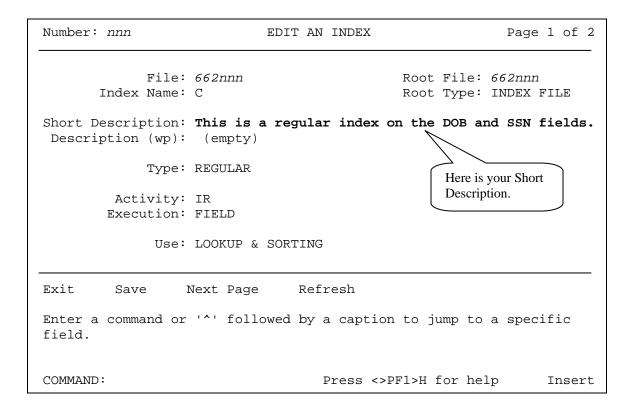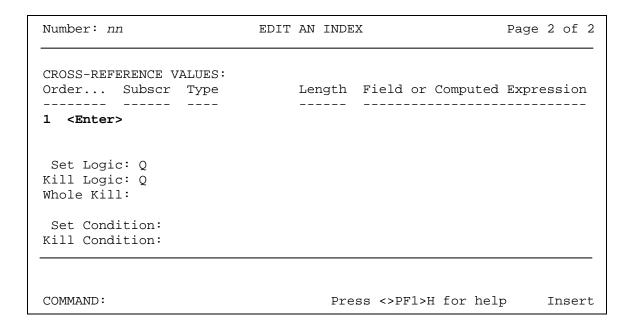
Tips: <CTRL-U> is a toggle between the values: null, last edited, and the default of the ScreenMan field you are editing.

Press **<PF1>Z** ("zoom") at any field to open up an editing window at the bottom of the screen. This editing window is often easier to use than the usual scrolling window, especially when the value of the field is very long, as is the case with the Set Logic.

Step 8. Page 2 of the ScreenMan form is used to enter field datA. Press <PF1><Down> or <PageDown> to go to page 2. Here you will specify the fields in your compound index.

In the "Order..." column, type in the number 1 and press Enter:

```
 ┌───────────────────────────────────────────────────────────────────────┐
 │                                                                         │
 │  Number: nn                    EDIT AN INDEX                Page 2 of 2  │
 │  ───────────────────────────────────────────────────────────────────   │
 │                                                                         │
 │  CROSS-REFERENCE VALUES:                                                │
 │  Order...  Subscr  Type           Length  Field or Computed Expression  │
 │  --------  ------  ----           ------  ----------------------------  │
 │  1  <Enter>                                                             │
 │                                                                         │
 │                                                                         │
 │                                                                         │
 │   Set Logic: Q                                                          │
 │  Kill Logic: Q                                                          │
 │  Whole Kill:                                                            │
 │                                                                         │
 │   Set Condition:                                                        │
 │  Kill Condition:                                                        │
 │  ───────────────────────────────────────────────────────────────────   │
 │                                                                         │
 │                                                                         │
 │  COMMAND:                              Press <>PF1>H for help    Insert  │
 └───────────────────────────────────────────────────────────────────────┘
```

Step 9. Tab or arrow to the "CROSS-REFERENCE VALUES TYPE OF VALUE:" prompt, enter "F" for FIELD:
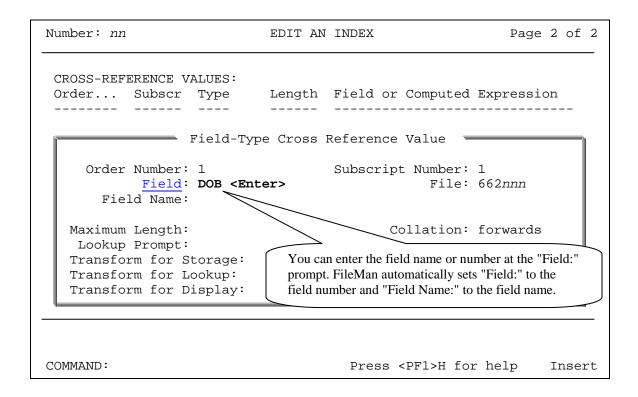
CROSS-REFERENCE VALUES TYPE OF VALUE: **F <Enter>**

This will bring up the pop-up window "Field-Type Cross Reference Value", shown in the next step.

ℹ Each cross-reference value can be a field or a computed value.

Step 10. At the "Field:" prompt in the pop-up window, enter DOB and press Enter:

```
 Number: nn                      EDIT AN INDEX               Page 2 of 2
 ───────────────────────────────────────────────────────────────────
  CROSS-REFERENCE VALUES:
  Order...  Subscr  Type     Length  Field or Computed Expression
  --------  ------  ----     ------  ----------------------------
        ┌──────────────── Field-Type Cross Reference Value ──────┐
        │                                                         │
        │   Order Number: 1             Subscript Number: 1       │
        │          Field: DOB <Enter>             File: 662nnn    │
        │     Field Name:                                         │
        │                                                         │
        │  Maximum Length:                      Collation: forwards│
        │   Lookup Prompt:                                        │
        │  Transform for Storage:        You can enter the field name or number at the "Field:"
        │  Transform for Lookup:         prompt. FileMan automatically sets "Field:" to the
        │  Transform for Display:        field number and "Field Name:" to the field name.
        └─────────────────────────────────────────────────────────┘
 ───────────────────────────────────────────────────────────────────

  COMMAND:                            Press <PF1>H for help    Insert
```

At this point we have entered to first field (DOB) of our two field compound index.

Step 11. Press <PF1>E to close the pop-up window.

> VA FileMan has automatically built the Set and Kill logic for your new cross-reference based on the DOB cross-reference value you added to the index.

Step 12. Tab to the second row of the table, type the number 2 in the "Order…" column and press Enter.

Step 13. At the "CROSS-REFERENCE VALUES TYPE OF VALUE:" prompt, enter "F" for FIELD.

Step 14. At the "Field:" prompt on the pop-up window, enter SSN and press <PF1>E or <PF1>C to close the "Field-Type Cross Reference Value" pop-up window.

At this point we have entered the second and last field (SSN) of our two field compound index.

After you've added these two cross-reference values, page 2 should look like this:

```
 Number: nnn                  EDIT AN INDEX                Page 2 of 2
 _____

  CROSS-REFERENCE VALUES:
  Order...  Subscr  Type     Length  Field or Computed Expression
  --------  ------  ----     ------  ------------------------------
  1         1       FIELD            DOB (#.03)
  2         2       FIELD      30    SSN (#.02)


   Set Logic: S ^DIZ(662nnn,"C",X(1),$E(X(2),1,30),DA)=""
  Kill Logic: K ^DIZ(662nnn,"C",X(1),$E(X(2),1,30),DA)
  Whole Kill: K ^DIZ(662nnn,"C")

   Set Condition:
  Kill Condition:
```

FileMan automatically builds the Set and Kill logic for your new cross-reference based on the DOB and SSN cross-reference values you added to the index.

```
 COMMAND:                             Press <PF1>H for help  Insert
```

Your index contains two fields, DOB and SSN, each of which is used as a subscript in your index. Notice above that the Set and Kill Logic is automatically generated for you. X(1) refers to your DOB field, and X(2) refers to your SSN field. The subscripts 1 and 2 in the X array correspond to the Order numbers of the cross-reference values.

Step 15. Press <PF1>E to exit the ScreenMan form.

Step 16. Press Enter at the prompt "Do you want to build the index now? YES//". This will build the new "C" index on your file.

Congratulations! You have just created your first New-Style cross-reference!

Lesson 1 Quiz

This is the quiz for Lesson 1 of the VA FileMan V. 22.0 Key and Index Tutorial. Test yourself on what you've learned in Lesson 1. The correct answers can be found at the bottom of the page, or by selecting the link at each question.

Question 1: A Regular index is one of the two types of New-style cross-references you can create. How is data stored in a Regular index?

    A. As a subscript(s) in the index.
    B. As an index entry set to an up-arrow delimited string
    C. Neither of the above.

Click here for answer

Question 2: The name of a Regular index (like 'C') becomes a subscript in the index itself.

    A. True
    B. False

Click here for answer

Question 3: A New-style cross-reference can be composed of:

    A. One field.
    B. More than one field.
    C. Both of the above.

Click here for answer

Question 4: A New-style cross-reference that is composed of a single field is called a:

    A. Common cross-reference.
    B. Simple cross-reference.
    C. Both of the above.

Click here for answer

Question 5: Field-level execution means that the cross-reference logic is executed immediately after:

    A. The entire record is edited.
    B. Each and every field in the cross-reference is edited.
    C. Neither of the above.

Click here for answer

Question 6: A New-style cross-reference that is composed of more than one field is called a:

        A. Compound cross-reference.
        B. Traditional cross-reference.
        C. Both of the above.

Click here for answer

Question 7: Record-level execution means that the cross-reference logic is executed immediately after:

        A. The entire record is edited.
        B. After each and every field in the cross-reference is edited.
        C. Neither of the above.

Click here for answer

Question 8: For Regular indexes, FileMan automatically generates the Set and Kill Logic for you.

        A. True.
        B. False.

Click here for answer

Lesson 1. Correct Answers to Quiz

Question 1: A Regular index is one of the two types of New-style cross-references you can create. How is data stored in a Regular index?

Answer: A. "As subscript(s) in the index."

Go Back

Question 2: The name of a Regular index (like 'C') becomes a subscript in the index itself.

Answer: A. "True."

Go Back

Question 3: A New-style cross-reference can be composed of:

Answer: C. "Both of the above." (One field and more than one field.)

Go Back

Question 4: A New-style cross-reference that is composed of a single field is called a:

Answer: B. "Simple cross-reference."

Go Back

Question 5: Field-level execution means that the cross-reference logic is executed immediately after:

Answer: B. "Each and every field in the cross-reference is edited."

Go Back

Question 6: A New-style cross-reference that is composed of more than one field is called a:

Answer: A. "Compound cross-reference."

Go Back

Question 7: Record-level execution means that the cross-reference logic is executed immediately after:

Answer: A. "The entire record is edited."

Go Back

Question 8: For Regular indexes, FileMan automatically generates the Set and Kill Logic for you.

Answer: A. "True." (In fact, you cannot edit the automatically generated logic for Regular indexes.)

Go Back

Lesson 2. Print the Definition of Your New Index to View Data

VA FileMan provides several data dictionary listing formats to view the definition of a file. In Version 22, the Standard data dictionary listing has been modified to display New-style indexes. In addition, a new Indexes Only type of listing displays only the cross-references, both new and Traditional, as they are defined on a file.

Exercise 2.1. See the New-Style Index You Created in Lesson 1

Step 1.   First, follow the menu path to the data dictionary listing option List File Attributes:

```
VA FileMan
   Data Dictionary
      List File Attributes
```

Step 2.   Select your *ZZINDIVIDUAL* test file:

```
START WITH WHAT FILE: ZZINDIVIDUAL
     GO TO WHAT FILE: ZZINDIVIDUAL// <Enter>
     Select SUB-FILE: <Enter>
Select LISTING FORMAT: STANDARD// ??
Choose from:
     1              STANDARD
     2              BRIEF
     3              CUSTOM-TAILORED
     4              MODIFIED STANDARD
     5              TEMPLATES ONLY
     6              GLOBAL MAP
     7              CONDENSED
     8              INDEXES ONLY
     9              KEYS ONLY
Select LISTING FORMAT: STANDARD// INDEXES ONLY
What type of cross-reference (Traditional or New)? Both// <Enter> BOTH
Which field: ALL// <Enter>
```

> Enter two question marks (??) to see the choices of listing types.

> Select the INDEXES ONLY format.

Step 3.   At the "Device:" prompt, enter ;80;999 to display the report on your terminal:

```
 DEVICE: ;80;999  <Enter>
```

The following is a report of all the cross-references defined on the *ZZINDIVIDUAL* test file:

```
 INDEX AND CROSS-REFERENCE LIST -- FILE #662nnn    09/07/00  PAGE 1
 ---------------------------------------------------------------

 File #662nnn
```

```
             Traditional Cross-References:

             B       REGULAR
                     Field:  NAME   (662nnn,.01)
                             1)= S ^DIZ(662nnn,"B",$E(X,1,30),DA)=""
                             2)= K ^DIZ(662nnn,"B",$E(X,1,30),DA)


             New-Style Indexes:

             C  (#nnn)     RECORD     REGULAR      IR    LOOKUP & SORTING
                Short Descr:  This is a Regular index on the DOB and SSN fields.
                    Set Logic:    S ^DIZ(662nnn,"C",X(1),$E(X(2),1,30),DA)=""
                    Kill Logic:   K ^DIZ(662nnn,"C",X(1),$E(X(2),1,30),DA)
                    Whole Kill:   K ^DIZ(662nnn,"C")
                        X(1):   DOB (662nnn,.03) (Subscr 1) (forwards)
                        X(2):   SSN (662nnn,.02) (Subscr 2) (Len 30) (forwards)
```

For regular indexes, the set and kill logic is generated automatically for you when you create the index. This code sets and kills an index entry.

```
             Subfile #662nnn.02

             Traditional Cross-References:

             B       REGULAR
                     Field:  ALIAS   (662nnn.02,.01)
                             1)= S ^DIZ(662nnn,DA(1),2,"B",$E(X,1,30),DA)=""
                             2)= K ^DIZ(662nnn,DA(1),2,"B",$E(X,1,30),DA)
```

How to Look at the Data in Your Index

From programmer mode, do a global listing and look at the data in the "C" index of the *ZZINDIVIDUAL* file as shown below:

```
    >D ^%G <Enter>

    Device: <Enter> Right margin: 80=> <Enter>

    Global ^DIZ(662nnn,"C" <Enter>
            DIZ(662nnn,"C"
    ^DIZ(662nnn,"C",2191031,666345678,9)=
    ^DIZ(662nnn,"C",2320205,666443333,2)=
    ^DIZ(662nnn,"C",2380409,666432123,10)=
    ^DIZ(662nnn,"C",2450520,666567890,13)=
    ^DIZ(662nnn,"C",2450520,"000221111",1)=
    ^DIZ(662nnn,"C",2470522,666996666,12)=
    ^DIZ(662nnn,"C",2480811,666678901,7)=
    ^DIZ(662nnn,"C",2550602,666889999,6)=
    ^DIZ(662nnn,"C",2590714,666776666,3)=
    ^DIZ(662nnn,"C",2691123,666223333,4)=
    ^DIZ(662nnn,"C",2700321,666765432,11)=
    ^DIZ(662nnn,"C",2710227,666654321,5)=
    ^DIZ(662nnn,"C",2730926,666891234,8)=
```

```
Global ^
```

As you can see, the first subscript after the index name "C" contains the data from the DOB field. The next subscript contains the data from the SSN field. The last subscript in the index is the internal entry number of the indexed record.

Prior to Version 22, the only way to create an index such as this was to create two MUMPS cross-references, one on the DOB field, and one on the SSN field. In fact, New-style Regular indexes are so powerful, the only time you should need to create a MUMPS cross-reference is for a cross-reference that performs some action, rather than sets an index.

End of Exercise 2.1.

Lesson 2. Quiz

This is the quiz for Lesson 2 of the VA FileMan V. 22.0 Key and Index Tutorial. Test yourself on what you've learned in Lesson 2. The correct answers can be found at the bottom of the page, or by selecting the link at each question.

Question 1: Which VA FileMan option would you use to print data dictionary listings for a given file?

      A. Inquire To File Entries.
      B. List File Attributes.
      C. Both of the above.

Click here for answer

Question 2: The Indexes Only format on the data dictionary listing option List File Attributes shows:

      A. The Traditional and New-style cross-references that are defined on a file.
      B. Only the New-style cross-references that are defined on a file.
      C. Neither of the above.

Click here for answer

Question 3: In a New-style Regular index, the index name:

      A. Is used as a subscript in the index.
      B. Is equivalent to the Short Description.
      C. Neither of the above.

Click here for answer

Question 4: In a New-style Regular index, the subscript(s) after the index name always contain(s):

      A. A record number.
      B. Indexed data.
      C. Neither of the above.

Click here for answer

Question 5: In a New-style Regular index, the last subscript in the index is always:

      A. The index name.
      B. A record number.
      C. Neither of the above.

Click here for answer

Correct Answers – Lesson 2 Quiz

Question 1: Which VA FileMan option would you use to print data dictionary listings for a given file?

Answer: B. "List File Attributes."

Go Back

Question 2: The Indexes Only format on the data dictionary listing option List File Attributes shows:

Answer: A. "The Traditional and New-style cross-references that are defined on a file."

Go Back

Question 3: In a New-style Regular index, the index name:

Answer: A. "Is used as a subscript in the index."

Go Back

Question 4: In a New-style Regular index, the subscript(s) after the index name always contain(s):

Answer: B. "Indexed data."

Go Back

Question 5: In a New-style Regular index, the last subscript in the index is always:

Answer: B. "A record number."

Go Back

Lesson 3. VA FileMan Lookups Using Compound Indexes

In this lesson you will see how your Regular index affects the behavior of the lookup utility IX^DIC. You will also change the collation and lookup prompt for the DOB field in your "C" index. The following topics are covered with exercises following each:

1. Interactive Lookups Using Compound Indexes.
2. Passing Lookup Values for Lookups Using Compound Indexes.
3. Collation Property and Lookup Prompt Property.

Interactive Lookups Using Compound Indexes

When you use a compound index for an interactive lookup, FileMan prompts you for values for each of the subscripts in the index.

Exercise 3.1. Use the New-Style Index in an Interactive IX^DIC Call

In this lesson you will see how your lookup index affects the behavior of the lookup utility IX^DIC. You will also change the collation and lookup prompt for the DOB field in your "C" index.

Step 1. From programmer mode, set up the input variables and perform an IX^DIC call using the New-style "C" index. Afterward, enter two question marks (??) at the "Select *ZZINDIVIDUAL* DOB:" prompt.

```
>S DIC="^DIZ(662nnn,",DIC(0)="QEAZ",D="C"
>D IX^DIC

Select ZZINDIVIDUAL DOB: ??

   Choose from:
   OCT 31, 1919    666345678      FMPATIENT,NINE
   FEB 05, 1932    666443333      FMPATIENT,TWO
   APR 09, 1938    666432123      FMPATIENT,TEN
   MAY 20, 1945    666567890      FMPATIENT,THIRTEEN
   MAY 20, 1945    000221111      FMPATIENT,ONE
   MAY 22, 1947    666996666      FMPATIENT,TWELVE
   AUG 11, 1948    666678901      FMPATIENT,SEVEN
   JUN 02, 1955    666889999      FMPATIENT,SIX
   JUL 14, 1959    666776666      FMPATIENT,THREE
   NOV 23, 1969    666223333      FMPATIENT,FOUR
   MAR 21, 1970    666765432      FMPATIENT,ELEVEN
   FEB 27, 1971    666654321      FMPATIENT,FIVE
   SEP 26, 1973    666891234      FMPATIENT,EIGHT

   Select ZZINDIVIDUAL DOB:
```

ℹ️ In the listing above FileMan displays the values stored in the index, DOB and SSN, as well as the value of the NAME field (#.01).
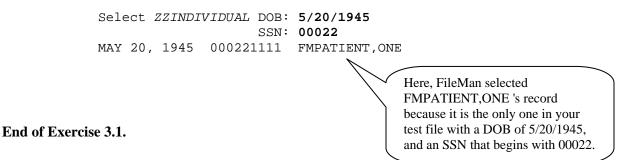
Step 2. At the "Select *ZZINDIVIDUAL* DOB:" prompt enter 5/20/1945. Press Enter at the "SSN:" prompt, then press Enter again at the "CHOOSE 1-2:" prompt.

> After you've entered the DOB, FileMan prompts you for SSN, the second cross-reference value in the "C" index.

```
Select ZZINDIVIDUAL DOB: 5/20/1945
                   SSN: <Enter>
     1   MAY 20, 1945  666567890  FMPATIENT,THIRTEEN
     2   MAY 20, 1945  000221111  FMPATIENT,ONE
CHOOSE 1-2: <Enter>
```

> Since you entered nothing at the "SSN:" prompt, FileMan displays only the two records in your test file that have DOBs of 5/20/1945.

Step 3. At the "Select *ZZINDIVIDUAL* DOB:" prompt enter 5/20/1945, then at the "SSN:" prompt enter 00022:

```
Select ZZINDIVIDUAL DOB: 5/20/1945
                    SSN: 00022
 MAY 20, 1945  000221111  FMPATIENT,ONE
```

> Here, FileMan selected FMPATIENT,ONE 's record because it is the only one in your test file with a DOB of 5/20/1945, and an SSN that begins with 00022.

End of Exercise 3.1.

Passing Lookup Values for Lookups Using Compound Indexes

Suppose you want to perform a lookup on the "C" index, but pass the lookup values to IX^DIC, rather than prompt the user for the values. Normally, in this kind of non-interactive lookup, you would pass the lookup value in the variable X. But with a compound index, you might want to look up a record based on the values of more than one field. FileMan allows you to pass in multiple values via the X(*n*) array, where n corresponds to the *subscript* number in the index. You can set X(1) to the lookup value for the data in *subscript* 1 (after the index name), X(2) to the lookup value for the data in *subscript* 2, etC.

Exercise 3.2. Use the New-Style Index in a Non-Interactive IX^DIC Call

In this exercise, you will make a non-interactive call to IX^DIC, passing in the lookup values in the X(*subscript*#) array.

Step 1. Set up the input variables for a non-interactive IX^DIC call using the "C" index:

```
>S DIC="^DIZ(662nnn,",DIC(0)="QZ",D="C"
```

Step 2. Set the X array to the lookup values shown below, and perform an IX^DIC call:

```
>S X(1)="5/20/1945",X(2)=00022
>D IX^DIC
```

Step 3. Enter ZWRITE (ZW) Y at the programmer mode prompt to see the results of the lookup.

```
>ZW Y <Enter>
Y="1^FMPATIENT,ONE"
Y(0)="FMPATIENT,ONE^000221111^2450520"
Y(0,0)="FMPATIENT,ONE"
```

End of Exercise 3.2.

Collation Property

New-style indexes allow you to control the direction in which FileMan traverses an index when it displays entries to the user. This is done via the **Collation** property of cross-reference subscripts. Normally, you would display entries in ascending alphabetical order, and so the collation for most subscripts is **forward**. But if you would like FileMan to traverse a subscript in an index in reverse order, you could change the collation of the subscript to **backward**.

Lookup Prompt Property

The Lookup prompt property of cross-reference subscripts can also be changed so that interactive lookups will use that string, rather than the field Label, to prompt the user for a lookup value.

Exercise 3.3. Change Collation and Lookup Prompt for DOB in "C" Index

In this exercise, you will change the collation of the DOB subscript in our "C" index so that most recent DOBs are displayed first. You will also change the Lookup prompt to DATE OF BIRTH instead of DOB.

Step 1. Follow the menu path to the option Cross Reference a Field or File:

```
VA FileMan
   Utility Functions
      Cross Reference a Field or File
```

Step 2. Select NEW for the type of cross-reference, then select your *ZZINDIVIDUAL* test file at the "MODIFY WHAT FILE:" prompt. Press Enter at the "Select Subfile:" prompt.

```
What type of cross-reference (Traditional or New)? Traditional// NEW

MODIFY WHAT FILE: // ZZINDIVIDUAL
Select Subfile: <Enter>

Current Indexes on file #662nnn:
  nnn    'C' Index
```

> Here you're presented with a list of new-style cross-references already defined on your test file.

```
Choose E (Edit)/D (Delete)/C (Create): E <Enter>
Which Index do you wish to edit? nnn// C <Enter>
```

Step 3. Next, you are presented the ScreenMan form where you can edit the properties of your "C" index. Press <PF1>Down or <PageDown> to go to page 2.

Step 4. Press Enter on the number 1 in the "Order…" column to edit the properties of the cross-reference value with Order number 1, the DOB field. A pop-up page opens where you can edit the properties of the DOB cross-reference value. Tab to Collation and enter **B** for backwards and press Enter.

Step 5. Tab to the "Lookup Prompt:" field, enter: **DATE OF BIRTH**.

Step 6. Press <PF1>E to close the pop-up page. Press <PF1>E again to save changes and exit the form.

Step 7. From programmer mode, set up the input variables and perform an IX^DIC call using the "C" index. Afterward, enter two question marks (??) at the "Select *ZZINDIVIDUAL* DATE OF BIRTH:" prompt.

```
>S DIC="^DIZ(662nnn,",DIC(0)="QEAZ",D="C"
>D IX^DIC

Select ZZINDIVIDUAL DATE OF BIRTH: ??

  Choose from:
   SEP 26, 1973   666891234      FMPATIENT,EIGHT
   FEB 27, 1971   666654321      FMPATIENT,FIVE
   MAR 21, 1970   666765432      FMPATIENT,ELEVEN
   NOV 23, 1969   666223333      FMPATIENT,FOUR
```

> The change to this prompt "Select ZZINDIVIDUAL DATE OF BIRTH" resulted from changing the "Lookup:" prompt property of the DOB cross-reference value.

```
        JUL 14, 1959   666776666      FMPATIENT,THREE
        JUN 02, 1955   666889999      FMPATIENT,SIX
        AUG 11, 1948   666678901      FMPATIENT,SEVEN
        MAY 22, 1947   666996666      FMPATIENT,TWELVE
        MAY 20, 1945   666567890      FMPATIENT,THIRTEEN
        MAY 20, 1945   000221111      FMPATIENT,ONE
        APR 09, 1938   666432123      FMPATIENT,TEN
        FEB 05, 1932   666443333      FMPATIENT,TWO
        OCT 31, 1919   666345678      FMPATIENT,NINE
```

End of Exercise 3.3.

In contrast to Exercise 3.1, this list is displayed in the order of the most recent DOB first. This was achieved by specifying a backward collation for the DOB cross-reference value. Backward collation causes FileMan to use a reverse $Order to traverse the data at the DOB subscript level.

Lesson 3. Quiz

This is the quiz for Lesson 3 of the VA FileMan V. 22.0 Key and Index Tutorial. Test yourself on what you've learned in Lesson 3. The correct answers can be found at the bottom of the page, or by selecting the link at each question.

Question 1: When using a compound index in an interactive lookup:

      A. FileMan prompts you for the values for each of the data subscripts in the index.
      B. FileMan expects you to pass the data subscript values in the index to IX^DIC.
      C. Both of the above.

Click here for answer

Question 2: You can look up a record based on more than one field value in a compound index:

      A. By passing the values in the variable X.
      B. By passing the values in the X(n) array.
      C. Neither of the above.

Click here for answer

Question 3: During a lookup, what does the 'n' in the X(n) array correspond to in the index?

      A. The subscript number.
      B. The field number.
      C. Neither of the above.

Click here for answer

Question 4: How do you use the X(n) array to lookup values in multiple data subscripts for compound indexes?

      A. Set X(1) to the lookup value for the data in subscript 1, X(2) to the lookup value for the data in subscript 2, etc.
      B. Set X(1) to the lookup value for the combined data in both subscripts 1 and 2.
      C. Neither of the above.

Click here for answer

Question 5: The Collation property of cross-reference subscripts for New-style indexes allows you to:

      A. Control the direction in which FileMan traverses a subscript in an index when it displays entries to the user.
      B. Control whether entries are displayed in MUMPS or ASCII collating sequence.
      C. Both of the above.

Click here for answer

Question 6: What cross-reference value property allows you to control the prompt issued to the user during an interactive lookup?

    A. Transform for Lookup.
    B. Lookup Prompt.
    C. Neither of the above.

Click here for answer

Correct Answers – Lesson 3 Quiz

Question 1: When using a compound index in an interactive lookup:

   Answer: A. "FileMan prompts you for the values for each of the data subscripts in the index."

Go Back

Question 2: You can look up a record based on more than one field value in a compound index:

   Answer: B. "By passing the values in the X(n) array."

Go Back

Question 3: During a lookup, what does the 'n' in the X(n) array correspond to in the index?

   Answer: A. "The subscript number."

Go Back

Question 4: How do you use the X(n) array to lookup values in multiple data subscripts for compound indexes?

   Answer: A. "Set X(1) to the lookup value for the data in subscript 1, X(2) to the lookup value for the data in subscript 2, etc."

Go Back

Question 5: The Collation property of cross-reference subscripts for New-style indexes allows you to:

   Answer: A. "Control the direction in which FileMan traverses a subscript in an index when it displays entries to the user."

Go Back

Question 6: What cross-reference value property allows you to control the prompt issued to the user during an interactive lookup?

   Answer: B. "Lookup Prompt."

Go Back

Lesson 4. Transforms on Subscripts

In this lesson you will learn about the three different transforms you can define on cross-reference values, and how they can help make your indexes more usable.

Three Types of Transforms for Cross-Reference Values

These are the three types of transforms you can perform in VA FileMan on cross-reference values used as subscripts in an index:
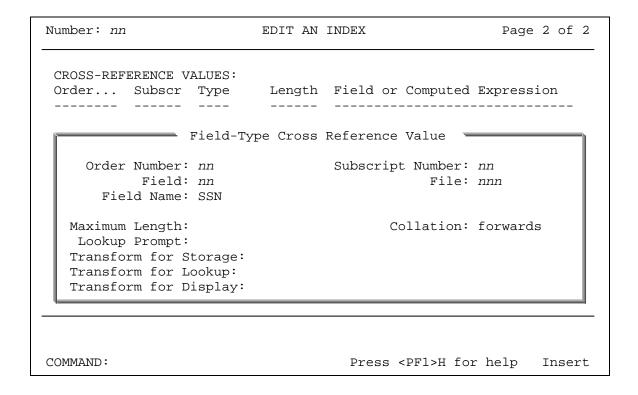
1. **Transform for Storage:** You can use Transform for Storage to transform the internal value of the field before it is stored as a subscript in the index. In the exercises in this lesson, you will create a New-style Regular "D" index on the SSN field and use the Transform for Storage to append a space to the end of the SSN before it's stored in the index. This will make all SSNs noncanonic, so that they will all sort in the ASCII collation sequence.

2. **Transform for Lookup:** When VA FileMan uses an index in a lookup, it first looks up the value exactly as it was input. If it can't find the lookup value as it was originally input in the index, it performs some basic transformations on the value, as shown below:

    *   Converts the lookup value to all uppercase,
    *   Truncates long input to the maximum subscript length,
    *   Performs comma-piecing, or
    *   Converts values to internal form for the following fields: date, set of codes, pointer, and variable pointers.

    Once this conversion has taken place, VA FileMan tries the lookup again - but this time, looking for the transformed value.

    With New-style indexes, you can define your own transform that VA FileMan will perform in addition to its usual transforms. In Exercise 4.2 of this lesson, you will use the Transform for Lookup in your "D" index to strip out any hyphens from the SSN entered by the user. For example, if the user enters "123-45-6789" VA FileMan would strip the hyphens and also look for 666456789 in the index.

3. **Transform for Display:** If a subscript in the index has a Transform for Storage, you would probably also want to define a Transform for Display that takes the value as it is stored in the index and converts it back into a form that can be displayed to the user. In Exercise 4.2 of this lesson, you will use the Transform For Display property to take the space out of the SSN. (This is not really necessary in our case, since the extra space displayed at the end of the SSN wouldn't confuse the user. However, we'll strip it anyway for illustration purposes.)

The Transform properties are located in the "Field-Type Cross Reference Value" pop-up window, shown in the next screen capture.

```
 ┌──────────────────────────────────────────────────────────────────────────┐
 │  Number: nn                   EDIT AN INDEX                Page 2 of 2     │
 │  ──────────────────────────────────────────────────────────────────────  │
 │                                                                           │
 │   CROSS-REFERENCE VALUES:                                                 │
 │   Order...   Subscr  Type     Length  Field or Computed Expression        │
 │   --------   ------  ----     ------  -----------------------------        │
 │                                                                           │
 │        ┌──────────────── Field-Type Cross Reference Value ───────────┐    │
 │        │                                                             │    │
 │        │    Order Number: nn              Subscript Number: nn        │    │
 │        │           Field: nn                        File: nnn         │    │
 │        │      Field Name: SSN                                         │    │
 │        │                                                             │    │
 │        │  Maximum Length:                      Collation: forwards    │    │
 │        │   Lookup Prompt:                                            │    │
 │        │  Transform for Storage:                                     │    │
 │        │  Transform for Lookup:                                      │    │
 │        │  Transform for Display:                                     │    │
 │        │                                                             │    │
 │        └─────────────────────────────────────────────────────────────┘    │
 │  ──────────────────────────────────────────────────────────────────────  │
 │                                                                           │
 │                                                                           │
 │   COMMAND:                             Press <PF1>H for help   Insert      │
 └──────────────────────────────────────────────────────────────────────────┘
```

Exercise 4.1. Create a New-Style Regular "D" Index on the SSN Field

To illustrate how and why you might use these transforms, you will create another New-style index, this time using the SSN field as the cross-reference value.

Step 1.   Create a New-style Regular "D" index on the SSN field. Here's an abbreviated procedure for doing this:

1. Select **UTILITY FUNCTIONS** and **CROSS-REFERENCE A FIELD OR FILE** on the VA FileMan menu.
2. Select **NEW** for type of cross-reference.
3. Select the *ZZINDIVIDUAL* file.
4. Choose **C** to create a new index.
5. Enter **REGULAR** for type of index.
6. Enter **LOOKUP & SORTING** for how the index should be used.
7. Enter **D** for the index name.
8. On page 1 of the ScreenMan form, at the "Short Description:" field enter **A simple index on the SSN field**.
9. On page 2, in the "Order..." column type **1** and press Enter.
10. For "Cross-Reference Values Type of Value," enter **FIELD**.
11. In the pop-up window, for Field, enter **SSN**.
12. Press <PF1>E twice and press Enter until you return to programmer mode.

i   Refer to Lesson 1, "Exercise 1.1. Create Your First Compound Index" for more information on creating a New-style Regular index on a field. However, don't forget to use these values.

The resulting Set and Kill Logic should look like this:

```
  Set Logic: S ^DIZ(662nnn,"D",$E(X,1,30),DA)=""
 Kill Logic: K ^DIZ(662nnn,"D",$E(X,1,30),DA)
 Whole Kill: K ^DIZ(662nnn,"D")
```

Step 2. This step and the next will illustrate two problems with the "D" index that you just created. The first is that the SSNs aren't sorted the way that you would expect. The second is that SSNs entered with hyphens are rejected.

From programmer mode, set up the input variables using the "D" index and perform an IX^DIC call. Afterward, enter two question marks (??) at the "Select *ZZINDIVIDUAL* SSN:" prompt to see a list of records in your test file as they are currently sorted in the "D" index.

```
>S DIC="^DIZ(662nnn,",DIC(0)="QEAZ",D="D"
>D IX^DIC

Select ZZINDIVIDUAL SSN: ??

 Choose from:
 666223333      FMPATIENT,FOUR
 666345678      FMPATIENT,NINE
 666432123      FMPATIENT,TEN
 666443333      FMPATIENT,TWO
 666567890      FMPATIENT,THIRTEEN
 666654321      FMPATIENT,FIVE
 666678901      FMPATIENT,SEVEN
 666765432      FMPATIENT,ELEVEN
 666776666      FMPATIENT,THREE
 666889999      FMPATIENT,SIX
 666891234      FMPATIENT,EIGHT
 666996666      FMPATIENT,TWELVE
 000221111      FMPATIENT,ONE
```

The record with the SSN of 000221111 appears at the end of the list. This is because in M, all canonic numbers sort before strings of characters that are noncanonic. A canonic number is a number that contains no nonsignificant leading zeros, no plus signs (+) in the case of positive numbers, and no insignificant trailing zeros to the right of the decimal point in the case of real numbers. The SSNs 666223333 through 666996666 are canonic numbers and are ordered numerically. All of them precede 000221111, which is noncanonic.

i   In exercise 4.2, you will define a Transform for Storage for the SSN field to have VA FileMan store the SSNs as non-canonic strings. This will make the SSN of 000221111 appear at the beginning of the list.

Step 3. At the "Select *ZZINDIVIDUAL* SSN:" prompt, enter 666-22-3333:

```
Select ZZINDIVIDUAL SSN: 666-22-3333 <Enter> ??
```

> Since the SSNs are stored in the "D" index without hyphens, VA FileMan is unable to find the SSN in this format: 666-22-3333.

ⓘ In exercise 4.2, you will define a Transform for Lookup that will allow you to look up an SSN, even if you enter it with hyphens.

End of Exercise 4.1.

Exercise 4.2. Modify the "D" Index to Include Transforms on the SSN Subscript

In this exercise you will modify the "D" index created in Exercise 4.1 so that:

1. The list of SSNs displayed on ?? is sorted as you would expect, regardless of whether the SSN is canonic or noncanonic.

2. Input of SSNs with hyphens (-) is accepted.

To do this you will use the three Transform properties of cross-reference values:

1. The Transform for Storage to force each SSN stored in the index to be noncanonic by appending a space at the end.

2. The Transform for Lookup to remove hyphens (-) from the lookup value.

3. The Transform for Display to remove the space at the end of the SSN.

Step 1. Edit the "D" index, and on page 2 of the ScreenMan form, press Enter in the "Order…" column for Order number 1 to select the SSN cross-reference value.

Step 2. In the pop-up window, Tab to the "Transform for Storage" and enter the following M code:

```
S X=X_" "
```

(This concatenates a space at the end of each SSN.)

Step 3. At the "Transform for Lookup", enter the following M code:

```
S X=$TR(X,"-")
```

(This removes all hyphens from any SSN entered by the user.)

Step 4. At the "Transform for Display", enter the following M code:

```
S:$E(X,$L(X))=" " X=$E(X,1,$L(X)-1)
```

(This removes the space, previously concatenated in Step 2, at the end when displaying the SSN.)

Step 5. Press <PF1>E to close the pop-up window. Press <PF1>E again to save changes and exit the form. Answer YES to the following two questions:

```
Do you want to delete the data in the old index now? YES// <Enter>
   Removing old index ... DONE!

Do you want to build the index now? YES// <Enter>
   Building new index ... DONE!
```

> **i** If you haven't installed Patch DI*22.0*58, you will not see this prompt and the index will be built automatically. This patch was released October 18, 2000.

Step 6. From programmer mode, set up the input variables and perform an IX^DIC call using the "D" index. Afterward, enter two question marks (??) at the "Select *ZZINDIVIDUAL* SSN:" prompt.

```
>S DIC="^DIZ(662nnn,",DIC(0)="QEAZ",D="D"
>D IX^DIC

Select ZZINDIVIDUAL SSN: ??

 Choose from:
 000221111      FMPATIENT,ONE
 666223333      FMPATIENT,FOUR
 666345678      FMPATIENT,NINE
 666432123      FMPATIENT,TEN
 666443333      FMPATIENT,TWO
 666567890      FMPATIENT,THIRTEEN
 666654321      FMPATIENT,FIVE
 666678901      FMPATIENT,SEVEN
 666765432      FMPATIENT,ELEVEN
 666776666      FMPATIENT,THREE
 666889999      FMPATIENT,SIX
 666891234      FMPATIENT,EIGHT
 666996666      FMPATIENT,TWELVE
```

This time, in contrast to Exercise 4.1, the entry with **SSN** 000221111 is displayed first.

Step 7. At the "Select *ZZINDIVIDUAL*:" prompt, enter 666-22-3333:

```
Select ZZINDIVIDUAL: 666-22-3333 <Enter> 666223333   FMPATIENT,FOUR
```

In contrast to Exercise 4.1, FileMan was able to find the entry you were looking for, even though you entered the SSN with hyphens, a form not found in the index.

End of Exercise 4.2.

Lesson 4. Quiz

This is the quiz for Lesson 4 of the VA FileMan V. 22.0 Key and Index Tutorial. Test yourself on what you've learned in Lesson 4. The correct answers can be found at the bottom of the page, or by selecting the link at each question.

Question 1: What are the three types of transforms you can perform in FileMan on cross-reference values used as subscripts in an index, 'Transform For ...':

    A. Storage, Lookup, and Display.
    B. Storage, Backup, and Display.
    C. Neither of the above.

Click here for answer

Question 2: You can use Transform for Storage to transform the internal value of:

    A. A field-type cross-reference value before it is stored as a subscript in the index.
    B. A subscript before it is stored as a field-type cross-reference in the index.
    C. Neither of the above.

Click here for answer

Question 3: Transform for Display takes the value, which has been transformed using the code in the Transform for Storage prior to storing the value in the index and:

    A. Converts it back into internal VA FileMan format for storage.
    B. Converts it back into a form that can be displayed to the user.
    C. Neither of the above.

Click here for answer

Question 4: FileMan uses Transform for Lookup during:

    A. A lookup.
    B. Display.
    C. Both of the above.

Click here for answer

Question 5: Each cross-reference value in a compound index can have its own Transform for Lookup.

    A. True.
    B. False.

Click here for answer

Question 6: If a user edits a field, FileMan executes the Transform for Storage and stores the transformed value in the field itself:

    A. True.
    B. False.

Click here for answer

Correct Answers – Lesson 4 Quiz

Question 1: What are the three types of transforms you can perform in FileMan on cross-reference values used as subscripts in an index, 'Transform For ...':

Answer: A. "Storage, Lookup, and Display."

Go Back

Question 2: You can use Transform for Storage to transform the internal value of:

Answer: A. "A field-type cross-reference value before it is stored as a subscript in the index."

Go Back

Question 3: Transform for Display takes the value, which has been transformed using the code in the Transform for Storage prior to storing the value in the index and:

Answer: B. "Converts it back into a form that can be displayed to the user."

Go Back

Question 4: FileMan uses Transform for Lookup during:

Answer: A. "A lookup."

Go Back

Question 5: Each cross-reference value in a compound index can have its own Transform for Lookup.

Answer: A. "True."

Go Back

Question 6: If a user edits a field, FileMan executes the Transform for Storage and stores the transformed value in the field itself:

Answer: B. "False."

Go Back

Lesson 5. Whole-File Indexes

In this lesson you will create a whole-file index based on the fields in the EMAIL multiple of the *ZZINDIVIDUAL* test file. This lesson will also illustrate a use for computed cross-reference values.

Exercise 5.1. Create a Whole-File Index Based on EMAIL NAME and EMAIL DOMAIN

In this exercise you will create a whole-file index that contains one computed subscript based on the EMAIL NAME and the EMAIL DOMAIN fields within the EMAIL multiple. The subscript in the index will look like:

   email_name@email_domain

Step 1. First, follow the menu path to the option Cross Reference a Field or File as shown below:

```
VA FileMan
   Utility Functions
      Cross Reference a Field or File
```

Step 2. Enter **N** for New at the prompt:

```
What type of cross-reference (Traditional or New)? Traditional//N
```

Step 3. Select the *ZZINDIVIDUAL* file and EMAIL subfile. Answer yes when prompted if you want to create a new Index.

```
MODIFY WHAT FILE: ZZINDIVIDUAL// <Enter>
Select Subfile: EMAIL <Enter> (Subfile #662nnn.02)

There are no INDEX file cross-references defined on subfile
#662nnn.02. Want to create a new Index for this file? No// Y <Enter>
YES
```

> ⓘ The fields you will select as cross-reference values in our index will be fields in the EMAIL subfile.

Step 4. For type of index, enter: **REGULAR**.

```
Type of index: REGULAR// <Enter> REGULAR
```

Step 5.   Because you selected a subfile (the EMAIL multiple), VA FileMan asks you whether you want to index the whole ZINDIVIDUAL file. Press Enter to select the default YES.

```
Want to index whole file ZZINDIVIDUAL (#662nnn)? Yes// <Enter> YES
```

If you answered NO here, the index would be stored at the subfile level. This would allow you to lookup an entry in the EMAIL subfile for a specific record in the *ZZINDIVIDUAL* file.

Step 6.   When asked how the index should be used, select SORTING ONLY:

```
Want index to be used for Lookup & Sorting
  or Sorting Only: LOOKUP & SORTING// SORTING ONLY
```

You want VA FileMan to use this index for lookup only if you explicitly specify the index in the call. If you choose LOOKUP & SORTING here, VA FileMan would automatically use this index whenever you passed the "M" flag to a lookup API.

Step 7.   For Index name select the default AC.

Sorting Only indexes must have names that start with the letter "A".

```
Index Name: AC// <Enter> AC
```

Step 8. You are now presented with the two-page ScreenMan form for editing the properties of your index. On the first page, enter the following as the Short Description:

> **This is a whole-file index with the computed subscript email**
> **name@email domain.**

```
Number: nnn                    EDIT AN INDEX              Page 1 of 2
─────────────────────────────────────────────────────────────────────

               File: 662nnn              Root File: 662nnn.02
        Index Name: AC                   Root Type Root File : WHOLE
FILE

   Short Description: This is a whole-file index with the computed
                      subscript email name@email domain.
   Description (wp):  (empty)

               Type: REGULAR

           Activity: IR
          Execution: FIELD

                Use: SORTING ONLY

─────────────────────────────────────────────────────────────────────


 COMMAND:                            Press <>PF1>H for help      Insert
```

> ℹ️ We're defining a whole-file index (Root type=WHOLE FILE), that contains fields at a
> subfile level (Root file=*662nnn*.02), but that resides at the top level (File = *662nnn*),
> the level at which you can use the index to lookup entries.

Step 9. Press <PF1><Down> or <PageDown> to go to page 2 of the ScreenMan form.

Step 10. Create two cross-reference values of type FIELD. Define the first as the EMAIL NAME field (#.01), and the second as the EMAIL DOMAIN field (#1) as shown below:

```
┌─────────────────────────────────────────────────────────────────────────┐
│  Number: nnn                   EDIT AN INDEX                 Page 2 of 2  │
│ ───────────────────────────────────────────────────────────────────────  │
│                                                                           │
│   CROSS-REFERENCE VALUES:                                                 │
│   Order...  Subscr  Type     Length  Field or Computed Expression         │
│   --------  ------  ----     ------  -----------------------------        │
│   1         1       FIELD     30     EMAIL NAME (#.01)                     │
│   2         2       FIELD     30     EMAIL DOMAIN (#1)                     │
│                                                                           │
│                                                                           │
│    Set Logic: S ^DIZ(662nnn,"AC",$E(X(1),1,30),$E(X(2),1,30),DA(1),DA)="" │
│   Kill Logic: K ^DIZ(662nnn,"AC",$E(X(1),1,30),$E(X(2),1,30),DA(1),DA)    │
│   Whole Kill: K ^DIZ(662nnn,"AC")                                         │
│                                                                           │
│    Set Condition:                                                         │
│   Kill Condition:                                                         │
│ ───────────────────────────────────────────────────────────────────────  │
│                                                                           │
│                                                                           │
│   COMMAND:                              Press <PF1>H for help  Insert      │
└─────────────────────────────────────────────────────────────────────────┘
```

> These two fields (EMAIL NAME and EMAIL DOMAIN) will not be used as subscripts in our index. Instead, you will create a single subscript based on those two fields.

Step 11. Tab to the "Subscr" column in the ScreenMan form, page 2, and delete the values 1 and 2.

> Since you are creating a Regular index, VA FileMan assumes that all fields you add as cross-reference values in the index will be used as subscripts and automatically assigns subscript numbers to them. Most of the time, this assumption is correct. However, if a cross-reference value is not to be used as a subscript in the index, as in this example, you must remember to delete the automatically generated subscript number.
>
> As you will see in Lesson 6, when you create a MUMPS cross-reference, FileMan assumes that cross-reference values will not be used as subscripts in an index.

Step 12. Tab to the next column labeled "Order…" on the third row of the table, type in the number 3 and press Enter.

Step 13. At the "CROSS-REFERENCE VALUES TYPE OF VALUE:" prompt enter "C" for COMPUTED.

```
CROSS-REFERENCE VALUES TYPE OF VALUE: C <Enter>
```

Step 14. Enter 1 as the subscript number.

Step 15. Tab to "Computed Code," enter:

```
S X=$$UP^XLFSTR(X(1)_"@"_X(2))
```

Step 16. Press <PF1>C to close the pop-up window.

Step 17. Delete all the values in the "Length" column. Both EMAIL NAME and EMAIL DOMAIN are free text fields from 1 to 20 characters in length. Our computed subscript, EMAIL NAME@EMAIL DOMAIN should therefore never exceed 41 characters (1 to 20 for each field combined and 1 for the "@" sign) in length, well within the limits of the M portability standards.

Here is how page 2 of the ScreenMan form should look:

```
 Number: nnn                    EDIT AN INDEX              Page 2 of 2
 _____

  CROSS-REFERENCE VALUES:
  Order...  Subscr  Type      Length  Field or Computed Expression
  --------  ------  ----      ------  ------------------------------
  1                 FIELD             EMAIL NAME (#.01)
  2                 FIELD             EMAIL DOMAIN (#1)
  3          1      COMPUTED          S X=$$UP^XLFSTR(X(1)_"@"_X(2))


   Set Logic: S ^DIZ(662nnn,"AC",X(3),DA(1),DA)=""
  Kill Logic: K ^DIZ(662nnn,"AC",X(3),DA(1),DA)
  Whole Kill: K ^DIZ(662nnn,"AC")

   Set Condition:
  Kill Condition:
 _____


  COMMAND:                            Press <PF1>H for help   Insert
```

Step 18. Press <PF1>E to save the changes and exit the form.

Step 19. If asked whether you to build the index now, answer Yes:

```
Do you want to build the index now? YES// <Enter>
```

Step 20. From programmer mode, do a %G listing to look at the "AC" index that was built:

```
>D ^%G <Enter>

Global ^DIZ(662nnn,"AC" <Enter>
        DIZ(662nnn,"AC"
^DIZ(662nnn,"AC","BASIL@ABC.DEF.COM",12,1) =
^DIZ(662nnn,"AC","BSNOW@XXX.YYY.COM",12,2) =
^DIZ(662nnn,"AC","CHERVIL@XXX.YYY.COM",11,1) =
^DIZ(662nnn,"AC","CPUDDLES@ABC.DEF.COM",11,2) =
^DIZ(662nnn,"AC","DILL@ABC.DEF.COM",5,1) =
^DIZ(662nnn,"AC","DTIDE@XXX.YYY.COM",5,2) =
^DIZ(662nnn,"AC","GINGER@AAA.BBB.COM",9,1) =
^DIZ(662nnn,"AC","HAZEL.FROST@ABC.DEF.COM",7,1) =
^DIZ(662nnn,"AC","HAZEL@ABC.DEF.COM",7,2) =
^DIZ(662nnn,"AC","HERB@ABC.DEF.COM",3,1) =
^DIZ(662nnn,"AC","HOLLY@ABC.DEF.COM",13,1) =
^DIZ(662nnn,"AC","HOLLYRIVERS@XXX.YYY.COM",13,2) =
^DIZ(662nnn,"AC","HWATERS@ABC.DEF.COM",3,2) =
^DIZ(662nnn,"AC","JASMINE.GEISER@AAA.BBB.COM",2,1) =
^DIZ(662nnn,"AC","JASMINE@XXX.YYY.COM",2,2) =
^DIZ(662nnn,"AC","MARIGOLD@AAA.BBB.COM",4,2) =
^DIZ(662nnn,"AC","MLAKE@XXX.YYY.COM",4,1) =
^DIZ(662nnn,"AC","PERIWINKLE@ABC.DEF.COM",10,1) =
^DIZ(662nnn,"AC","PWELLS@XXX.YYY.COM",10,2) =
^DIZ(662nnn,"AC","RC@AAA.BBB.COM",1,1) =
^DIZ(662nnn,"AC","ROSE@XXX.YYY.COM",1,2) =
^DIZ(662nnn,"AC","SAFFRON@AAA.BBB.COM",8,1) =
^DIZ(662nnn,"AC","SAGEBROOKS@AAA.BBB.COM",6,1) =
^DIZ(662nnn,"AC","SRIPPLE@ABC.DEF.COM",8,2) =
Global ^
```

End of Exercise 5.1.

Lesson 5. Quiz

This is the quiz for Lesson 5 of the VA FileMan V. 22.0 Key and Index Tutorial. Test yourself on what you've learned in Lesson 5. The correct answers can be found at the bottom of the page, or by selecting the link at each question.

Question 1: Whole-file indexes allow you to lookup records in a file or subfile based on field values within a subfile beneath that file or subfile. Where is a whole-file index stored?

    A. At the file level in which the fields are defined.
    B. At a file level above the level in which the fields are defined.
    C. Neither of the above.

Click here for answer

Question 2: When using the SORTING ONLY designation for an index, the index name must start with a:

    A. 'B' or a letter that alphabetically follows 'B'.
    B.   Sorting Only indexes must have names that start with the letter 'A'.
    C. Neither of the above.

Click here for answer

Question 3: In a Regular index, FileMan assumes that all fields you add as cross-reference values in the index will be used as subscripts and automatically assigns subscript numbers to them. Most of the time, this assumption However, if a cross-reference value is not to be used as a subscript in the index, you must remember to:

    A. Delete the automatically generated subscript number.
    B. Use FileMan to turn off automatic generation of subscript numbers.
    C. Neither of the above.

Click here for answer

Question 4: To create a single subscript for an index based on two fields, first, create two cross-reference values (order numbers 1 and 2), one for each of the two fields. Second, delete the automatically generated subscript numbers. Third, create a computed cross-reference value (order number 3) designated as subscript 1.

    A. True.
    B. False.

Click here for answer

Question 5: New with VA FileMan Version 22.0, the advanced capabilities of Regular cross-references, which include the ability to define compound indexes, computed subscripts, and subscript transforms allow you to define as New-style Regular indexes. (Those indexes you previously had to define as MUMPS cross-references.)

A. True.
B. False.

[Click here for answer](#)

Correct Answers – Lesson 5 Quiz

Question 1: Whole-file indexes allow you to lookup records in a file or subfile based on field values within a subfile beneath that file or subfile. Where is a whole-file index stored?

Answer: B. "At a file level above the level in which the fields are defined."

Go Back

Question 2: When using the SORTING ONLY designation for an index, the index name must start with a:

Answer: B. "Sorting Only indexes must have names that start with the letter 'A'."

Go Back

Question 3: In a Regular index, FileMan assumes that all fields you add as cross-reference values in the index will be used as subscripts and automatically assigns subscript numbers to them. Most of the time, this assumption However, if a cross-reference value is not to be used as a subscript in the index, you must remember to:

Answer: A. "Delete the automatically generated subscript number."

Go Back

Question 4: To create a single subscript for an index based on two fields, first, create two cross-reference values (order numbers 1 and 2), one for each of the two fields. Second, delete the automatically generated subscript numbers. Third, create a computed cross-reference value (order number 3) designated as subscript 1.

Answer: A. "True."

Go Back

Question 5: New with VA FileMan Version 22.0, the advanced capabilities of Regular cross-references, which include the ability to define compound indexes, computed subscripts, and subscript transforms allow you to define as New-style Regular indexes. (Those indexes you previously had to define as MUMPS cross-references.)

Answer: A. "True."

Go Back

Part 2. <u>MUMPS Cross-References</u>

VA FileMan Version 22.0 allows you to create two types of New-style cross-references: Regular and MUMPS. As you have seen in previous lessons, Regular cross-references build indexes that can be used to lookup and/or sort entries in a file.

The advanced capabilities of Regular cross-references, which include the ability to define compound indexes, computed subscripts, and subscript transforms, means that most of the indexes you previously had to define as Traditional MUMPS cross-references can, with Version 22.0, be defined as New-style Regular indexes. You can usually reserve MUMPS cross-references for performing actions other than building indexes.

Lesson 6. Create a New-Style MUMPS Cross-Reference

This lesson introduces you to New-style MUMPS cross-references. It covers two topics:

1. The X, X1, and X2 arrays.
2. The set and kill logic and conditions of cross-references.

The X, X1, and X2 Arrays

The <u>set and kill logic</u> and the set and Kill Conditions of New-style cross-references can reference the X, X1, and X2 arrays. Here is what those arrays contain when a record is edited, and the cross-reference is fired:

X(order#)	Contains the **old** cross reference values when the kill logic and Kill Condition are executed. Contains the **new** cross reference values when the Set Logic and Set Condition are executed.
X1(order#)	Contains the **old** cross reference values when the set and kill logic and conditions are executed.
X2(order#)	Contains the **new** cross reference values when the set and kill logic and conditions are executed.

"Order#" is the order number of the cross-reference value. (Recall that new-style cross-reference can have more than one cross-reference value.) The order number is displayed in the "Order..." column on page 2 of the ScreenMan form when a cross-reference is edited or created.

When a new record is added to a file, VA FileMan sets the .01 field and immediately fires the Set Logic of all the cross-references on the .01 field. If a New-style cross-reference contains the .01 field as a cross-reference value, the X1(order#) array element that corresponds to the .01 field is guaranteed to be null at this point.

When a record is deleted from a file, the kill logic of all cross-references for that record is executed. All X2(order#) array elements are null at this point.

Set and Kill Condition

You can define Set and Kill Conditions on both Regular and MUMPS cross-references to prevent a cross-reference from firing under conditions you specify.

If there is a Set Condition, the Set Logic will be executed only if the Set Condition sets X to Boolean true (that is, after the Set Condition code is executed, the command IF X evaluates to true). Similarly, if there is Kill Condition, the Kill Logic is executed only if the Kill Condition sets X to Boolean true.

Typically, your Set and Kill Conditions would check the X, X1 and/or X2 arrays to decide whether the Set and Kill Logic should be executed.

Exercise 6.1. Create Your First New-Style MUMPS Cross-Reference

In this exercise, you will create a MUMPS cross-reference that should be executed only when a new record is added to the file. To do this you will add the .01 field as the only cross-reference value in the cross-reference, make it order number 1, and include a Set Condition that sets X to 1 only if X1(1) is null. The Set Logic itself will update the DATE CREATED field with the current date and time.

Step 1.   Use VA FileMan's Cross Reference a Field or File option to create a MUMPS cross-reference named "AD".

```
Select OPTION: UTILITY FUNCTIONS
Select UTILITY OPTION: CROSS-REFERENCE A FIELD OR FILE

What type of cross-reference (Traditional or New)? Traditional// NEW

MODIFY WHAT FILE: ZZINDIVIDUAL// <Enter>
Select Subfile: <Enter>

Current Indexes on file #662nnn:
  nnn    'C' index
  nnn    'D' index

Choose E (Edit)/D (Delete)/C (Create): CREATE

Want to create a new Index for this file? No// YES

Type of index: REGULAR// MUMPS

How is this MUMPS cross reference to be used: ACTION// <Enter>
```

> **i** Recall that our other choices at this prompt are LOOKUP & SORTING and SORTING ONLY. Instead of setting index for lookup and/or sorting, our MUMPS cross-reference will perform an action, namely, set the DATE CREATED field (#4.1) to the current date and time.

```
Index Name: AD// <Enter>
```

> ACTION cross-references, just like SORTING ONLY indexes, must have names that begin with the letter "A".

Step 2. Use the ScreenMan form to select the .01 field as a cross-reference value for the MUMPS cross-reference.

1. On page 1 of the ScreenMan form, at the "Short Description:" field enter **Set the DATE CREATED field when a new record is added**.
2. On page 2, in the "Order..." column type **1** and press Enter.
3. For "Cross-Reference Values Type of Value", enter **FIELD**.
4. In the pop-up window at the "Field:" prompt, enter **.01**.
5. Press <PF1>C to close the pop-up page.

> ⓘ Because you are defining a MUMPS cross-reference, FileMan assumes your cross-reference value will not be used as a subscript, and so doesn't automatically assign it a subscript number. Since most of the time, cross-references that set an index can be defined as Regular indexes, most MUMPS cross-references don't have values that are used as subscripts in an index. If your MUMPS cross-reference has cross-reference values that are used as subscripts, you must manually enter subscript numbers for them.

Step 3. At the "Set Logic:" prompt enter (on a single line):

```
N ZZFDA,ZZMSG,DIERR S ZZFDA(662nnn,DA_",",4.1)="NOW"
D FILE^DIE("E","ZZFDA","ZZMSG")
```

> ⓘ This code sets a node in the ZZFDA array, which is passed to the Filer (FILE^DIE). The Filer sets the current date and time (NOW) into the DATE CREATED field (#4.1) of the *ZZINDIVIDUAL* file. To simplify the example, you will ignore any error messages the Filer returns in the ZZMSG array.

> ⓘ **TIPS: <CTRL-U>** is a toggle between the values: null, last edited, and the default of the ScreenMan field you are editing.
>
> Press **<PF1>Z** ("zoom") at any field to open up an editing window at the bottom of the screen. This editing window is often easier to use than the usual scrolling window, especially when the value of the field is very long, as is the case with the Set Logic.

Step 4. At the "Set Condition:" prompt enter:

```
S X=X1(1)=""
```

> **i** This Set Condition sets X to 1 if X1(1)=""; otherwise, it sets it to 0. Since X1(1) refers
> to the old value of the .01 field (the cross-reference value with order number 1), the Set
> Condition means that the Set Logic of our MUMPS cross-reference should be executed
> only if the record is new.

```
 Number: nnn                    EDIT AN INDEX              Page 2 of 2
 _____

 CROSS-REFERENCE VALUES:
 Order...   Subscr  Type     Length  Field or Computed Expression
 --------   ------  ----     ------  ----------------------------
 1                  FIELD    30      NAME (#.01)


  Set Logic: N ZZFDA,ZZMSG,DIERR S ZZFDA(662nnn,DA_",",4.1)="NOW"
             D FILE^DIE("E"
 Kill Logic: Q
 Whole Kill:

  Set Condition: S X=X1(1)=""
 Kill Condition:

 _____

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

 COMMAND:                              Press <PF1>H for help   Insert
```

Step 5. Press <PF1>E to exit the form.

Step 6. At the "Do you want to cross reference existing data now?" prompt, answer NO.

```
Do you want to cross reference existing data now? NO
```

> **i** If you answered "YES" at this point, VA FileMan will loop through all the entries in
> the file, and execute the Set Condition. If the Set Condition sets X to true (1), VA
> FileMan would execute the Set LogiC. We want FileMan to update the DATE
> CREATED field only for new records subsequently added to the file, not for entries
> already in the file. Also, X1(1) would evaluate to true (1) for every existing entry,
> since all entries have non-null .01 field values, so the Set Logic wouldn't get executed
> anyway.

End of Exercise 6.1.

Exercise 6.2. Test the New MUMPS Cross-Reference

In this exercise, you will check the behavior of the new MUMPS cross-reference you created in Exercise
6.1. To do this, you will add a record to the file and make sure the DATE CREATED field is correctly
updated with the current date and time. You will then edit the .01 field of an existing record to make sure
the DATE CREATED field is **not** updated.

Step 1. Use VA FileMan's Enter or Edit File Entries option to add a new record 'NEW,ENTRY' to the
ZZINDIVIDUAL file:

```
Select OPTION NAME: ENTER OR EDIT FILE ENTRIES

INPUT TO WHAT FILE: ZZINDIVIDUAL// <Enter>
EDIT WHICH FIELD: ALL// .01  NAME
THEN EDIT FIELD: <Enter>

Select ZZINDIVIDUAL NAME: NEW,ENTRY
  Are you adding 'NEW,ENTRY' as a new ZZINDIVIDUAL (the 14TH)? No//
Y  (Yes)
```

Step 2. Use VA FileMan's Inquire to File Entries to look at the 'NEW,ENTRY' record you just added:

```
Select OPTION NAME: INQUIRE TO FILE ENTRIES

OUTPUT FROM WHAT FILE: ZZINDIVIDUAL// <Enter>
Select ZZINDIVIDUAL NAME: <space><Enter>   NEW,ENTRY
ANOTHER ONE: <Enter>
STANDARD CAPTIONED OUTPUT? Yes// <Enter>  (Yes)
Include COMPUTED fields:  (N/Y/R/B): NO// <Enter> - No record number
(IEN), no
Computed Fields

NAME: NEW,ENTRY                    DATE CREATED: FEB 21,2001@10:42:27
```

> **i** The MUMPS cross-reference stuffed the current date and time into the DATE
> CREATED field.

Step 3. Use VA FileMan's Enter or Edit File Entries option to edit the NAME field of our
'NEW,ENTRY' record to 'MODIFIED,ENTRY':

```
Select OPTION NAME: ENTER OR EDIT FILE ENTRIES

INPUT TO WHAT FILE: ZZINDIVIDUAL// <Enter>
EDIT WHICH FIELD: ALL// NAME
THEN EDIT FIELD: <Enter>

Select ZZINDIVIDUAL NAME:  <space><Enter>  NEW,ENTRY
NAME: NEW,ENTRY// MODIFIED,ENTRY
```

Step 4. Use VA FileMan's Inquire to File Entries again to see if the DATE CREATED field has changed:

```
Select OPTION NAME: INQUIRE TO FILE ENTRIES

OUTPUT FROM WHAT FILE: ZZINDIVIDUAL// <Enter>
Select ZZINDIVIDUAL NAME:  <space><Enter>  MODIFIED,ENTRY
ANOTHER ONE: <Enter>
STANDARD CAPTIONED OUTPUT? Yes//  <Enter> (Yes)
Include COMPUTED fields:  (N/Y/R/B): NO// <Enter> - No record number
(IEN), no
Computed Fields

 NAME: MODIFIED,ENTRY              DATE CREATED: FEB 21,2001@10:42:27
```

> ℹ The date and time stored in the DATE CREATED field has not changed, even though the value of the .01 field was changed. Our Set Condition prevented the Set Logic from executing since during the edit, the values of the .01 field and the corresponding X1(1) array element are not null.

End of Exercise 6.2.

Congratulations! You have just created your first New-Style MUMPS cross-reference!

Lesson 6. Quiz

This is the quiz for Lesson 6 of the VA FileMan V. 22.0 Key and Index Tutorial. Test yourself on what you've learned in Lesson 6. The correct answers can be found at the bottom of the page, or by selecting the link at each question.

Question 1: The logic of a New-style MUMPS cross-reference, which is defined by the programmer, is executed whenever a field in the cross-reference is edited, and usually performs some action other than maintaining an index.

   A. True.
   B. False.

Click here for answer

Question 2: The Set and Kill logic and the Set and Kill conditions of New-style cross-references can reference the X, X1, and X2 arrays. What does the X(order#) array contain when a record is edited, and the cross-reference is fired?

   A. The old cross-reference values when the Kill logic and Kill condition are executed.
   B. The new cross-reference values when the Set logic and Set condition are executed.
   C. Both of the above.

Click here for answer

Question 3: What does the X1(order#) array contain when a record is edited, and the cross-reference is fired?

   A. The old cross-reference values when the Set and Kill logic and conditions are executed.
   B. The new cross-reference values when the Set and Kill logic and conditions are executed.
   C. Neither of the above.

Click here for answer

Question 4: What does the X2(order#) array contain when a record is edited, and the cross-reference is fired?

   A. The old cross-reference values when the Set and Kill logic and conditions are executed.
   B. The new cross-reference values when the Set and Kill logic and conditions are executed.
   C. Neither of the above.

Click here for answer

Question 5: When a new record is added to a file, FileMan sets the .01 field and immediately fires the Set logic of all the cross-references on the .01 field. The X1(order#) array element that corresponds to the .01 field is null at this point.

    A. True.
    B. False.

Click here for answer

Question 6: When a record is deleted from a file, the Kill logic of all cross-references for that record is executed. All X2(order#) array elements are null at this point.

    A. True.
    B. False.

Click here for answer

Question 7: You can define Set and Kill Conditions on both Regular and MUMPS cross-references to prevent a cross-reference from firing under conditions you specify. If there is a Set Condition, the Set Logic will be executed only if the Set Condition sets X to Boolean true (i.e., after the Set Condition code is executed, the command IF X evaluates to true).

    A. True.
    B. False.

Click here for answer

Question 8: If there is Kill Condition on a Regular and MUMPS cross-reference, the Kill Logic is executed only if the Kill Condition sets X to Boolean false.

    A. True.
    B. False.

Click here for answer

Question 9: An Action-type cross-reference is a cross-reference with Set and Kill logic that performs some action other than building an index. Most MUMPS cross-references are Action-type cross-references. An Action-type cross-reference must have a name that starts with the letter:

    A. "A"
    B. "C"
    C. Neither of the above.

Click here for answer

Correct Answers – Lesson 6 Quiz

Question 1: The logic of a New-style MUMPS cross-reference, which is defined by the programmer, is executed whenever a field in the cross-reference is edited, and usually performs some action other than maintaining an index.

    Answer: A. "True."

Go Back

Question 2: The Set and Kill logic and the Set and Kill conditions of New-style cross-references can reference the X, X1, and X2 arrays. What does the X(order#) array contain when a record is edited, and the cross-reference is fired?

    Answer: C. "Both of the above." (The X(order#) array contains [1] the old cross-reference values when the Kill Logic and Kill Condition are executed, and [2] the new cross-reference values when the Set Logic and Set Condition are executed.)

Go Back

Question 3: What does the X1(order#) array contain when a record is edited, and the cross-reference is fired?

    Answer: A. "The old cross-reference values when the Set and Kill logic and conditions are executed."

Go Back

Question 4: What does the X2(order#) array contain when a record is edited, and the cross-reference is fired?

    Answer: B. "The new cross-reference values when the Set and Kill logic and conditions are executed."

Go Back

Question 5: When a new record is added to a file, FileMan sets the .01 field and immediately fires the Set logic of all the cross-references on the .01 field. The X1(order#) array element that corresponds to the .01 field is null at this point.

    Answer: A. "True."

Go Back

Question 6: When a record is deleted from a file, the Kill logic of all cross-references for that record is executed. All X2(order#) array elements are null at this point.

Answer: A. "True."

Question 7: You can define Set and Kill Conditions on both Regular and MUMPS cross-references to prevent a cross-reference from firing under conditions you specify. If there is a Set Condition, the Set Logic will be executed only if the Set Condition sets X to Boolean true (i.e., after the Set Condition code is executed, the command IF X evaluates to true).

Answer: A. "True."

Question 8: If there is Kill Condition on a Regular and MUMPS cross-reference, the Kill Logic is executed only if the Kill Condition sets X to Boolean false.

Answer: B. "False." (The Kill Logic is executed only if the Kill Condition sets X to Boolean true.)

Question 9: An Action-type cross-reference is a cross-reference with Set and Kill logic that performs some action other than building an index. Most MUMPS cross-references are Action-type cross-references. An Action-type cross-reference must have a name that starts with the letter:

Answer: A. "A".

Lesson 7. Cross-Reference Execution

In this lesson you will learn when VA FileMan executes the Set and Kill Logic of New-style cross-references. You will also use the X1 and X2 arrays in the Set Logic of a MUMPS cross-reference.

When Does Cross-Reference Logic Get Executed?

In general, for New-style cross-references both Regular and MUMPS, when a field or group of fields defined in a New-style cross-reference is edited, both the Kill and the Set Logic of that cross-reference are executed. First, the Kill logic is executed with the X(order#) array set to the old (pre-edited) field values. Then the Set Logic is executed with the X(order#) array set to the new values.

The exceptions to this are as follows:

1. **Null subscript values.**

    If a cross-reference value is used as a subscript (that is, it has a subscript number), and the old value is null, the Kill logic is not executed. Similarly, the Set Logic is not executed if a new value used as a subscript is null.

2. **Record deletion.**

    When an entire record is deleted from a file, only the Kill Logic, not the Set Logic, is executed. The X2(order#) array elements, which normally contain the new cross-reference values, are all set to null.

3. **Record creation.**

    Immediately after a record is added to a file, only the Set Logic, not the Kill Logic, of New-style cross-references that contain the .01 field is executed. The X1(order#) array element that corresponds to the .01 field is set to null at this point.

4. **Kill and Set Conditions.**

    As illustrated in Lesson 6, if the Set or Kill Condition sets X to Boolean false, the Set or Kill Logic will not be executed.

The following table summarizes when the Kill and Set Logic are executed for New-style cross-references.

New-style cross-references

	Kill Logic	Set Logic
Record creation	Not executed	**Executed** immediately after the .01 is set (if .01 is a cross-reference value.)
Record edit	**Executed**[1]	**Executed**[2]
Record deletion	**Executed**[1]	Not executed

[1] only if no old values used as subscripts are null, and the Set Condition, if any, sets X to true (1).
[2] only if no new values used as subscripts are null, and the Kill Condition, if any, sets X to true (1).

Note that in "Record edit," it doesn't matter if individual field values are created (go from null to non-null) or deleted (go from non-null to null) - both the Kill and the Set Logic are executed as long as none of the field values used as subscripts evaluate to null.

This behavior is slightly different than that of Traditional cross-references. Since Traditional cross-references are defined on a specific field, the Kill and Set Logic are executed depending on whether that one field value is created, deleted, or edited.

The following table summarizes when the Kill and Set Logic are executed for Traditional cross-references.

Traditional cross-references

	Kill Logic	**Set Logic**
Field value "creation" - value goes from null to non-null.	Not executed	**Executed**
Field edit - both old and new values are non-null.	**Executed**	**Executed**
Field value deletion - value goes from non-null to null.	**Executed**	Not executed
Record creation	Not executed	**Executed** for all x-refs on the .01 field, immediately after the .01 is set.
Record deletion	**Executed**	Not executed

Execution

The Execution property of New-style cross-references controls the timing of cross-reference execution. Execution can be either of the following:

1. **Field** execution means that the cross-reference logic is executed immediately after each and every field in the cross-reference is edited.

2. **Record** execution means that the cross-reference logic is executed once at the end of an editing session after all fields in the record are edited.

Usually, simple cross-references (those that have only one field-type cross-reference value) have Field-level execution. Compound cross-references (those that have more than one field-type cross-reference value) have Record-level execution.

Exercise 7.1. Create a MUMPS Cross-Reference that Makes Use of the X, X1, and X2 Arrays

In this exercise, you will create a MUMPS cross-reference that contains two field-type cross-reference values, AREA CODE field (#3.1) and LOCAL NUMBER field (#3.2). When either or both of those fields are updated, the OLD PHONE NUMBER field (#3.4) will be set to:

   (Old_Area_Code) Old_Local_Number

and the PHONE NUMBER field (#3.3) is set to:

   (New_Area_Code) New_Local_Number

In addition, if the old area code, new area code, old local number, or new local number is null, you will store the value as the string "null" in the OLD PHONE NUMBER and PHONE NUMBER fields. To accomplish this task, you will use computed cross-reference values.

Step 1.   Use VA FileMan's Cross Reference a Field or File option to create a MUMPS cross-reference named "AE":
```
Select OPTION: UTILITY FUNCTIONS
Select UTILITY OPTION: CROSS-REFERENCE A FIELD OR FILE

What type of cross-reference (Traditional or New)? Traditional// NEW

MODIFY WHAT FILE: ZZINDIVIDUAL// <Enter>
Select Subfile: <Enter>

Current Indexes on file #662nnn:
  nnn    'C' index
  nnn    'D' index
  nnn    'AD' index

Choose E (Edit)/D (Delete)/C (Create): CREATE
Want to create a new Index for this file? No// YES

Type of index: REGULAR// MUMPS

How is this MUMPS cross reference to be used: ACTION// ACTION

Index Name: AE// <Enter> AE
```

Step 2.   On the first page of the ScreenMan form, enter the following Short Description:

> **Update PHONE NUMBER & OLD PHONE NUMBER if AREA CODE & LOCAL NUMBER are edited.**

Step 3.   Tab to Execution and enter **R** for RECORD.

```
 Number: nnn                 EDIT AN INDEX               Page 1 of 2
─────────────────────────────────────────────────────────────────────

              File: 662nnn              Root File: 662nnn
        Index Name: AE                  Root Type: INDEX FILE

 Short Description: Update PHONE NUMBER & OLD PHONE NUMBER if
                    AREA CODE & LOCAL NUMBER are edited.
 Description (wp):  (empty)

              Type: MUMPS

          Activity: IR
         Execution: RECORD

               Use: SORTING ONLY

─────────────────────────────────────────────────────────────────────


  COMMAND:                       Press <>PF1>H for help    Insert
```

> Our "AE" cross-reference will contain two fields, AREA CODE and LOCAL
> NUMBER, and so should have an execution value of Record.

Step 4.   On page 2 of the ScreenMan form, add the AREA CODE field as order number 1:

```
Order...: 1
CROSS-REFERENCE VALUES TYPE OF VALUE: FIELD
Field: AREA CODE
```

Step 5.   Add the LOCAL NUMBER field as order number 2:

```
Order...: 2
CROSS-REFERENCE VALUES TYPE OF VALUE: FIELD
Field: LOCAL NUMBER
```

Page 2 of the ScreenMan form should now look like this:

```
 Number: nn                  EDIT AN INDEX               Page 2 of 2
 ───────────────────────────────────────────────────────────────────

  CROSS-REFERENCE VALUES:
  Order...   Subscr  Type     Length  Field or Computed Expression
  --------   ------  ----     ------  ----------------------------
   1                 FIELD            AREA CODE (#3.1)
   2                 FIELD     30     LOCAL NUMBER (#3.2)


   Set Logic: Q
  Kill Logic: Q
  Whole Kill:

   Set Condition:
  Kill Condition:
 ───────────────────────────────────────────────────────────────────


  COMMAND:                         Press <>PF1>H for help     Insert
```

Step 6.   Add a cross-reference value of type Computed that evaluates either to the AREA CODE field
or to the string "null", if the AREA CODE field is missing.

```
Order...: 3
CROSS-REFERENCE VALUES TYPE OF VALUE: COMPUTED
Computed Code: S X=$S(X(1)="":"null",1:X(1))
```

ℹ   X(**1**) in the computed expression is the value of the AREA CODE field, the cross-
reference value with order number **1**. The computed expression sets the variable X to
the AREA CODE (X(1)), or if the AREA CODE is null (X(1)=""), it sets X to the
string "null".

Step 7.   Similarly, add a cross-reference value of type Computed that evaluates to the LOCAL
NUMBER field, or, if the LOCAL NUMBER field is missing, the string "null."

```
Order...: 4
CROSS-REFERENCE VALUES TYPE OF VALUE: COMPUTED
Computed Code: S X=$S(X(2)="":"null",1:X(2))
```

> **i**   X(**2**) in the computed expression is the cross-reference value with order number **2**, in
> this case, the LOCAL NUMBER field.

Page 2 of the ScreenMan form should now look like this:

```
 Number: nnn                     EDIT AN INDEX                Page 2 of 2
 ─────────────────────────────────────────────────────────────────────

 CROSS-REFERENCE VALUES:
 Order...  Subscr  Type      Length  Field or Computed Expression
 --------  ------  ----      ------  ---------------------------
 1                 FIELD             AREA CODE (#3.1)
 2                 FIELD             LOCAL NUMBER (#3.2)
 3                 COMPUTED          S X=$S(X(1)="":"null",1:X(1))
 4                 COMPUTED          S X=$S(X(2)="":"null",1:X(2))


  Set Logic: Q
 Kill Logic: Q
 Whole Kill:

  Set Condition:
 Kill Condition:
 ─────────────────────────────────────────────────────────────────────


  COMMAND:                        Press <>PF1>H for help      Insert
```

Step 8.   Tab to Set Logic and enter (on one line):

```
N ZZFDA,ZZMSG,DIERR
 S ZZFDA(662nnn,DA_",",3.3)="(" X2(3) ") " X2(4),
ZZFDA(662nnn,DA_",",3.4)="(" X1(3) ") " X1(4)
 D FILE^DIE("E","ZZFDA","ZZMSG")
```

> **i**   This code calls the Filer (FILE^DIE) to set the PHONE NUMBER field (#3.3) to the
> new (Area_Code) Local_Number, and the OLD PHONE NUMBER field (#3.4) to the
> old (Area_Code) Local_Number.

Step 9. Press <PF1>E to exit the form.

Step 10. At the "Do you want to cross reference existing data now?" prompt, answer NO, shown below:

```
Do you want to cross reference existing data now? NO
```

> **i** If you answered "YES" at this point, VA FileMan will execute the Set Logic for all entries in the file. For each record, the PHONE NUMBER field (#3.3) and the OLD PHONE NUMBER field (#3.4) would both be populated with (Area_Code) Local_Number. This is not a problem, but we'll choose to have those fields updated only upon subsequent edits.

End of Exercise 7.1.

Exercise 7.2. Test the New MUMPS Cross-Reference

Step 1. Use VA FileMan's Enter or Edit File Entries to edit the AREA CODE field (#3.1) and the LOCAL NUMBER field (#3.2) of the MODIFIED,ENTRY record added in Lesson 6.

```
Select OPTION NAME: ENTER OR EDIT FILE ENTRIES

INPUT TO WHAT FILE: ZZINDIVIDUAL// <Enter>
EDIT WHICH FIELD: ALL// 3.1 <Enter>   AREA CODE
THEN EDIT FIELD: 3.2 <Enter>   LOCAL NUMBER
THEN EDIT FIELD:

Select ZZINDIVIDUAL NAME: MODIFIED,ENTRY
AREA CODE: 415
LOCAL NUMBER: 555-1234
```

Step 2. Use VA FileMan's Inquire to File Entries option to look at the data in the MODIFIED,ENTRY record.

```
Select OPTION NAME: INQUIRE TO FILE ENTRIES

OUTPUT FROM WHAT FILE: ZZINDIVIDUAL// <Enter>
Select ZZINDIVIDUAL NAME: MODIFIED,ENTRY
ANOTHER ONE: <Enter>
STANDARD CAPTIONED OUTPUT? Yes// <Enter> (Yes)
Include COMPUTED fields:  (N/Y/R/B): NO// <Enter> - No record number
(IEN), no Computed Fields

NAME: MODIFIED,ENTRY              AREA CODE: 415
  LOCAL NUMBER: 555-1234          PHONE NUMBER: (415) 555-1234
    OLD PHONE NUMBER: (null) null  DATE CREATED: <Date/Time Created>
```

> ℹ️ The OLD PHONE NUMBER field is set to the previous phone number (area code and local phone were both null), and the PHONE NUMBER field is set to the new phone number (area code and local number).

Step 3. Repeat steps 1 and 2, but this time delete the area code. At the end of this step, the field values should be:

```
NAME: MODIFIED,ENTRY                LOCAL NUMBER: 555-1234
  PHONE NUMBER: (null) 555-1234    OLD PHONE NUMBER: (415) 555-1234
  DATE CREATED: <Date/Time Created>
```

> ℹ️ The PHONE NUMBER and OLD PHONE NUMBER fields again correctly reflect the original and new values of the AREA CODE and LOCAL NUMBER fields.

Step 4. Repeat Steps 1 and 2, but this time enter 510 as the Area Code and delete the Local Number. At the end of this step, the field values should be:

```
NAME: MODIFIED,ENTRY                AREA CODE: 510
  PHONE NUMBER: (510) null        OLD PHONE NUMBER: (null) 555-1234
  DATE CREATED: <Date/Time Created>
```

Step 5. Repeat Steps 1 and 2, but this time enter 555-3412 as the Local Number. At the end of this step, the field values should be:

```
NAME: MODIFIED,ENTRY                AREA CODE: 510
  LOCAL NUMBER: 555-3412          PHONE NUMBER: (510) 555-3412
  OLD PHONE NUMBER: (510) null    DATE CREATED: <Date/Time Created>
```

> ℹ️ In all cases, Steps 1 through 5, the Set Logic was executed, even when the Area Code and/or the Local Number field values were deleted.

End of Exercise 7.2.

Lesson 7. Quiz

This is the quiz for Lesson 7 of the VA FileMan V. 22.0 Key and Index Tutorial. Test yourself on what you've learned in Lesson 7. The correct answers can be found at the bottom of the page, or by selecting the link at each question.

Question 1: The Kill Logic for a New-style cross-reference is executed when a record is edited and:

      A. Old values used as subscripts are null, and the Kill Condition, if any, sets X to true (1).
      B. No old values used as subscripts are null, and the Kill Condition, if any, sets X to true (1).
      C. Neither of the above.

Click here for answer

Question 2: The Set Logic for a Traditional cross-reference on a field is executed if:

      A. The field value goes from null to non-null.
      B. The field value goes from non-null to null.
      C. Neither of the above.

Click here for answer

Question 3: If a field is part of a New-style MUMPS cross-reference, and the field value is deleted:

      A. The Kill Logic will not be executed under any circumstances.
      B. The Kill Logic will not be executed if the field is used as a subscript (has a subscript number assigned to it).
      C. Neither of the above.

Click here for answer

Question 4: The Execution property of New-style cross-references controls the timing of cross-reference execution. Execution can be either Field or Record. Field execution means that:

      A. The cross-reference logic is executed after all the fields in the cross-reference are edited.
      B. The cross-reference logic is executed immediately after each and every field in the cross-reference is edited.
      C. Neither of the above.

Click here for answer

Question 5: Record execution means that:

    A. The cross-reference logic is executed once at the end of an editing session after all fields in the record are edited.
    B. The cross-reference logic is executed immediately after each and every field in the cross-reference is edited.
    C. Neither of the above.

Question 6: Usually, simple cross-references (those that have only one field-type cross-reference value) have Field-level execution?

    A. True.
    B. False.

Question 7: Usually, compound cross-references (those that have more than one field-type cross-reference value) have Record-level execution:

    A. True.
    B. False.

Correct Answers – Lesson 7 Quiz

Question 1: The Kill Logic for a New-style cross-reference is executed when a record is edited and:

Answer: B. "No old values used as subscripts are null, and the Kill Condition, if any, sets X to true (1)."

Go Back

Question 2: The Set Logic for a Traditional cross-reference on a field is executed if:

Answer: A. "The field value goes from null to non-null."

Go Back

Question 3: If a field is part of a New-style MUMPS cross-reference, and the field value is deleted:

Answer: B. "The Kill Logic will not be executed if the field is used as a subscript (has a subscript number assigned to it)."

Go Back

Question 4: The Execution property of New-style cross-references controls the timing of cross-reference execution. Execution can be either Field or Record. Field execution means that:

Answer: B. "The cross-reference logic is executed immediately after each and every field in the cross-reference is edited."

Go Back

Question 5: Record execution means that:

Answer: A. "The cross-reference logic is executed once at the end of an editing session after all fields in the record are edited."

Go Back

Question 6: Usually, simple cross-references (those that have only one field-type cross-reference value) have Field-level execution?

Answer: A. "True."

Go Back

Question 7: Usually, compound cross-references (those that have more than one field-type cross-reference value) have Record-level execution:

Answer: A. "True."

[Go Back](#)

Lesson 8. Using ACTIVITY to Suppress Cross-Reference Execution

In this lesson you will learn about the Activity property of New-style cross-references.

Activity

Cross-references are executed when a record is re-indexed via one of the ^DIK entry points or when the Re-Index File option on the VA FileMan Utility Functions menu is selected. Also, during a Kernel Installation and Distribution System (KIDS) installation, if the KIDS Build sends a file with data, records in the file may be reindexed automatically.

The Activity cross-reference property allows you to prevent a particular cross-reference from firing during a re-indexing operation and/or a KIDS installation. By default, when you create a New-style cross-reference, Activity is automatically set to "IR". Activity does not control whether a cross-reference is fired when a record is added, edited, or deleted.

If you wish to suppress the firing of the cross-reference when VA FileMan's reindexing APIs (xxx^DIK) and when the Re-Index File option is called, you can remove the "R" from Activity.

Note, however, that if you explicitly select a cross-reference in an EN^DIK, EN1^DIK, or ENALL^DIK call, or in the UTILITY FUNCTIONS/RE-INDEX FILE option on the VA FileMan menu, that cross-reference will be fired whether or not its Activity contains an "R".

If you wish to suppress firing a cross-reference during a KIDS installation, you can remove the "I" from Activity.

Exercise 8.1. Reindex a Single Record with IX1^DIK

In this exercise, you will reindex the MODIFIED,ENTRY record via the IX1^DIK entry point. This causes the Set Logic of the "AE" cross-reference you created in Lesson 7 to be executed.

Step 1. Use VA FileMan's Enter or Edit File Entries option to edit the AREA CODE field (#3.1) and the LOCAL NUMBER field (#3.2) of the MODIFIED,ENTRY record. First change the area code to 111 and the local number to 111-1111. Then change them again to 415 and 555-1234, respectively. This step simply establishes a starting point for the values of the fields OLD PHONE NUMBER and PHONE NUMBER.

```
   Select OPTION NAME: ENTER OR EDIT FILE ENTRIES

   INPUT TO WHAT FILE: ZZINDIVIDUAL// <Enter>
   EDIT WHICH FIELD: ALL// 3.1 <Enter>  AREA CODE
   THEN EDIT FIELD: 3.2 <Enter>  LOCAL NUMBER
```

```
THEN EDIT FIELD:

Select ZZINDIVIDUAL NAME: MODIFIED,ENTRY
AREA CODE: 111
LOCAL NUMBER: 111-1111

Select ZZINDIVIDUAL NAME: <space><Enter> MODIFIED,ENTRY
AREA CODE: 111// 415
LOCAL NUMBER: 111-1111// 555-1234
```

Step 2. Use VA FileMan's Inquire to File Entries option to look at the values of OLD PHONE
NUMBER and PHONE NUMBER. Select a STANDARD CAPTIONED OUTPUT, and when
asked whether to include COMPUTED fields, answer R to include the Record Number in the
display.

```
Select OPTION NAME: INQUIRE TO FILE ENTRIES

OUTPUT FROM WHAT FILE: ZZINDIVIDUAL// <Enter>
Select ZZINDIVIDUAL NAME:  <space><Enter>  MODIFIED,ENTRY
ANOTHER ONE: <Enter>
STANDARD CAPTIONED OUTPUT? Yes// <Enter> (Yes)
Include COMPUTED fields:  (N/Y/R/B): NO// Record Number (IEN)
```

> This number may be different on your system if
> you've added other records to your test file.
>
> Use the number displayed on your system in Step 3.

```
NUMBER: 14                          NAME: MODIFIED,ENTRY
  AREA CODE: 415                      LOCAL NUMBER: 555-1234
  PHONE NUMBER: (415) 555-1234      OLD PHONE NUMBER: (111) 111-1111
  DATE CREATED: <Date/Time Created>
```

Step 3. Use IX1^DIK to reindex the MODIFIED,ENTRY. Set DA to the record number displayed in
Step 2.

```
>S DIK="^DIZ(662nnn,",DA=14
>D IX1^DIK
```

Step 4. Use VA FileMan's Inquire to File Entries option to see the effect of reindexing the entry.

```
Select OPTION NAME: INQUIRE TO FILE ENTRIES

OUTPUT FROM WHAT FILE: ZZINDIVIDUAL// <Enter>
Select ZZINDIVIDUAL NAME:  <space><Enter>  MODIFIED,ENTRY
ANOTHER ONE: <Enter>
STANDARD CAPTIONED OUTPUT? Yes// <Enter> (Yes)
Include COMPUTED fields:  (N/Y/R/B): NO// Record Number (IEN)

NUMBER: 14                          NAME: MODIFIED,ENTRY
  AREA CODE: 415                      LOCAL NUMBER: 555-1234
  PHONE NUMBER: (415) 555-1234      OLD PHONE NUMBER: (415) 555-1234
  DATE CREATED: <Date/Time Created>
```

The reindexing call IX1^DIK caused the Set Logic of the MUMPS "AE" index you created in Lesson 7 to be executed. When an entry is reindexed, fields aren't being edited, and so the old field values in the X1 array and the new field values in the X2 array both equal the current field values. Hence, when the Set Logic of our "AE" cross-reference is executed, both the OLD PHONE NUMBER and PHONE NUMBER fields reflect the current values of the AREA CODE (#3.1) and LOCAL NUMBER (#3.2) fields.

End of Exercise 8.1.

Exercise 8.2. See How Removing "R" from Activity Affects the IX1^DIK Call

In this exercise, you will remove the "R" from the Activity property of our "AE" cross-reference. This will cause the Set Logic of that cross-reference not to be executed when you reindex a record with IX1^DIK.

Step 1. Before you begin this exercise, you need to re-establish the starting point for the values of the fields PHONE NUMBER and OLD PHONE NUMBER. To do this, enter the following command from programmer mode:

```
>D MODENT^A6AKIT(662nnn)
```

Where *662nnn* is the number of your copy of the tutorial test file. This is equivalent to performing Steps 1 and 2 from exercise 8.1.

Step 2. Edit the "AE" MUMPS cross-reference:

```
Select OPTION NAME: UTILITY FUNCTIONS
Select UTILITY OPTION NAME: CROSS-REFERENCE A FIELD OR FILE

What type of cross-reference (Traditional or New)? Traditional// NEW

MODIFY WHAT FILE: ZZINDIVIDUAL// <Enter>
Select Subfile: <Enter>

Current Indexes on file #662nnn:
  nnn    'C' index
  nnn    'D' index
  nnn    'AD' index
  nnn    'AE' index

Choose E (Edit)/D (Delete)/C (Create): EDIT

Which Index do you wish to edit? AE  <Enter>
```

Step 3. On the first page of the ScreenMan, tab to the Activity field and remove the "R".

```
 ┌─────────────────────────────────────────────────────────────────────┐
 │  Number: nnn                  EDIT AN INDEX              Page 1 of 2  │
 │ ──────────────────────────────────────────────────────────────────  │
 │                                                                      │
 │                 File: 662nnn            Root File: 662nnn            │
 │           Index Name: AE                Root Type: INDEX FILE        │
 │                                                                      │
 │     Short Description: Update the PHONE NUMBER field when the AREA    │
 │                        CODE and LOCAL N…                             │
 │      Description (wp): (empty)                                       │
 │                                                                      │
 │                 Type: MUMPS                                          │
 │                                                                      │
 │             Activity: I                                             │
 │            Execution: RECORD                                         │
 │                                                                      │
 │                  Use: ACTION                                         │
 │                                                                      │
 │ ──────────────────────────────────────────────────────────────────  │
 │                                                                      │
 │                                                                      │
 │   COMMAND:                          Press <>PF1>H for help    Insert │
 └─────────────────────────────────────────────────────────────────────┘
```

> Here is your Short Description. Remember, only a portion of it can be seen at one time in the ScreenMan editing window.

ℹ️ You actually would probably also want to remove the "I" from Activity to suppress the firing of this cross-reference during a KIDS installation when data is sent with the file.

Step 4. Press <PF1>E to exit (and save) the form.

Step 5. As in Step 3 of Exercise 1, make an IX1^DIK call to reindex the MODIFIED,ENTRY entry:
```
>S DIK="^DIZ(662nnn,",DA=14 D IX1^DIK
```

Step 6. Use VA FileMan's Inquire to File Entries option to see the effect of reindexing the entry.
```
NUMBER: 14                          NAME: MODIFIED,ENTRY
  AREA CODE: 415                      LOCAL NUMBER: 555-1234
  PHONE NUMBER: (415) 555-1234       OLD PHONE NUMBER: (111) 111-1111
  DATE CREATED: <Date/Time Created>
```

ℹ️ The PHONE NUMBER and OLD PHONE NUMBER fields haven't been changed. The reindexing call IX1^DIK did not execute the "AE" MUMPS cross-reference because you removed the "R" from the Activity property of that cross-reference.

End of Exercise 8.2.

Lesson 8. Quiz

This is the quiz for Lesson 8 of the VA FileMan V. 22.0 Key and Index Tutorial. Test yourself on what you've learned in Lesson 8. The correct answers can be found at the bottom of the page, or by selecting the link at each question.

Question 1: The Activity cross-reference property allows you to prevent a particular cross-reference from firing during a re-indexing operation and/or a KIDS installation. By default, when you create a New-style cross-reference, Activity is automatically set to:

    A. 'XR'.
    B. 'IR'.
    C. Neither of the above.

Click here for answer

Question 2: Which character label must be removed from Activity to suppress firing a cross-reference when FileMan's reindexing APIs (xxx^DIK) and Re-Index File option are called?

    A. 'I'.
    B. 'R'.
    C. Neither of the above.

Click here for answer

Question 3: If you explicitly select a cross-reference in an EN^DIK, EN1^DIK, or ENALL^DIK call, or in the UTILITY FUNCTIONS/RE-INDEX FILE option on the VA FileMan menu, that cross-reference will be fired whether or not its Activity contains an 'R'.

    A. True.
    B. False.

Click here for answer

Question 4: Which character label must be removed from Activity to suppress firing a cross-reference during a KIDS installation?

    A. 'I'.
    B. 'R'.
    C. Neither of the above.

Click here for answer

Question 5: FileMan automatically fires cross-references during an edit, regardless of Activity, though you can control whether a cross-reference is fired by entering Set and Kill Conditions:

A. True.
B. False.

Click here for answer

Question 6: In this lesson, the reindexing call IX1^DIK caused the Set Logic of the MUMPS 'AE' index you created in Lesson 7 to be executed. When an entry is reindexed, fields aren't being edited, and so the old field values in the X1 array and the new field values in the X2 array both equal the current field values. Hence, in this lesson, when the Set Logic of our 'AE' cross-reference is executed, both the OLD PHONE NUMBER and PHONE NUMBER fields reflected:

A. The current values of the AREA CODE and LOCAL NUMBER fields.
B. The old values of the AREA CODE and LOCAL NUMBER fields.
C. Neither of the above.

Click here for answer

Correct Answers – Lesson 8 Quiz

Question 1: The Activity cross-reference property allows you to prevent a particular cross-reference from firing during a re-indexing operation and/or a KIDS installation. By default, when you create a New-style cross-reference, Activity is automatically set to:

   Answer: B. "IR".

Go Back

Question 2: Which character label must be removed from Activity to suppress firing a cross-reference when FileMan's reindexing APIs (xxx^DIK) and Re-Index File option are called?

   Answer: B. "R".

Go Back

Question 3: If you explicitly select a cross-reference in an EN^DIK, EN1^DIK, or ENALL^DIK call, or in the UTILITY FUNCTIONS/RE-INDEX FILE option on the VA FileMan menu, that cross-reference will be fired whether or not its Activity contains an 'R'.

   Answer: A. "True."

Go Back

Question 4: Which character label must be removed from Activity to suppress firing a cross-reference during a KIDS installation?

   Answer: A. "I".

Go Back

Question 5: FileMan automatically fires cross-references during an edit, regardless of Activity, though you can control whether a cross-reference is fired by entering Set and Kill Conditions:

   Answer: A. "True."

Go Back

Question 6: In this lesson, the reindexing call IX1^DIK caused the Set Logic of the MUMPS 'AE' index you created in Lesson 7 to be executed. When an entry is reindexed, fields aren't being edited, and so the old field values in the X1 array and the new field values in the X2 array both equal the current field values. Hence, in this lesson, when the Set Logic of our 'AE' cross-reference is executed, both the OLD PHONE NUMBER and PHONE NUMBER fields reflected:

   Answer: A. "The current values of the AREA CODE and LOCAL NUMBER fields."

Go Back

Part 3. [Keys](#)

Version 22 of VA FileMan allows you to define a database key on a file. A database key is a set of one or more fields that, when taken together, uniquely identifies a record in a file.

VA FileMan stores key definitions in the KEY file (#.31), under the global root ^DD("KEY").

Lesson 9. Create a Key

This lesson covers the following topics:

1. Key Integrity
2. Uniqueness Index
3. Primary and Secondary Keys

In the exercise that follows, you will create a key on the *ZZINDIVIDUAL* file.

Key Integrity

When you define a key, VA FileMan automatically enforces the integrity of that key. Key integrity means that:

1. The key is unique for all records in a file.
2. A field that is part of a key must have a value (i.e., it cannot be null).

Uniqueness Index

When you create a key in VA FileMan, an index, called the **Uniqueness Index**, is automatically created. This index contains as subscripts the fields in that key. The Uniqueness Index is simply a New-style Regular index that supports a key. VA FileMan uses the Uniqueness Index of a key to enforce the integrity of that key and to look up entries in the file based on the fields in the key.

For example, if you define a key and select NAME and SSN as the fields in that key, FileMan automatically creates a corresponding Uniqueness Index that contains as subscripts the NAME and SSN fields.

Primary and Secondary Keys

If a file has a key, exactly one key in that file must be designated the **primary key**. All other keys, if any, are **secondary keys**. VA FileMan enforces key integrity equally for both primary and secondary keys, but it uses the primary key as the principal means for looking up entries in a file. For example, in a ^DIC lookup, if the user enters a question mark (?) at the Select prompt, VA FileMan automatically displays the data in the primary key fields for each record listed.

> The .01 field should be defined as part of the primary key.

Exercise 9.1. Create Your First Key

In this exercise you will define your first key on the *ZZINDIVIDUAL* file. The fields in that key will be the NAME field (#.01) and the SSN field (#.02).

Step 1. Use VA FileMan's Key Definition option to create a new key named "A":

```
Select OPTION: UTILITY FUNCTIONS
Select UTILITY OPTION: KEY DEFINITION

MODIFY WHAT FILE: ZZINDIVIDUAL// <Enter>
Select Subfile: <Enter>

There are no Keys defined on file #662nnn.
Want to create a new Key for this file? No// YES

Enter a Name for the new Key: A// <Enter> A
  Creating new Key 'A' ...
```

> **i** Key names must be one uppercase letter. You should give the primary key of your file or subfile the name "A". Subsequent secondary keys should be given the names "B", "C", and so on.

Step 2. You are now presented with a one-page ScreenMan form. Here you can select the fields in your new key.

In the first row of the "KEY FIELDS" section, under the "Field" column, enter NAME (or .01). In the "Seq No." column, enter 1. In the second row of the "KEY FIELDS" section, enter SSN (or .02), and in the "Seq No." column, enter 2.

The screen should now look like this:

```
Number: nn                    EDIT A KEY                    Page 1 of 1
_____

   File: 662nnn          Name: A                      Priority: PRIMARY

   KEY FIELDS:
   =========
   Field           Seq No.  File            Field Name
   -----           -------  ----            ----------
   .01                1     662nnn          NAME
   .02                2     662nnn          SSN



   Uniqueness Index:

    Index Details...

_____


 COMMAND:                              Press <PF1>H for help    Insert
```

> **i** The sequence number corresponds to the subscript number of the cross-reference value as it will be stored in the Uniqueness Index. The first field of every key should be given a sequence number of 1, the second 2, and so on.

Step 3. Press <PF1>E to exit the form.

Step 4. FileMan then indicates that it will create a Uniqueness Index to support the key you just created and prompts you for an index name. Select the default name "E".

```
I'm going to create a new Uniqueness Index to support Key 'A'
of File #662nnn.

Index Name: E// <Enter>

   One moment please ...
```

Step 5. If asked whether you want to build the new index, press Enter to select the default YES and press Enter again to continue.

```
Do you want to build the index now? YES// <Enter>
  Building new index ...  DONE!
```

> ⓘ At this point a Uniqueness Index to support Key A has been created. It is a compound index named "E", and its subscripts (cross-reference values) are the NAME and SSN fields.

Step 6. Answer YES to check key integrity now.
```
Do want to check the integrity of this key now? YES

Checking key integrity ...
```

Step 7. If any of the records in your file violate key integrity, you are presented with a list of options. A problem will probably be found. If so, select option 3 to ignore the problem for now. We know that we want to define the NAME and SSN fields as Key A. If any of the existing data in the file causes key integrity to be violated, we will correct it later.

```
ERROR: The key is not unique and/or some records have key
field values missing.

      Select one of the following:

          1         Delete the Key (also selected on up-arrow)
          2         Re-Edit the Key
          3         Ignore problem (Be sure to fix later)

Enter response: 3 <Enter>  Ignore problem (Be sure to fix later)
```

> ⓘ The problem here is that one or more records in the file violate the integrity of the key. There may be fields with missing key values, or two records may have exactly the same NAME and SSN. You will find out what the specific problem is in the next few steps.

Step 8. At this point, the key is successfully defined, and FileMan shows you the basic information about the key. At the list of options, choose VERIFY to determine which record(s) violated key integrity in Step 7.
```
Keys defined on file #662nnn:

   A  PRIMARY KEY    Uniqueness Index: E
         Field(s):  1) NAME (#.01)
                    2) SSN (#.02)

Choose V (Verify)/E (Edit)/D (Delete)/C (Create): VERIFY
```

Step 9. Select Key A, the key you just created, as the key to verify. Since our test file has only a few records in it, don't store the internal entry numbers of the records that violate key integrity in a template - just press Enter at the "STORE THESE ENTRY ID'S IN TEMPLATE:" prompt. Press Enter at the "DEVICE:" prompt to print the results to the screen.

```
Which Key do you wish to verify? A// <Enter>
STORE THESE ENTRY ID'S IN TEMPLATE: <Enter>

DEVICE: HOME// <Enter>
```

```
KEY INTEGRITY CHECK                      SEP 21, 2000 14:35    PAGE 1
────────────────────────────────────────────────────────────────────────

             Key: A (#nn), File #662nnn
Uniqueness Index: E (#nnn)


ENTRY #    NAME                     ERROR
-------    ----                     -----
14         MODIFIED,ENTRY           Missing Key Fields(s):
                                      SSN [662nnn,.02]
```

You created this entry in a previous lesson. However, you never added an SSN for this record, and it is the only record that violates key integrity.

Step 10. Use VA FileMan's Enter or Edit File Entries option to correct edit the SSN field for the MODIFIED,ENTRY record to 666456789.

```
Select OPTION NAME: ENTER OR EDIT FILE ENTRIES

INPUT TO WHAT FILE: ZZINDIVIDUAL// <Enter>
EDIT WHICH FIELD: ALL// SSN
THEN EDIT FIELD: <Enter>

Select ZZINDIVIDUAL NAME: MODIFIED,ENTRY
SSN: 666456789
```

Step 11. Now, go back to the Key Definition option and check key integrity again.

```
Select OPTION NAME: UTILITY FUNCTIONS
Select UTILITY OPTION NAME: KEY DEFINITION

MODIFY WHAT FILE: ZZINDIVIDUAL// <Enter>
Select Subfile: <Enter>

Keys defined on file #662nnn:

  A  PRIMARY KEY    Uniqueness Index: E
         Field(s):  1) NAME (#.01)
                    2) SSN (#.02)
```

```
Choose V (Verify)/E (Edit)/D (Delete)/C (Create): VERIFY

Which Key do you wish to verify? A// <Enter>
STORE THESE ENTRY ID'S IN TEMPLATE: <Enter>

DEVICE: HOME// <Enter>  SYSTEM
```

```
KEY INTEGRITY CHECK                        SEP 22, 2000 10:00    PAGE 1
_____

               Key: A (#nn), File #662nnn
Uniqueness Index: E (#nnn)


** NO PROBLEMS **
```

This time, no problems were found! Every entry in the file has values for NAME and SSN, and the combination of those two fields is unique for all entries in the file.

End of Exercise 9.1.

Congratulations! You have just created your first key!

Lesson 9. Quiz

This is the quiz for Lesson 9 of the VA FileMan V. 22.0 Key and Index Tutorial. Test yourself on what you've learned in Lesson 9. The correct answers can be found at the bottom of the page, or by selecting the link at each question.

Question 1: Version 22 of VA FileMan allows you to define a database key on a file. What is a database key?

    A. A set of one or more fields that, when taken together, uniquely identifies a record in a file.
    B. A combination of the .01 field and any other field, that, when taken together, uniquely identifies a record in a file.
    C. Neither of the above.

Click here for answer

Question 2: The definition of a key is stored in what file?

    A. KEY AND INDEX file (#.13).
    B. KEY file (#.31).
    C. Neither of the above.

Click here for answer

Question 3: When you define a key, FileMan automatically enforces the integrity of that key. Key integrity means that:

    A. The key is unique for all records in a file and that all fields that are part of the key, must have values (i.e., they cannot be null).
    B. The key is unique for all records in a file and that the .01 field must have a value (i.e., it cannot be null).
    C. Both of the above.

Click here for answer

Question 4: When you create a key in FileMan, a New-style Regular index, called the Uniqueness Index, is automatically created. This index contains as subscripts:

    A. The fields in that key.
    B. The IEN of the key.
    C. Neither of the above.

Click here for answer

Question 5: FileMan uses the Uniqueness Index of a key to:

    A. Enforce the integrity of that key.
    B. Lookup entries in the file based on the fields in the key.
    C. Both of the above.

Click here for answer

Question 6: If a file has a key, exactly one key in that file must be designated the primary key. All other keys, if any, are secondary keys.

    A. True.
    B. False.

Click here for answer

Question 7: FileMan enforces key integrity equally for both primary and secondary keys, but it uses the secondary key as the principal means for looking up entries in a file.

    A. True.
    B. False.

Click here for answer

Question 8: If the NAME and SSN fields make up a key, which of the following conditions would violate key integrity?

    A. The SSN for a record is missing.
    B. Two records may have exactly the same NAME and SSN.
    C. Both of the above.

Click here for answer

Correct Answers – Lesson 9 Quiz

Question 1: Version 22 of VA FileMan allows you to define a database key on a file. What is a database key?

Answer: A. "A set of one or more fields that, when taken together, uniquely identifies a record in a file."

Go Back

Question 2: The definition of a key is stored in what file?

Answer: B. "KEY file (#.31)."

Go Back

Question 3: When you define a key, FileMan automatically enforces the integrity of that key. Key integrity means that:

Answer: A. "The key is unique for all records in a file and that all fields that are part of the key, must have values (i.e., they cannot be null)."

Go Back

Question 4: When you create a key in FileMan, a New-style Regular index, called the Uniqueness Index, is automatically created. This index contains as subscripts:

Answer: A. "The fields in that key."

Go Back

Question 5: FileMan uses the Uniqueness Index of a key to:

Answer: C. "Both of the above." (FileMan uses the Uniqueness Index of a key to enforce the integrity of that key and to lookup entries in the file based on the fields in the key.)

Go Back

Question 6: If a file has a key, exactly one key in that file must be designated the primary key. All other keys, if any, are secondary keys.

Answer: A. True.

Go Back

Question 7: FileMan enforces key integrity equally for both primary and secondary keys, but it uses the secondary key as the principal means for looking up entries in a file.

Answer: B. False. (FileMan uses the primary key as the principal means for looking up entries in a file.)

Go Back

Question 8: If the NAME and SSN fields make up a key, which of the following conditions would violate key integrity?

Answer: C. Both of the above. (If the SSN for a record is missing and if two records have exactly the same NAME and SSN, key integrity would be violated.)

Go Back

Lesson 10. Print Key Definition and View Uniqueness Index

In Lesson 9, you created a primary key on the *ZZINDIVIDUAL* file, with key fields NAME (#.01) and SSN (#.02). In this lesson, you will print the definition of the key and look at the Uniqueness Index automatically created. You will also explore different methods for defining the fields in the key.

The "Keys Only" Data Dictionary Listing

The standard data dictionary listing in Version 22 has been modified to display the keys defined on a file or subfile. In addition, a new **Keys Only** type of listing has been created to display only the keys defined on a file.

Exercise 10.1. Display the Key You Created in Lesson 9

In this exercise, you will use the Keys Only data dictionary listing to take a look at the key (Key A) you defined in Lesson 9. You will also look at the Uniqueness Index VA FileMan automatically created to support the key.

Step 1.   Use VA FileMan's List File Attributes option to select the "Keys Only" listing format:

```
Select OPTION: DATA DICTIONARY UTILITIES
Select DATA DICTIONARY UTILITY OPTION: LIST FILE ATTRIBUTES
START WITH WHAT FILE: ZZINDIVIDUAL
    GO TO WHAT FILE: ZZINDIVIDUAL// <Enter>
    Select SUB-FILE: <Enter>
Select LISTING FORMAT: STANDARD// ??

   Choose from:
   1          STANDARD
   2          BRIEF
   3          CUSTOM-TAILORED
   4          MODIFIED STANDARD
   5          TEMPLATES ONLY
   6          GLOBAL MAP
   7          CONDENSED
   8          INDEXES ONLY
   9          KEYS ONLY

Select LISTING FORMAT: STANDARD// KEYS ONLY
DEVICE: ;80;999  <Enter>
```

The output is a report of all the keys defined on the *ZZINDIVIDUAL* test file (there is only one):

```
KEY LIST -- FILE #662nnn                              09/26/00   PAGE 1
_____

  FILE #662nnn
  -----------
  PRIMARY KEY:        A (#nnn)
    Uniqueness Index: E (#nnn)
          File, Field: 1) NAME (662nnn,.01)  2) SSN (662nnn,.02)
```

Step 2. Do another data dictionary listing, but this time select the INDEXES ONLY format to view the definition of the "E" Uniqueness Index.

```
Select DATA DICTIONARY UTILITY OPTION NAME: LIST FILE ATTRIBUTES
   START WITH WHAT FILE: ZZINDIVIDUAL// <Enter>
       GO TO WHAT FILE: ZZINDIVIDUAL// <Enter>
       Select SUB-FILE: <Enter>
Select LISTING FORMAT NAME: STANDARD// INDEXES ONLY
What type of cross-reference (Traditional or New)? Both// <Enter>
BOTH
Which field: ALL// <Enter>
DEVICE: ;80;999

INDEX AND CROSS-REFERENCE LIST -- FILE #662nnn   09/26/00 PAGE 1
---------------------------------------------------------------

File #662nnn

...

  E (#nnn)    RECORD    REGULAR    IR    LOOKUP & SORTING
       Unique for:  Key A (#nn), File #662nnn
      Short Descr:  Uniqueness Index for Key 'A' of File #662nnn
        Set Logic:  S ^DIZ(662nnn,"E",X(1),X(2),DA)=""
       Kill Logic:  K ^DIZ(662nnn,"E",X(1),X(2),DA)
       Whole Kill:  K ^DIZ(662nnn,"E")
             X(1):  NAME  (662nnn,.01)  (Subscr 1)
             X(2):  SSN  (662nnn,.02)  (Subscr 2)

...
```

> **i** The report includes the definition of the "E" index. The "Unique for" line indicates that this Regular compound index is the Uniqueness Index for Key A. VA FileMan created this Uniqueness Index automatically when you created the key.

Step 3. Now, from programmer mode, use ^%G to look at the data in the "E" index, the Uniqueness Index for Key A.

```
>D ^%G
```

```
Device: <Enter>  Right margin: 80=> <Enter>

Global ^DIZ(662nnn,"E"
       DIZ(662nnn,"E"
^DIZ(662nnn,"E","FMPATIENT,EIGHT",666891234,8) =
^DIZ(662nnn,"E","FMPATIENT,ELEVEN",666765432,11)=
^DIZ(662nnn,"E","FMPATIENT,FIVE",666654321,5)=
^DIZ(662nnn,"E","FMPATIENT,FOUR",666223333,4)=
^DIZ(662nnn,"E","FMPATIENT,NINE",666345678,9)=
^DIZ(662nnn,"E","FMPATIENT,ONE","000221111",1)=
^DIZ(662nnn,"E","FMPATIENT,SEVEN",666678901,7)=
^DIZ(662nnn,"E","FMPATIENT,SIX",666889999,6)=
^DIZ(662nnn,"E","FMPATIENT,TEN",666432123,10)=
^DIZ(662nnn,"E","FMPATIENT,THIRTEEN",666567890,13)=
^DIZ(662nnn,"E","FMPATIENT,THREE",666776666,3)=
^DIZ(662nnn,"E","FMPATIENT,TWELVE",666996666,12)=
^DIZ(662nnn,"E","FMPATIENT,TWO",666443333,2)=
^DIZ(662nnn,"E","MODIFIED,ENTRY",666456789,14)=
```

End of Exercise 10.1.

Defining the Fields in a Key

The Key Definition option gives you two ways to define fields in a key:

1. You can enter the fields in the key in the KEY FIELDS section on the ScreenMan form, and let VA FileMan create or modify the Uniqueness Index for you. This is how you will usually define a key, and is the method you used to create Key A in Lesson 9.

2. You can select a New-style Regular index as the Uniqueness Index for the key, and when prompted, instruct VA FileMan to make all the fields in the index key fields. The Uniqueness Index you select, however, must meet the following conditions:

    *   It must be a Regular, New-style cross-reference.
    *   It must be used for lookup and sorting; that is, it cannot have a name that starts with the letter "A".
    *   It cannot have any Set or Kill Conditions.
    *   It must consist of only field-type cross-reference values, all of which are used as subscripts; that is, it can contain no computed values.
    *   No subscripts can have transforms.

In the next three exercises you will see how the above two methods work in defining the key fields in Key A.

Exercise 10.2. Add DOB Field (#.03) to Primary Key A

In this exercise, you will add the DOB field to Primary Key A and have VA FileMan automatically modify the "E" Uniqueness Index to include the DOB field.

Step 1. Select VA FileMan's KEY DEFINITION option, and edit Key A.

```
Select OPTION NAME: UTILITY FUNCTIONS
Select UTILITY OPTION NAME: KEY DEFINITION

MODIFY WHAT FILE: ZZINDIVIDUAL// <Enter>
Select Subfile: <Enter>

Keys defined on file #662nnn:

  A  PRIMARY KEY    Uniqueness Index: E
        Field(s):  1) NAME (#.01)
                   2) SSN (#.02)

Choose V (Verify)/E (Edit)/D (Delete)/C (Create): EDIT

Which Key do you wish to edit? A// <Enter>
```

Step 2. You are now presented with a one-page ScreenMan form where you can edit the properties of the key.

```
Number: nn                    EDIT A KEY                  Page 1 of 1
 ───────────────────────────────────────────────────────────────────

   File: 662nnn        Name: A                 Priority: PRIMARY

   KEY FIELDS:
   =========
   Field          Seq No.  File              Field Name
   -----          -------  ----              ----------
   .01               1     662nnn            NAME
   .02               2     662nnn            SSN



   Uniqueness Index: 662nnn                  E
               Uniqueness Index for Key 'A' of File #662nnn

    Index Details...
 ───────────────────────────────────────────────────────────────────


 COMMAND:                        Press <PF1>H for help    Insert
```

> ⓘ The Uniqueness Index field on the form shows you some of the information about the
> Uniqueness Index used by Key A; namely, the file on which the index resides, the
> name of the index, and the short description of the index.

Step 3. Navigate to "Index Details…" and press Enter. A pop-up window displays more detailed information about the Uniqueness Index.

```
       File: 662nnn                          Root File: 662nnn
  Index Name: E                              Root Type: INDEX FILE

  Short Description: Uniqueness Index for Key 'A' of File #662nnn
   Description (wp): (empty)

  Order...  Subscr  Length  [File,Field] Field Name
  --------  ------  ------  ----------------------
     1         1            [662nnn,.01] NAME
     2         2            [662nnn,.02] SSN

  Set Logic: S ^DIZ(662nnn,"E",X(1),X(2),DA)=""
 Kill Logic: K ^DIZ(662nnn,"E",X(1),X(2),DA)
```

> **ℹ** On this pop-up window you can edit only some of the properties of the Uniqueness Index - the Index Name, the Short Description, the Description (wp), and the subscript Length.

Step 4. Press <PF1>C to close the pop-up window. You won't make any changes to the Uniqueness Index here.

Step 5. Navigate to the third row in the "Field" column of the KEY FIELDS section. Enter DOB (or .03) to add the DOB field as a third field primary Key A. In the "Seq No." column for the DOB field, enter 3.

```
Number: nn                    EDIT A KEY                 Page 1 of 1
_____

  File: 662nnn         Name: A                    Priority: PRIMARY

  KEY FIELDS:
  =========
  Field           Seq No.  File           Field Name
  -----           -------  ----           ----------
  .01               1      662nnn         NAME
  .02               2      662nnn         SSN
  DOB <Enter>       3 <Enter>


  Uniqueness Index: 662nnn                 E
              Uniqueness Index for Key 'A' of File #662nnn

   Index Details...
  _____


  COMMAND:                           Press <PF1>H for help    Insert
```

Step 6. Press <PF1>E to exit the form and save changes.

Step 7. At this point, the Uniqueness Index "E" contains only the two fields NAME and SSN. VA FileMan notices the discrepancy between this and the fact that you specified the three fields NAME, SSN, and DOB in the "KEY FIELDS" section of the form. So, VA FileMan asks you how you want to resolve this discrepancy.

```
The Key fields and the fields in the Uniqueness Index don't match.

      Select one of the following:

          1    Re-Edit the Key
          2    Make Key match Uniqueness Index (also selected on
               up-arrow)
          3    Make Uniqueness Index match Key

  Enter response: 3 <Enter> Make Uniqueness Index match Key

     Modifying fields in Key ... DONE!
```

Select option number 3, "Make Uniqueness Index match Key." This will make VA FileMan modify the definition of the Uniqueness Index so that it contains the three fields listed in the KEY FIELDS section of the form.

Step 8. If prompted to delete the old and rebuild the new index, answer YES to both questions. Press Enter again to continue.

```
Do you want to delete the old index now? YES// <Enter>
  Removing old index ...  DONE!

Do you want to build the index now? YES// <Enter>
  Building new index ...  DONE!
```

Step 9. Next, you are asked whether you want to check the integrity of the key. Since you are going to modify this key again in the next exercise, answer NO to this question.

```
Do want to check the integrity of this key now? NO
```

End of Exercise 10.2

Exercise 10.3. Make the "C" Index the Uniqueness Index for Key A

In this exercise, you will select the "C" index created in Lesson 1 as the Uniqueness Index for Key A. Then you will have VA FileMan automatically redefine the key so that the fields in that index (DOB and SSN) are the key fields.

Step 1. Select VA FileMan's KEY DEFINITION option, and edit Key A.

```
Select OPTION NAME: UTILITY FUNCTIONS
Select UTILITY OPTION NAME: KEY DEFINITION

MODIFY WHAT FILE: ZZINDIVIDUAL// <Enter>
Select Subfile: <Enter>

Keys defined on file #662nnn:

  A  PRIMARY KEY    Uniqueness Index: E
        Field(s):  1) NAME (#.01)
                   2) SSN (#.02)
                   3) DOB (#.03)

Choose V (Verify)/E (Edit)/D (Delete)/C (Create): EDIT

Which Key do you wish to edit? A// <Enter>
```

Step 2. On the ScreenMan form, navigate to "Index Details…" and press Enter. Notice that as a result of Exercise 10.2, VA FileMan added the DOB field to the "E" Uniqueness Index. Press <PF1>C to close the pop-up window.

Step 3. Now, navigate to "Uniqueness Index" and enter C to select as the Uniqueness Index for Key A the "C" index you created in Lesson 1 of this tutorial. The "C" index is a compound index with two fields, DOB and SSN.

```
┌─────────────────────────────────────────────────────────────────────────┐
│ Number: nn                    EDIT A KEY               Page 1 of 1        │
│ ─────────────────────────────────────────────────────────────────────── │
│                                                                           │
│   File: 662nnn          Name: A                   Priority: PRIMARY       │
│                                                                           │
│   KEY FIELDS:                                                             │
│   ==========                                                             │
│   Field            Seq No.  File            Field Name                    │
│   -----            -------  ----            ----------                    │
│   .01                1      662nnn          NAME                          │
│   .02                2      662nnn          SSN                           │
│   .03                3      662nnn          DOB                           │
│                                                                           │
│                                                                           │
│   Uniqueness Index: C <Enter>                                            │
│           This is a Regular index on the DOB and SSN fields.             │
│                                                                           │
│    Index Details...                                                       │
│                                                                           │
│  ─────────────────────────────────────────────────────────────────────  │
│                                                                           │
│                                                                           │
│  COMMAND:                              Press <PF1>H for help    Insert    │
└─────────────────────────────────────────────────────────────────────────┘
```

> **i** Notice that entering "C" as the "Uniqueness Index:" property displayed a description of the index. If you'll remember, this is the Short Description you entered for the "C" index in Lesson 1 of this tutorial.

Step 4. Press <PF1>E to exit the form and save changes.

Step 5. Since you changed the Uniqueness Index for Key A from "E" to "C", VA FileMan asks if you no longer need the "E" index and whether you want to delete it. Answer NO. In a later step, you will again make index "E" the Uniqueness Index for Key A.

```
Do you want to delete the 'E' Uniqueness Index (#nnn) on File
#662nnn previously used by Key 'A' of File #662nnn? NO
```

Step 6. Now VA FileMan recognizes that the fields listed in the KEY FIELDS section of the ScreenMan form (NAME, SSN, and DOB) don't match the fields defined in the "C" index (DOB and SSN), and asks you how you want to resolve this discrepancy. This time, select option 2, "Make Key match Uniqueness Index."

```
The Key fields and the fields in the Uniqueness Index don't match.
```

```
      Select one of the following:

          1    Re-Edit the Key
          2    Make Key match Uniqueness Index (also selected on
               up-arrow)
          3    Make Uniqueness Index match Key

   Enter response: 2 Make Key match Uniqueness Index (also selected on
            up-arrow)

   Modifying fields in Key ...   DONE!
```

Selecting option 2 makes the KEY FIELDS match the fields in the "C" Uniqueness Index, in this case, the DOB and SSN fields.

Step 7. Answer NO when asked if you want to check the integrity of the key.

```
   Do want to check the integrity of this key now? NO

   Keys defined on file #662nnn:

   A  PRIMARY KEY    Uniqueness Index: C
          Field(s):  1) DOB (#.03)
                     2) SSN (#.02)
```

The Uniqueness Index is now "C", and the two key fields are DOB and SSN.

End of Exercise 10.3

Exercise 10.4. Restore Key A To Its Original Definition

In this exercise you will restore the definition of Key A so that it contains the fields NAME and SSN, and uses "E" as its Uniqueness Index.

Step 1. Select the VA FileMan's KEY DEFINITION option, and edit Key A.

```
   Select OPTION NAME: UTILITY FUNCTIONS
   Select UTILITY OPTION NAME: KEY DEFINITION

   MODIFY WHAT FILE: ZZINDIVIDUAL// <Enter>
   Select Subfile: <Enter>

   Keys defined on file #662nnn:

     A  PRIMARY KEY    Uniqueness Index: C
            Field(s):  1) DOB (#.03)
                       2) SSN (#.02)

   Choose V (Verify)/E (Edit)/D (Delete)/C (Create): EDIT

   Which Key do you wish to edit? A// <Enter>
```

Step 2. On the ScreenMan form, navigate to the second row in the "Field" column of the "KEY FIELDS" section and change the .03 to .01.

Step 3. Navigate to "Uniqueness Index" and enter E. The screen should look like this:

```
Number: nn                      EDIT A KEY                    Page 1 of 1
_____

  File: 662nnn          Name: A                    Priority: PRIMARY

  KEY FIELDS:
  =========
  Field          Seq No.  File            Field Name
  -----          -------  ----            ----------
  .02               2     662nnn          SSN
  .01               1     662nnn          NAME


  Uniqueness Index: 662nnn                   E
         Uniqueness Index for Key 'A' of File #662nnn

   Index Details...
  _____


  COMMAND:                              Press <PF1>H for help    Insert
```

Step 4. Press <PF1>E to exit the form and save changes.

Step 5. Answer NO when asked if you want to delete the "C" index.

```
Do you want to delete the 'C' Uniqueness Index (#nnn) on File
#662nnn previously used by Key 'A' of File #662nnn? NO
```

Step 6. Again the KEY FIELDS (NAME and SSN) don't match the fields in the "E" Uniqueness Index (NAME, SSN, and DOB). Select option 3 to have FileMan modify the "E" index.

```
The Key fields and the fields in the Uniqueness Index don't match.

     Select one of the following:

          1    Re-Edit the Key
          2    Make Key match Uniqueness Index (also selected on
               up-arrow)
          3    Make Uniqueness Index match Key

Enter response: 3 Make Uniqueness Index match Key

   Modifying fields in Key ...  DONE!
```

Step 7. Answer YES if asked whether you want to delete the old and build the new "E" index. Press Enter to continue.

```
Do you want to delete the old index now? YES// <Enter>
  Removing old index ...  DONE!

Do you want to build the index now? YES// <Enter>
  Building new index ...  DONE!
```

Step 8. Answer YES to check key integrity. There should be no problems.

```
Do want to check the integrity of this key now? YES

Checking key integrity ...  NO PROBLEMS

Press RETURN to continue: <Enter>

Keys defined on file #662nnn:
   A PRIMARY KEY Uniqueness Index: E
      Field(s): 1) NAME (#.01)
                2) SSN (#.02)
```

End of Exercise 10.4

Lesson 10. Quiz

This is the quiz for Lesson 10 of the VA FileMan V. 22.0 Key and Index Tutorial. Test yourself on what you've learned in Lesson 10. The correct answers can be found at the bottom of the page, or by selecting the link at each question.

Question 1: The Keys Only data dictionary listing is new with FileMan V. 22.0. The output is a report of:

    A. All the keys defined on a selected file.
    B. All the primary and secondary keys defined on a selected file.
    C. Both of the above.

Click here for answer

Question 2: The Indexes Only data dictionary listing displays the definition of the Uniqueness Index for a key. In this listing, the 'Unique for' line gives what information?

    A. The name of the key, the IEN of the key in the KEY file, and the number of the file on which the key is defined.
    B. The cross-reference values for the key, and the file number for which this is a key.
    C. Both of the above.

Click here for answer

Question 3: How do you define fields in a key?

    A. You can enter the fields in the key in the KEY FIELDS section on the ScreenMan form, and let FileMan create or modify the Uniqueness Index for you.
    B. You can select a New-style Regular index as the Uniqueness Index for the key. Then, when prompted, instruct FileMan to make all the fields in the index the fields in the key.
    C. Both of the above.

Click here for answer

Question 4: Which FileMan option would you select to resolve a discrepancy where the Uniqueness Index you've selected contains only two fields, but you want the key to contain the three fields you specified in the 'KEY FIELDS' section of the ScreenMan form?

    A. Make Uniqueness Index match Key.
    B. Make Key match Uniqueness Index.
    C. Neither of the above.

Click here for answer

Question 5: You can select a MUMPS cross-reference as the Uniqueness Index of a key as long as that cross-reference maintains an index?

     A. True.
     B. False.

Click here for answer

Question 6: The Key Definition option allows you to:

     A. Create, edit, and delete a key.
     B. Verify the integrity of a key.
     C. Both of the above.

Click here for answer

Correct Answers – Lesson 10 Quiz

Question 1: The Keys Only data dictionary listing is new with FileMan V. 22.0. The output is a report of:

Answer: C. "Both of the above." (Primary and secondary are the only possible key definitions on any given VA FileMan file.)

Go Back

Question 2: The Indexes Only data dictionary listing displays the definition of the Uniqueness Index for a key. In this listing, the 'Unique for' line gives what information?

Answer: A. "The name of the key, the IEN of the key in the KEY file, and the number of the file on which the key is defined."

Go Back

Question 3: How do you define fields in a key?

Answer: C. "Both of the above." (1. You can enter the fields in the key in the KEY FIELDS section on the ScreenMan form, and let FileMan create or modify the Uniqueness Index for you. 2. You can select a New-style Regular index as the Uniqueness Index for the key. Then, when prompted, instruct FileMan to make all the fields in the index the fields in the key.)

Go Back

Question 4: Which FileMan option would you select to resolve a discrepancy where the Uniqueness Index you've selected contains only two fields, but you want the key to contain the three fields you specified in the 'KEY FIELDS' section of the ScreenMan form?

Answer: A. "Make Uniqueness Index match Key."

Go Back

Question 5: You can select a MUMPS cross-reference as the Uniqueness Index of a key as long as that cross-reference maintains an index?

Answer: B. "False."

Go Back

Question 6: The Key Definition option allows you to:

Answer: C. "Both of the above." (The Key Definition option allows you to create, edit, and delete a key, as well as verify key integrity.)

Go Back

Lesson 11. VA FileMan Key Integrity

In this lesson, you will see examples of how VA FileMan enforces the integrity of the key you created in Lesson 9, and worked with in Lesson 10.

When Does VA FileMan Check Key Integrity?

VA FileMan checks the integrity of keys at the following times:

1. **In the Key Definition option**
   When you create or edit a key, VA FileMan asks if you want to check the integrity of the key. You can select to verify a key at any time. (You've already seen examples of this.)
2. **In the Mandatory/Required Field Check option** (on VA FileMan's Utility Functions menu)
   If the field selected for checking is a key field, this option verifies that the field has a value (i.e., not null) for all records in the file.
3. **In the Verify Fields option** (on VA FileMan's Utility Functions menu)
   If the field selected for verification is a key field, the integrity of the key(s) of which it is a part is checked.
4. **When records are added**
   All of the VA FileMan APIs and options that add records (FILE^DICN, UPDATE^DIE, the Enter or Edit File Entries option, etc.) have been modified to verify the integrity of keys as records are added to the file. By default, if the new record violates key integrity, the record is deleted.
5. **When records are edited**
   Similarly, all of the VA FileMan APIs and options that allow you to edit the value of key fields (^DIE, FILE^DIE, ^DDS, the Enter or Edit File Entries option, etc.) have been modified to ensure that edits don't cause key integrity to be violated.
6. **In the Validator APIs**
   $$KEYVAL^DIE, VAL^DIE, and VALS^DIE all check the integrity of key fields.

The Editing APIs and Key Integrity

All of VA FileMan's editing APIs, including ^DIE and FILE^DIE, automatically enforce key integrity when you edit key fields.

^DIE checks the integrity of **simple keys** (i.e., one-field keys) immediately after the field is edited. It checks the integrity of **compound keys** (i.e., keys that are composed of more than one field) at the very end of the editing session, after all the fields in the record have been edited.

Exercise 11.1. Try to Delete the Value of the Key Field SSN

In this exercise, you will see how VA FileMan's Enter or Edit File Entries automatically enforces key integrity.

Step 1. Use VA FileMan's Enter or Edit File Entries option to edit the NAME field and the SSN field, the two fields in the Primary Key A you created in Lesson 9.

```
Select OPTION NAME: ENTER OR EDIT FILE ENTRIES

INPUT TO WHAT FILE: ZZINDIVIDUAL// <Enter>
EDIT WHICH FIELD: ALL// SSN
THEN EDIT FIELD: <Enter>
```

Step 2. Select the MODIFIED,ENTRY record and enter and at-sign (@) at the SSN field to try to delete it.

```
Select ZZINDIVIDUAL NAME: MODIFIED,ENTRY

SSN: 666456789// @ ??  Required Key field
SSN: 666456789// <Enter>
```

Because the SSN field is a key field, VA FileMan does not let you delete its value.

End of Exercise 11.1

Exercise 11.2. Try to Create a Duplicate Key

In this exercise, you will try to change the NAME and SSN of the MODIFIED,ENTRY record to see how VA FileMan handles edits that cause duplicate keys.

Step 1. Use VA FileMan's Enter or Edit File Entries option to edit the fields in Primary Key A: NAME field (#.01) and SSN field (#.02) in the *ZZINDIVIDUAL* file.

```
Select OPTION NAME: E <Enter>NTER OR EDIT FILE ENTRIES

INPUT TO WHAT FILE: ZZINDIVIDUAL// <Enter>
EDIT WHICH FIELD: ALL// .01  NAME
THEN EDIT FIELD: SSN
THEN EDIT FIELD: <Enter>
```

Step 2. Select the MODIFIED,ENTRY record and try to change the Name to FMPATIENT,FOUR and the SSN to 666223333.

```
Select ZZINDIVIDUAL NAME: MODIFIED,ENTRY     666456789
NAME: MODIFIED,ENTRY// FMPATIENT,FOUR
SSN: 666456789// 666223333

***** NOTE *****

Some of the previous edits are not valid because they create one or
more duplicate keys. Some fields have been restored to their pre-
edited values.
```

 ⓘ Another record in the *ZZINDIVIDUAL* already has these exact values for NAME and
    SSN. VA FileMan displayed a message indicating that this would violate key integrity.
    In addition, VA FileMan automatically restored the values of the NAME and SSN to
    their pre-edited values.

Step 3.   Answer YES when asked if you want to see a list of the fields that were edited with values that
    violate key integrity.

```
Do you want to see a list of those fields? YES// <Enter>

The following field(s) have been restored to their pre-edited
values:

File: ZZINDIVIDUAL File (#662nnn)
  Key: A
    Record: 'MODIFIED,ENTRY' (#14)

              Field: NAME (#.01)
      Invalid value: FMPATIENT,FOUR
        Restored to: MODIFIED,ENTRY

              Field: SSN (#.02)
      Invalid value: 666223333
        Restored to: 666456789
```

 ⓘ VA FileMan displays the record that was edited, the invalid key field values, and the
    original values to which the fields were restored.

Step 4.   Now try changing just the name of MODIFIED,ENTRY to FMPATIENT,FOUR.

```
Select OPTION NAME: ENTER OR EDIT FILE ENTRIES

INPUT TO WHAT FILE: ZZINDIVIDUAL// <Enter>
EDIT WHICH FIELD: ALL// .01 <Enter>  NAME
THEN EDIT FIELD: SSN
THEN EDIT FIELD: <Enter>

Select ZZINDIVIDUAL NAME: MODIFIED,ENTRY    666456789
NAME: MODIFIED,ENTRY// FMPATIENT,FOUR
SSN: 666456789// <Enter>
```

 ⓘ This time FileMan accepted our edits. Now there are two entries in the
    ZZINDIVIDUAL file with the name FMPATIENT,FOUR, but since they have different
    SSNs, key integrity has not been violated.

End of Exercise 11.2

The DIEFIRE Variable

You can set the variable DIEFIRE in an input template or within any of the semicolon-pieces of the DR input variable to ^DIE to instruct VA FileMan to fire the record-level indexes and check the integrity of all compound keys at that point in the template or DR string. If the integrity of any key is violated, VA FileMan sets the variable X to the string "BADKEY", which can be checked by M code in the next field in the template, or the next semicolon-piece of the DR string.

You can control what VA FileMan does if it finds any problems with the keys by setting DIEFIRE to null or any combination of the following flags:

M	=	Prints an error message to user.
L	=	Returns the DIEBADK array.
R	=	Restores invalid key fields to their pre-edited values.

If you include the L flag in DIEFIRE and a key is invalid, VA FileMan sets the DIEBADK array, which can be checked by M code in the next field in the template, or the next semicolon-piece of the DR string. The format of the DIEBADK array is as follows:

DIEBADK(rFile#,key#,file#,IENS,field#,"O") = The original value of the field.
DIEBADK(rFile#,key#,file#,IENS,field#,"N") = The new (invalid) value of the field.

where,

rFile#	=	The root file of the Uniqueness Index of the key. This is the file or subfile number of the fields that make up the key.
key#	=	The internal entry number (IEN) of the key in the KEY file.
file#	=	The file of the Uniqueness Index of the key. This is the file or subfile where the Uniqueness Index resides. For whole file indexes, this is a file or subfile at a higher level than root file.
IENS	=	The IENS of the record that - with the edits - would have a non-unique key.
field#	=	The field number of the field being edited.

Exercise 11.3. Use the DIEFIRE Variable to Check Key Integrity at a Specific Point

In this lesson you will create an input template that contains the NAME and SSN fields. Immediately after the SSN field, the input template sets the DIEFIRE variable to "R" to check the integrity of Key A and restore the fields to their original values if the edits to NAME and SSN cause a duplicate key. If edits result in a duplicate key, the input template sets Y to jump back to the NAME field, so that the user can try entering values for NAME and SSN again.

Step 1. Use VA FileMan's Enter or Edit File Entries option to create an input template called *ZZINDIVIDUAL*.

```
Select OPTION NAME: ENTER OR EDIT FILE ENTRIES

INPUT TO WHAT FILE: ZZINDIVIDUAL// <Enter>
```

```
EDIT WHICH FIELD: ALL// @1
THEN EDIT FIELD: .01 <Enter>  NAME
THEN EDIT FIELD: .02 <Enter>  SSN
```

> **i** The @1, which is entered as the first field, serves as a label to which we return if a duplicate key would be created with the user's new NAME and SSN values.

Step 2. Enter as the next field code to set DIEFIRE to an R.

```
THEN EDIT FIELD: S DIEFIRE="R"
```

> **i** This tells VA FileMan to fire the compound (record-level) cross-references and to validate any keys at this point. The "R" tells FileMan to restore the NAME and SSN to their pre-edited values if the user-supplied values resulted in a duplicate key.

Step 3. Enter the following as the next field in the template.

```
THEN EDIT FIELD: I X="BADKEY" W !!,"An entry already exists
with that NAME and SSN. Please try again.",! S Y="@1"
THEN EDIT FIELD: <Enter>
```

> **i** Here, immediately after the code that sets DIEFIRE, we check the variable X to see if it is set to "BADKEY". If so, the new NAME and SSN caused a duplicate, so we print a message to the user and branch back to label @1 in the template to prompt for NAME and SSN again.

Step 4. Save the input template as *ZZINDIVIDUAL*.

```
STORE THESE FIELDS IN TEMPLATE: ZZINDIVIDUAL
  Are you adding 'ZZINDIVIDUAL' as a new INPUT TEMPLATE? No//
  Y <Enter>  (Yes)
```

Step 5. Select the FMPATIENT,FOUR entry with SSN 666456789.

```
Select ZZINDIVIDUAL NAME: LAKE
     1    FMPATIENT,FOUR    666223333
     2    FMPATIENT,FOUR    666456789
CHOOSE 1-2: 2 <Enter>  FMPATIENT,FOUR  666456789
```

Step 6. Try to change the SSN to 666223333.

```
NAME: FMPATIENT,FOUR// <Enter>
SSN: 666456789// 666223333
```

```
An entry already exists with that NAME and SSN. Please try
again.
```

Step 7. FileMan now branches us back to the NAME field in our input template. This time enter MODIFIED,ENTRY for NAME and accept the default SSN.

```
NAME: FMPATIENT,FOUR// MODIFIED,ENTRY
SSN: 666456789// <Enter>
```

This time, no messages are printed, since this new NAME-SSN combination is not a duplicate. FileMan does not branch you back to NAME, and you exit the template.

End of Exercise 11.3.

Lesson 11. Quiz

This is the quiz for Lesson 11 of the VA FileMan V. 22.0 Key and Index Tutorial. Test yourself on what you've learned in Lesson 11. The correct answers can be found at the bottom of the page, or by selecting the link at each question.

Question 1: The Mandatory/Required Field Check option checks that all fields that are key fields or designated as required:

      A. Contain datA.
      B. Are unique.
      C. Both of the above.

Question 2: What FileMan option allows you to use a field's definition to verify the data stored in a file?

      A. Verify Fields.
      B. Modify File Attributes.
      C. Neither of the above.

Question 3: How has the FileMan option Enter or Edit File Entries been modified to automatically enforce key integrity?

      A. FileMan does not let you delete key field values.
      B. FileMan does not allow you to edit key field values.
      C. Both of the above.

Question 4: If you edit the key fields in a record, and FileMan encounters another record in the file that has the same key, a message is displayed indicating that making these edits would violate key integrity. In addition, FileMan automatically restores the key fields to their pre-edited values.

      A. True.
      B. False.

Question 5: How do you instruct FileMan to fire the record-level indexes and check the integrity of all compound keys at a particular point in an input template or the DR input variable to ^DIE?

    A. You can call the entry point FIRE^DIE.
    B. You can set the variable DIEFIRE.
    C. Both of the above.

Click here for answer

Question 6: If you set DIEFIRE to null ("") in an input template, record-level cross-references on edited fields would be fired at that point. If in addition, edits to key fields violate key integrity:

    A. The key fields are restored to their pre-edited values.
    B. X is set 'BADKEY'.
    C. Both of the above.

Click here for answer

Question 7: In an input template, how would you instruct FileMan to fire record-level cross-references, and, if key integrity is violated, restore the key fields to their pre-edited values?

    A. Set DIEFIRE to "R".
    B. Set DIEFIRE to "M".
    C. Both of the above.

Click here for answer

Correct Answers – Lesson 11 Quiz

Question 1: The Mandatory/Required Field Check option checks that all fields that are key fields or designated as required:

Answer: A. "Contain data."

Question 2: What FileMan option allows you to use a field's definition to verify the data stored in a file?

Answer: A. "Verify Fields."

Question 3: How has the FileMan option Enter or Edit File Entries been modified to automatically enforce key integrity?

Answer: A. "FileMan does not let you delete key field values."

Question 4: If you edit the key fields in a record, and FileMan encounters another record in the file that has the same key, a message is displayed indicating that making these edits would violate key integrity. In addition, FileMan automatically restores the key fields to their pre-edited values.

Answer: A. "True."

Question 5: How do you instruct FileMan to fire the record-level indexes and check the integrity of all compound keys at a particular point in an input template or the DR input variable to ^DIE?

Answer: B. "You can set the variable DIEFIRE."

Question 6: If you set DIEFIRE to null ("") in an input template, record-level cross-references on edited fields would be fired at that point. If in addition, edits to key fields violate key integrity:

Answer: B. "X is set 'BADKEY'."

Question 7: In an input template, how would you instruct FileMan to fire record-level cross-references, and, if key integrity is violated, restore the key fields to their pre-edited values?

    Answer: A. "Set DIEFIRE to 'R.'"

[Go Back](#)

Appendix: Test File

Standard Data Dictionary Listing of the Test File

The following is a standard data dictionary listing of the test file. In this listing, the name of the test file is *ZZINDIVIDUAL*, with file number *662nnn*, stored in global root ^*DIZ*(*662nnn*).

```
STANDARD DATA DICTIONARY #662nnn -- ZZINDIVIDUAL FILE          10/3/00  PAGE 1
STORED IN ^DIZ(662nnn,  (13 ENTRIES)   SITE: XXX    UCI: XXX,YYY


DATA            NAME                    GLOBAL        DATA
ELEMENT         TITLE                   LOCATION      TYPE
-----------------------------------------------------------------------------


               DD ACCESS: @
               RD ACCESS: @
               WR ACCESS: @
              DEL ACCESS: @
            LAYGO ACCESS: @
            AUDIT ACCESS: @


CROSS
REFERENCED BY: NAME(B)

   CREATED ON: OCT 3,2000 by USER,TEST

662nnn,.01     NAME                    0;1 FREE TEXT (Required)

               INPUT TRANSFORM:  K:$L(X)>30!($L(X)<3)!'(X'?1P.E) X
               HELP-PROMPT:      Answer must be 3-30 characters in length.
               CROSS-REFERENCE:  662nnn^B
                                 1)= S ^DIZ(662nnn,"B",$E(X,1,30),DA)=""
                                 2)= K ^DIZ(662nnn,"B",$E(X,1,30),DA)


662nnn,.02     SSN                     0;2 FREE TEXT

               INPUT TRANSFORM:  K:$L(X)>9!($L(X)<9)!'(X?9N) X
               HELP-PROMPT:      Answer must be 9 characters in length.

662nnn,.03     DOB                     0;3 DATE

               INPUT TRANSFORM:  S %DT="EX" D ^%DT S X=Y K:Y<1 X

662nnn,2       EMAIL                   2;0 Multiple #662nnn.02


662nnn.02,.01  EMAIL NAME              0;1 FREE TEXT (Multiply asked)

               INPUT TRANSFORM:  K:$L(X)>20!($L(X)<1) X
```

```
                    HELP-PROMPT:      Answer must be 1-20 characters in length.
                CROSS-REFERENCE:  662nnn.02^B
                                  1)= S
^DIZ(662nnn,DA(1),2,"B",$E(X,1,30),DA)=""
                                  2)= K ^DIZ(662nnn,DA(1),2,"B",$E(X,1,30),DA)
```

662nnn.02,1   EMAIL DOMAIN   0;2 FREE TEXT

```
                INPUT TRANSFORM:K:$L(X)>20!($L(X)<1) X
                HELP-PROMPT:    Answer must be 1-20 characters in length.
```

662nnn,3.1   AREA CODE   3;1 NUMBER

```
                INPUT TRANSFORM:  K:+X'=X!(X>999)!(X<100)!(X?.E1"."1N.N) X
                HELP-PROMPT:      Type a Number between 100 and 999, 0 Decimal
                                  Digits
```

662nnn,3.2   LOCAL NUMBER   3;2 FREE TEXT

```
                INPUT TRANSFORM:  K:$L(X)>8!($L(X)<8)!'(X?3N1"-"4N) X
                HELP-PROMPT:      Answer must be 8 characters in length.
```

662nnn,3.3   PHONE NUMBER   3;3 FREE TEXT

```
                INPUT TRANSFORM:  K:$L(X)>15!($L(X)<1) X
                HELP-PROMPT:      Answer must be 1-15 characters in length.
```

662nnn,3.4   OLD PHONE NUMBER   3;4 FREE TEXT

```
                INPUT TRANSFORM:  K:$L(X)>15!($L(X)<1) X
                HELP-PROMPT:      Answer must be 1-15 characters in length.
```

662nnn,4.1   DATE CREATED   4;1 DATE

```
                INPUT TRANSFORM:  S %DT="ESTXR" D ^%DT S X=Y K:Y<1 X
```

Entries in the Tutorial Test File

The following is a captioned printout, including record numbers, of the data in the *ZZINDIVIDUAL* test
file installed by routine A6AKIT.

```
ZZINDIVIDUAL LIST                    OCT  3,2000  10:43    PAGE 1
----------------------------------------------------------------------

NUMBER: 1                                NAME: FMPATIENT,ONE
  SSN: 000221111                         DOB: MAY 20,1945
EMAIL NAME: rc                           EMAIL DOMAIN: aaa.bbb.com
EMAIL NAME: rose                         EMAIL DOMAIN: xxx.yyy.com
  AREA CODE: 206                         LOCAL NUMBER: 555-4112

NUMBER: 2                                NAME: FMPATIENT,TWO
  SSN: 666443333                         DOB: FEB 5,1932
EMAIL NAME: jasmine.geiser              EMAIL DOMAIN: aaa.bbb.com
EMAIL NAME: jasmine                      EMAIL DOMAIN: xxx.yyy.com
  AREA CODE: 206                         LOCAL NUMBER: 555-8257

NUMBER: 3                                NAME: FMPATIENT,THREE
  SSN: 666776666                         DOB: JUL 14,1959
EMAIL NAME: herb                         EMAIL DOMAIN: abc.def.com
EMAIL NAME: hwaters                      EMAIL DOMAIN: abc.def.com
  AREA CODE: 406                         LOCAL NUMBER: 555-5834

NUMBER: 4                                NAME: FMPATIENT,FOUR
  SSN: 666223333                         DOB: NOV 23,1969
EMAIL NAME: mlake                        EMAIL DOMAIN: xxx.yyy.com
EMAIL NAME: marigold                     EMAIL DOMAIN: aaa.bbb.com
  AREA CODE: 503                         LOCAL NUMBER: 555-3612

NUMBER: 5                                NAME: FMPATIENT,FIVE
  SSN: 666654321                         DOB: FEB 27,1971
EMAIL NAME: dill                         EMAIL DOMAIN: abc.def.com
EMAIL NAME: dtide                        EMAIL DOMAIN: xxx.yyy.com
  AREA CODE: 505                         LOCAL NUMBER: 555-5939

NUMBER: 6                                NAME: FMPATIENT,SIX
  SSN: 666889999                         DOB: JUN 2,1955
EMAIL NAME: sagebrooks                   EMAIL DOMAIN: aaa.bbb.com
  AREA CODE: 307                         LOCAL NUMBER: 555-4680

NUMBER: 7                                NAME: FMPATIENT,SEVEN
  SSN: 666678901                         DOB: AUG 11,1948
EMAIL NAME: hazel.frost                  EMAIL DOMAIN: abc.def.com
EMAIL NAME: hazel                        EMAIL DOMAIN: abc.def.com
  AREA CODE: 503                         LOCAL NUMBER: 555-6874

NUMBER: 8                                NAME: FMPATIENT,EIGHT
  SSN: 666891234                         DOB: SEP 26,1973
EMAIL NAME: saffron                      EMAIL DOMAIN: aaa.bbb.com
EMAIL NAME: sripple                      EMAIL DOMAIN: abc.def.com
```

```
   AREA CODE: 503                    LOCAL NUMBER: 555-7555

NUMBER: 9                           NAME: FMPATIENT,NINE
   SSN: 666345678                   DOB: OCT 31,1919
EMAIL NAME: ginger                  EMAIL DOMAIN: aaa.bbb.com
   AREA CODE: 208                   LOCAL NUMBER: 555-8097

NUMBER: 10                          NAME: FMPATIENT,TEN
   SSN: 666432123                   DOB: APR 9,1938
EMAIL NAME: periwinkle              EMAIL DOMAIN: abc.def.com
EMAIL NAME: pwells                  EMAIL DOMAIN: xxx.yyy.com
   AREA CODE: 415                   LOCAL NUMBER: 555-5938

NUMBER: 11                          NAME: FMPATIENT,ELEVEN
   SSN: 666765432                   DOB: MAR 21,1970
EMAIL NAME: chervil                 EMAIL DOMAIN: xxx.yyy.com
EMAIL NAME: cpuddles                EMAIL DOMAIN: abc.def.com
   AREA CODE: 907                   LOCAL NUMBER: 555-7584

NUMBER: 12                          NAME: FMPATIENT,TWELVE
   SSN: 666996666                   DOB: MAY 22,1947
EMAIL NAME: basil                   EMAIL DOMAIN: abc.def.com
EMAIL NAME: bsnow                   EMAIL DOMAIN: xxx.yyy.com
   AREA CODE: 503                   LOCAL NUMBER: 555-9573

NUMBER: 13                          NAME: FMPATIENT,THIRTEEN
   SSN: 666567890                   DOB: MAY 20,1945
EMAIL NAME: holly                   EMAIL DOMAIN: abc.def.com
EMAIL NAME: hollyrivers             EMAIL DOMAIN: xxx.yyy.com
   AREA CODE: 509                   LOCAL NUMBER: 555-7969
```

Glossary

A (#*nn*)	This is the name and number of the key. Key definitions are stored in the KEY file (#.31), a FileMan file, and the key number (#nn) is the internal entry number of the key in that file. Like index numbers, key numbers vary from system to system.

<div align="center">

Back to tutorial

</div>

ACTION-TYPE CROSS-REFERENCE	An Action-type cross-reference is a cross-reference with set and kill logic that performs some action other than building an index. Most MUMPS cross-references are Action-type cross-references. An Action-type cross-reference must have a name that starts with the letter "A".

<div align="center">

Back to tutorial

</div>

ACTIVITY	Activity is a set of flags that that controls whether FileMan fires a cross-reference during an installation and/or a re-cross-referencing operation. The possible flags are:

      I   Installing an entry at a site.
      R   Re-cross-referencing this index.

Activity can contain either "I", "R" or both flags together.

FileMan automatically fires cross-references during an edit, regardless of Activity, though you can control whether a cross-reference is fired by entering Set and Kill Conditions.

Also, if you explicitly select a cross-reference in an EN^DIK, EN1^DIK, or ENALL^DIK call, or in the UTILITY FUNCTIONS/RE-INDEX FILE option on the VA FileMan menu, that cross-reference will be fired whether or not its Activity contains an "R".

<div align="center">

Back to tutorial

</div>

BACKWARD AND FORWARD COLLATION	This is the direction FileMan's lookup utilities should $ORDER through this subscript when entries are returned or displayed to the user. If for example, you have a compound index on a Date of Birth field and a Name field, and you specify a COLLATION of backwards on the Date of Birth value, the Lister and the Finder will return entries in reverse-date order. Likewise, question mark (?) help and partial matches in interactive ^DIC lookups will display entries in reverse-date order.

<div align="center">

Back to tutorial

</div>

"C" INDEX	This is the name of the index, which for Regular indexes corresponds to the index subscript.

COLLATION	Collation is where you can specify the forwards or backwards direction in which VA FileMan's lookup utilities loop through a subscript in an index when entries are returned or displayed to the user. This is especially useful for dates. Developers can store dates in their natural internal VA FileMan date format, and still display entries in the date index in reverse date order.

COMPUTED CODE	The computed code sets the variable X, and makes use of the X(order#) array. In this case X(1) refers to the EMAIL NAME, the cross-reference value we defined with order number 1, and X(2) refers to EMAIL ADDRESS, the cross-reference value we defined with order number 2. Our computed expression sets X to email_name@email_domain, and converts the result to uppercase. This will be our third cross-reference value (order number 3), which will be subscript 1 (the only data subscript) in our index.

CROSS-REFERENCE VALUE: FIELD PROPERTY	If this cross-reference value is a field, answer with the field number or name. Enter ?? to see a list of selectable fields.

^DDS: SCREENMAN	This ScreenMan API allows you to use a screen-mode interface to edit an entry in a file.

^DIE	This classic VA FileMan API allows you to edit an existing record in a file.

^DIZ GLOBAL ROOT	DIZ Global Root ^DIZ(662100, is the global root of the file that stores the datA. When you create a file, FileMan prompts you for an internal global reference. ^DIZ(file#, is the default. This global root is used for internal purposes such as test files or files used locally.

E (#*nnn*)	The key's uniqueness index is the E index, stored as entry #nnn in the INDEX file (#.11). (Internally, a uniqueness index is a new-style index that is pointed to by an entry in the KEY file [#.31].)

ENTER OR EDIT FILE ENTRIES	This VA FileMan option allows you to add and edit an entry in a file.

EXECUTION   Field - This indicates that the cross-reference logic is executed
                               immediately after each and every field that makes up the cross-
                               reference is edited. Most simple (single-field) cross-references
                               should have Field execution.

                               Record - This indicates that the cross-reference logic is executed
                               only after all fields in the record have been edited. Most compound
                               (multi-field) cross-references should have Record execution.

FILE #*662nnn*   This is the number of the file on which the keys are defined.

FILE, FIELD   This section lists the fields that make up the key. Key A contains
                               two fields: NAME (#.01), which is sequence number 1 (subscript 1
                               in the uniqueness index), and SSN (#.02), which is sequence
                               number 2 (subscript 2 in the uniqueness index).

FILE^DICN   This classic VA FileMan API adds a new entry to a file.

FILE^DIE: FILER   The Filer puts validated data that is in internal FileMan format into
                               the database; or validates data that is in external (user-provided)
                               format, converts it into FileMan internal format, and files valid data
                               into the database.

                               The following is a breakdown of parameters passed to the Filer in
                               the call:

```
>D FILE^DIE("E","ZZFDA","ZZMSG")
```

                               1. "E" This is the flag parameter. "E" indicates that the
                                  FDA contains the external form of the datA. The Filer
                                  validates the value and converts it to internal form
                                  before it files it in the database.
                               2. "ZZFDA" This is the fda_root parameter and equals
                                  the root of the FDA that contains the data to file.
                               3. "ZZMSG" This is the msg_root parameter and equals
                                  the root of the array in which the Filer returns error
                                  messages.

FILE^DIE: FILER	This DBS server API allows you to put validated data that is internal format into a file; or to validate data that is in external (user-provided) format, convert it to internal FileMan format, and file the valid data into the database.

Back to tutorial

FILEMAN DATA ARRAY (FDA)	Data is passed to and from the DBS as values in the FileMan Data Array (FDA). The FDA contains the file, internal entry numbers, and field information in its subscripting scheme. The format of the FDA is:

        FDA_ROOT(FILE#,"IENS",FIELD#)="VALUE"

In this case, we are setting the FDA for field 4.1, the DATE CREATED field in file *662*100, to "NOW", and passing this FDA to FILE^DIE, the Filer.

Back to tutorial

FORWARD AND BACKWARD COLLATION	This is the direction FileMan's lookup utilities should $ORDER through this subscript when entries are returned or displayed to the user. If for example, you have a compound index on a Date of Birth field and a Name field, and you specify a COLLATION of backwards on the Date of Birth value, the Lister and the Finder will return entries in reverse-date order. Likewise, question mark (?) help and partial matches in interactive ^DIC lookups will display entries in reverse-date order.

Back to tutorial

INDEX DETAILS	If a Uniqueness Index already exists for the key, you press return at the Index Details… field to display the properties of that index.

Back to tutorial

INDEX NAME	The name of the index is used as a subscript in the index. For example, your index will be stored in global ^DIZ(*662*100,"C"). The index name is the subscript "C". (^DIZ(*662*100, is the global root of the file).

Back to tutorial

INDEX NAME ("E")	This is the name of the uniqueness index, which corresponds to the subscript in the index.

Back to tutorial

INDEXES ONLY DATA DICTIONARY FORMAT	The Indexes Only format shows the Traditional and New-Style cross-references that are defined on a file.

Back to tutorial

INPUT VARIABLES TO IX^DIC LOOKUP	DIC Set to the global root of the file. DIC(0) - Set to the lookup parameters:

        Q   Question erroneous input with two question marks (??).

        E   Echo information (dialog with the user is allowed).

        A   Ask the user for the lookup value; if erroneous, ask again.

        Z   Return the zero node of the selected entry in Y(0), and the   external form of the .01 field.

    D   Set to the name of the index to use in the lookup.

Back to tutorial

INPUT VARIABLES TO IX1^DIK	DIK - This variable is set to the global root of the file or subfile that contains the record to be reindexed. DA - This variable is set to the IEN of the record to be reindexed.

Back to tutorial

INTERNAL ENTRY NUMBER	This is the internal entry number of the indexed record.

Back to tutorial

IR (INSTALLATION/REINDEXING)	This is the Activity designation of the index. New-Style cross-references can have an Activity of "R" and/or "I" to allow you to control whether the cross-reference should be fired during Reindexing and a KIDS Installation. If you call the entry points in ^DIK for reindexing, or if you select the Re-Index File option on VA FileMan's Utility Functions submenu, only those new-style cross-references that contain an "R" in Activity will be fired. During a KIDS installation, when a file is reindexed, a new-style cross-reference is executed only if its Activity contains an "I".

Back to tutorial

IX^DIC LOOKUP	This entry point is similar to ^DIC, except for the way it uses cross-references to perform lookup. IX^DIC starts with the cross-reference you specify, or uses only the cross-reference you specify. We are using the IX^DIC call so that we can specify the "C" index for the lookup.

Back to tutorial

IX1^DIK LOOKUP	This entry point executes the set logic of all cross-references for a single entry in the file.

Back to tutorial

KEY	A key is a set of one or more fields in a file that together uniquely identifies a record in that file. If you define a key, FileMan V.22.0 automatically enforces the integrity of that key, which means that:

    1. No key field is null.
    2. The key (that is, the combination of fields in a key) is unique for all records in the file.

FileMan stores the definition of keys in the KEY file (#.31), which has the global root ^DD("KEY").

<div align="center">

[Back to tutorial](#)

</div>

KEY DEFINITION OPTION	The Key Definition option is locate on FileMan's Utility Functions menu and allows you to:

    1. Create a key
    2. Edit a key
    3. Delete a key
    4. Verify the integrity of a key

<div align="center">

[Back to tutorial](#)

</div>

KEY FIELDS	In this section of the form, you can view and edit the fields that make up the key.

<div align="center">

[Back to tutorial](#)

</div>

$$KEYVAL^DIE: KEY VALIDATOR	The Key Validator extrinsic function verifies that new values contained in a FileMan Data Array (FDA) do not produce an invalid key. All keys in which any field in the FDA participates are checked. If the value for a field in a key being checked is not present in the FDA, the value used to verify the key is obtained from the previously filed datA.

<div align="center">

[Back to tutorial](#)

</div>

LIST FILE ATTRIBUTES	This option is used to print data dictionary listings for a given file. This listing is useful for programmers, analysts, and others interested in data base structures.

<div align="center">

[Back to tutorial](#)

</div>

LOOKUP AND SORTING INDEX	A Lookup and Sorting index is automatically used by FileMan lookup APIs (^DIC, FIND^DIC, and $$FIND^DIC) and must have a name that starts with "B" or a letter that alphabetically follows "B".

A Sorting Only index is used by FileMan's lookup APIs only if it is explicitly specified in the input parameters to the API. A Sorting Only index must have a name that starts with the letter "A".

Both Lookup & Sorting indexes and Sorting Only indexes are available for use by FileMan's Sort and Print modules (EN1^DIP).

LOOKUP AND SORTING INDEX	A Lookup and Sorting index is automatically used by FileMan lookup APIs (^DIC, FIND^DIC, and $$FIND^DIC) and must have a name that starts with "B" or a letter that alphabetically follows "B". A Sorting Only index is used by FileMan's lookup APIs only if it is explicitly specified in the input parameters to the API. A Sorting Only index must have a name that starts with the letter "A". Both Lookup & Sorting indexes and Sorting Only indexes are available for use by FileMan's Sort and Print modules (EN1^DIP).

"M" FLAG (MULTIPLE-INDEX LOOKUP ALLOWED)	The "M" flag input variable will allow a multiple lookup on all of the file's cross-references from B on to the end of the alphabet.

MAKE KEY MATCH UNIQUENESS INDEX	If you select this option, FileMan will make the key fields match the fields defined in the uniqueness index.

MAKE UNIQUENESS INDEX MATCH KEY	If you select this option, FileMan will modify the uniqueness index so that it matches the fields in the key (the fields in the Key Fields section of the form).

MANDATORY/REQUIRED FIELD CHECK	The Mandatory/Required Field Check option checks that all fields that are key fields or designated as required contain datA. It can check one, a series, or all entries in a file.

MUMPS CROSS-REFERENCES	MUMPS is one of the two types of new-style cross-references you can create. (The other is Regular.) In contrast to Regular cross-references, you define the set and kill logic of MUMPS cross-references. The logic of a MUMPS cross-reference is executed whenever a field in the cross-reference is edited, and usually performs some action other than just maintaining an index.

NAME FIELD ("FMPATIENT,EIGHT")	This is the data in the NAME field (#.01), which is sequence number 1 in the key.

NEW-STYLE CROSS-REFERENCE	A new-style cross reference can be composed of one or more fields. Its definition is stored in the INDEX file (#.11), which has the global root ^DD("IX").
	New-style cross-references that are composed of a single field are called simple cross-references, and by default have field-level execution, which means that the cross-reference logic is executed immediately after the field is edited.
	New-style cross-references that are composed of more than one field are called compound cross-references, and by default have record-level execution. Record-level execution means that the cross-reference logic is executed only after an entire record is edited, after all the fields in the cross-reference have been edited.
	There are two types of New-Style cross-references: Regular and MUMPS.

<p align="center">Back to tutorial</p>

"#nnn" (RECORD'S INTERNAL ENTRY NUMBER)	The definitions of new-style indexes are stored in the INDEX file (#.11), a file used internally by VA FileMan. The #nnn is the IEN of the record in the INDEX file that contains this index definition.
	The index number varies from system to system, just as the internal entry number of records in the Input and Print Template files vary from system to system.

<p align="center">Back to tutorial</p>

OUTPUT VARIABLE Y FOR THE IX^DIC CALL	Y = N^S	N is the Internal Entry Number of the selected entry in the file, and S is the value of the .01 field of that entry.
	Y(0)	This variable is set only if DIC(0) contains a Z. When the variable is set, it is equal to the entire zero node of the entry that was selected.
	Y(0,0)	This variable is set only if DIC(0) contains a Z. When the variable is set, it is equal to the external form of the .01 field of the selected entry.

<p align="center">Back to tutorial</p>

PHONE NUMBER AND OLD PHONE NUMBER	These two fields were updated via the set logic of our MUMPS cross-reference. That logic called the Filer to update the numbers based on our edits to the AREA CODE and LOCAL NUMBER fields.

<p align="center">Back to tutorial</p>

PRIMARY KEY	This indicates that the key is a primary key, rather than a secondary key.

<p align="center">Back to tutorial</p>

PRIORITY   Priority indicates whether the key is the primary key of the file or a
                                  secondary key. By default, the first key you create for a file or
                                  subfile is designated the primary key.

                                  Back to tutorial

RECORD CREATION   Record creation occurs when a new record is added to a file or
                                  subfile.

                                  When a record is first created, FileMan sets the .01 field for that
                                  record and immediately fires the set logic for cross-references on
                                  the .01 field. Note that if additional fields are edited after the .01 is
                                  set, cross-references on those fields may also be executed,
                                  including those that may have already been fired when the .01 field
                                  was initially set.

                                  Back to tutorial

RECORD DELETION   Record deletion occurs when an entire record is removed from a
                                  file or subfile.

                                  Back to tutorial

RECORD EDIT   Record edit occurs when one or more field values for a record are
                                  changed.

                                  Back to tutorial

RECORD LEVEL EXECUTION   This is the Execution designation of the index. Record-level
                                  execution means that FileMan will execute the cross-reference
                                  logic only once after all the fields in a record are edited, typically at
                                  the end of an editing session. This is in contrast to Field-level
                                  execution, where FileMan executes the cross-reference logic after
                                  each and every field in the cross-reference is edited.

                                  For example, if both the DOB and SSN fields are contained in a
                                  single input template, but the index is defined as having Field-level
                                  execution, the cross-reference logic would be fired twice: once
                                  when the DOB field is edited, and then again when SSN is edited.
                                  Record-level execution means that even if the user edits both the
                                  DOB and SSN fields for a record, FileMan will execute the cross-
                                  reference logic only once at the end of the editing session.

                                  In most cases, simple indexes should have Field-level execution,
                                  while compound indexes should have Record-level execution.

                                  Back to tutorial

RE-EDIT THE KEY   If you select this option, FileMan will take you back into the
                                  ScreenMan form where you can edit the properties of the key.

                                  Back to tutorial

REGULAR CROSS-REFERENCE (INDEX)	Regular is one of the two types of new-style cross-references you can create. (The other is MUMPS.) In a regular cross-reference, data is stored as subscript(s) in an index. The index can be used for sorting the data and for looking up entries in a file based on the indexed data.

Back to tutorial

REGULAR INDEX (CROSS-REFERENCE)	Regular is one of the two types of new-style cross-references you can create. (The other is MUMPS.) In a regular cross-reference, data is stored as subscript(s) in an index. The index can be used for sorting the data and for looking up entries in a file based on the indexed data.

Back to tutorial

SCREENMAN	ScreenMan is VA FileMan's screen-oriented data entry tool. It is an alternative to the Scrolling Mode approach. With ScreenMan, data is entered in forms. Each form field occupies a fixed position on the screen (instead of scrolling off!). You can see many data fields at once, and use simple key combinations to edit data and move from field to field on a screen. You can also move from one screen to another like turning through the pages of a book.

Back to tutorial

SCREENMAN FORM: FILE PROPERTY	This is the file on which the index will physically reside. In this case, though the fields in our index come from the EMAIL multiple (subfile #662100.02), the index will reside at the top level, file #662100. By definition, whole-file indexes reside at a file level above the level in which the fields are defined.

Back to tutorial

SCREENMAN FORM: ORDER COLUMN	FileMan evaluates cross-reference values by order of "Order Number" and places each value in the X(order#) array. The set and kill logic, for example, can use X(2) to refer to the cross-reference value with order number 2.

Back to tutorial

SCREENMAN FORM: ROOT FILE PROPERTY	This is the file or subfile on which the cross-reference is defined. It corresponds to our responses to the previous Step 3, the "File:" and "Subfile:" prompts. Field-type cross-reference values must be fields from this Root File -- in this case, the EMAIL multiple.

Back to tutorial

SCREENMAN FORM: ROOT TYPE PROPERTY	This can be either INDEX FILE or WHOLE FILE. If we're defining a WHOLE FILE index, the fields in the index are defined in a subfile, but the index itself physically resides at a higher level. In our case, the fields in the index will come from the EMAIL multiple, but the index will reside at the top level of the file. This enables us to lookup entries in the ZZINDIVIDUAL file given an email address.
	If we're not defining a whole-file index, the Root Type is INDEX FILE and File and Root File are equal. The index resides at the same file level as the fields in the index. This enables us to look up entries in the EMAIL multiple for a particular ZZINDIVIDUAL file entry.

<div align="center">

Back to tutorial

</div>

SET AND KILL LOGIC	FileMan executes the logic of a cross-reference when the values of any fields that make up the cross-reference are edited. FileMan first executes the kill logic, and then the set logiC.
	In the set and kill logic, you can assume that the DA array describes the entry number of the record to be cross-referenced. The X(order#) array contains cross-reference values after the transform for storage is applied, but before the truncation to the maximum length. The variable X equals X(order#) of the lowest order number.
	The X1(order#) array contains the old cross-reference values, and the X2(order#) array contains the new cross-reference values. If a record is being added, and there is an X1(order#) array element that corresponds to the .01 field, it is set to null. When a record is deleted, all X2(order#) array elements are null.

<div align="center">

Back to tutorial

</div>

Back to tutorial

SIMPLE AND COMPOUND CROSS-REFERENCES	Simple Cross-Reference

Simple Cross-Reference
- A cross-reference that is composed of one field.
- By default, has field-level execution; where the cross-reference logic is executed immediately after the field is edited.

Compound Cross-Reference
- A cross-reference that is composed of more than one field.
- By default, has record-level execution, where the cross-reference logic is executed only after the entire record is edited.

Back to tutorial

SORTING ONLY INDEX

SORTING ONLY - The index name starts with "A". Calls to Classic FileMan lookup (^DIC) or the Finder (FIND^DIC or $$FIND1^DIC) will not use this index unless it is specified in the input parameters. The index will be available for use by the FileMan Sort and Print (EN1^DIP).

Back to tutorial

SSN FIELD (777889999)

This is the data in the SSN field (#.02), which is sequence number 2 in the key.

Back to tutorial

STANDARD CAPTIONED OUTPUT

The standard captioned output in the Inquire to File Entries option prints out all fields that contain data for each entry in your report. It is the fastest way to choose which fields to print.

Back to tutorial

STANDARD DATA DICTIONARY

The most complete information about a file is obtained by using the Standard data dictionary format, which is the default for the List File Attributes option. In addition to detailed information about every field in the file, the Standard data dictionary format gives the file access, identifiers, cross-references, other files pointing to the file, files pointed to by the file, and any templates (including forms and blocks) associated with the file.

Back to tutorial

SUBSCR (SUBSCRIPT) COLUMN

If this cross-reference value is used as a subscript in an index, enter the subscript position number. The first subscript to the right of the index name is subscript number 1.

Back to tutorial

TRADITIONAL CROSS-REFERENCE	A traditional cross-reference is defined on a single field, and its definition is stored under ^DD(file#,field#,1). In general, the logic for a traditional cross-reference is executed when the field is edited.

There are seven types of Traditional cross-references: Regular, MUMPS, Trigger, Bulletin, Soundex, KWIC, and Mnemonic.

[Back to tutorial](#) |
| TRANSFORM FOR DISPLAY | FileMan uses this transform only during lookup.

The contents of this transform should be M code that sets the variable X to a new value. X is the only variable that is guaranteed to be defined and is equal to the value of the subscript from the index.

TRANSFORM FOR DISPLAY should be set only for an index that has been transformed using the code in the TRANSFORM FOR STORAGE prior to storing the value in the index.

The code should take the internal value from the index subscript X, and convert it back to a format that can be displayed to an end user. During lookup, if a match or matches are made to a lookup value that was transformed using the TRANSFORM FOR LOOKUP code on this index, then FileMan will execute the TRANSFORM FOR DISPLAY code before displaying the index value(s) to the end user.

[Back to tutorial](#) |
| TRANSFORM FOR LOOKUP | FileMan uses this transform only during lookup.

The contents of this transform should be M code that sets the variable X to a new value. X is the only input variable that is guaranteed to be defined and is equal to the lookup value entered by the user.

During lookup, if the lookup value is not found in the index, FileMan will execute the TRANSFORM FOR LOOKUP code to transform the lookup value X. It will then search this index looking for a match to the transformed lookup value.

[Back to tutorial](#) |
| TRANSFORM FOR STORAGE | FileMan uses this transform only when setting or killing an entry in the index.

The contents of this transform should be M code that sets the variable X to a new value. X is the only input variable that is guaranteed to be defined and is equal to the internal value of the field.

TRANSFORM FOR STORAGE can be used on field-type cross- |

reference values to transform the internal value of the field before it is stored as a subscript in the index.

If a match is made on this index during a lookup, then in order to properly display the resulting index value to the user, the developer may need to enter code into the TRANSFORM FOR DISPLAY field to transform the index value back to a displayable format

[Back to tutorial](#)

TUTORIAL TEST FILE (I.E., *ZZINDIVIDUAL*)

The exercises in this tutorial make use of a simple test file that contains some sample datA. The file is installed as part of Lesson 1, Exercise 1.1, and each user taking the tutorial works with his or her own copy of the tutorial test file. Each user's copy of the test file has a unique file name and number.

In this tutorial, all references to this test file will be of file name: *ZZINDIVIDUAL*, file number: *#662nnn*, and global root: *^DIZ(662nnn,*.

[Back to tutorial](#)

UNIQUENESS INDEX

If a Uniqueness Index already existed for this key, you would see information about that index here.

As you will see in Lesson 10, you can also select an already existing new-style index as the Uniqueness Index for the key, and have FileMan make all the fields in that index the fields in the key. You can do this in lieu of selecting fields in the Key Fields section of the form.

[Back to tutorial](#)

UPDATE^DIE: UPDATER

This DBS server API adds a new entry to a file.

[Back to tutorial](#)

VAL^DIE: VALIDATOR

The Validator takes the external form of a value (the user input) and determines if that value is valid (i.e., if that value can be put into the VA FileMan database).

[Back to tutorial](#)

VALS^DIE: FIELDS VALIDATOR

The Fields Validator validates data for a group of fields and converts valid data to internal VA FileMan format. It is intended for use with a set of fields that comprise a logical record; fields from more than one file can be validated by a single call. By default, the integrity of any keys affected by the new values is checked.

[Back to tutorial](#)

VERIFY FIELDS

The Verify Fields option uses a field's definition to verify the data stored in a file. After invoking this option, you can ask to verify all existing values of a particular field by entering its label at the "VERIFY WHICH FIELD:" prompt; or you can ask that all fields at a given file level be verified by entering ALL at the prompt.

If more than one discrepancy is found between the current definition and the data on file, you will be asked if you want to save the list of those entries containing the inconsistent data in a template. Later you would be able to "SORT BY:" the entries in this template to display or edit them.

Back to tutorial

WHOLE KILL

This code, like the set and kill logic is, automatically generated for you for regular indexes. This is the code that FileMan executes when requested to delete an entire index from a file.

Back to tutorial

WHOLE-FILE INDEX

A whole-file index is stored at a file level above the level in which the fields are defined. Whole-file indexes allow you to lookup records in a file based on field values within a subfile.

For example, a whole-file index containing fields from the EMAIL multiple can be stored at the top level of the file, and allows you to lookup records in the *ZZINDIVIDUAL* file based on field values within that multiple.

Back to tutorial

X(1), X(2)

These lines shows information about the cross-reference values. Every cross-reference value must have a unique order number. X(1) corresponds to the cross-reference value with order number 1; X(2) corresponds to the cross-reference value with order number 2.

X(1) in this case is the value of the DOB field (file #*662*100, field #.03), which is stored in the first subscript (Subscr 1) after "C" in the index. The forwards designation indicates that lookups will traverse this subscript in a forwards direction. (This is the Collation property of a subscript, and will be discussed further in Lesson 3).

The subscript number for a cross-reference value need not equal the order number. In fact there may be times when you'd create a cross-reference value that has no subscript number. In that case, even though the value isn't stored in the index, you can still refer to it via X(order#) in the cross-reference logiC. (You'll see an example of this in Lesson 5.)

Back to tutorial

X1(3) ARRAY ELEMENT	X1(**3**) - the old value of the cross-reference value with order number 3, a computed value. This is the old AREA CODE field, or the string "null" if the old AREA CODE field is null.

[Back to tutorial](#)

X1(4) ARRAY ELEMENT	X1(**4**) - the old value of the cross-reference value with order number 4, a computed value. This is the old LOCAL NUMBER field, or the string "null" if the old LOCAL NUMBER field is null.

[Back to tutorial](#)

X2(3) ARRAY ELEMENT	X2(**3**) - the new value of the cross-reference value with order number 3, a computed value. This is the new AREA CODE field, or the string "null" if the new AREA CODE field is null.

[Back to tutorial](#)

X2(4) ARRAY ELEMENT	X2(**4**) - the new value of the cross-reference value with order number 4, a computed value. This is the new value of the LOCAL NUMBER field, or the string "null" if the new LOCAL NUMBER field is null.

[Back to tutorial](#)

ZZINDIVIDUAL (I.E., THE TUTORIAL TEST FILE)	The exercises in this tutorial make use of a simple test file that contains some sample datA. The file is installed as part of Lesson 1, Exercise 1.1, and each user taking the tutorial works with his or her own copy of the tutorial test file. Each user's copy of the test file has a unique file name and number. In this tutorial, all references to this test file will be of file name: *ZZINDIVIDUAL*, file number: *#662nnn*, and global root: *^DIZ(662nnn,*.

[Back to tutorial](#)

Index

F

G

H

I

K

L