M

VA M Programming

Intermediate

Lesson 5

Creating and Maintaining Global Files

Veterans Health Administration
Office of Information
OI National Training and Education Office

Table of Contents

Acknowledgements

Developing a comprehensive training package for the ***M*** computer programming language as it is used in VA was not an easy task. Much gratitude goes to the members of the M Programming Language Training Development Team who spent countless hours developing this training program. Sincere appreciation to:

Debbie Channell
Tuscaloosa OI Field Office, Customer Services

Harris H. Lloyd
Central Arkansas Veterans Health Care System

Bob Lushene
Hines OI Field Office, Technical Services

Guide to Symbols Used

<ENTER>	Press the Enter or Return key
<TAB>	Press the Tab key
<space>	One space
	What follows is from part of the VA Programming Standards and Conventions.
	This indicates an OPTIONAL reading reference. Additional information on this topic may be found in the references listed below.

Optional Reading References

Lewkowicz, John. *The Complete M: Introduction and Reference Manual for the M Programming Language*. 1989.

Walters, Richard. *M Programming – A Comprehensive Guide*. 1997.

The optional reading references are available through:

M Technology Association
1738 Elton Road, Suite 205
Silver Spring, MD 20903
301-431-4070

References

The VA M Programming Standards and Conventions shown were taken from "VA DHCP Programming Standards and Conventions" document, prepared March 19, 1991, modified in 1995 and approved January 22, 1996.
http://vaww.vista.med.va.gov/Policies/sacc.htm.

Document Update Note

January 2007 – To meet current standards on personal data privacy and accessibility, this document was updated. It was edited to include document metadata and edited to de-identify any personal data. Only the information in this document is affected. This should have no effect on the M routines.

Objectives

The objective of this lesson is to prepare you to construct a complete system using the following commands and techniques:

- Local variables and global variables
- M files: creating and accessing nodes
- String and numeric subscripts
- $ORDER function
- Canonic order
- $DATA function
- Five types of indirection
- Precautions in the use of indirection
- Recognizing the limitations of the sample system

Storing data permanently

Local variables versus global variables

Up to this point, we have been assigning values to variables called ***local variables***. When we are finished with a session, these variables and their associated values are erased (i.e., not saved permanently). In addition, even during the session, the values of the variables are only available to our terminal in our workspace.

However, most applications involve data that is stored permanently (most often on disks), either in files or in databases. For example, if we are maintaining an inventory of office supplies, we want to create a file that retains the inventory data: description, quantity, vendor, etc. In addition, we want to be able to share this data with users at other locations.

In M, we designate that the value of a variable is to be stored permanently by preceding the variable name with a caret, often called an up-arrow (^). These then become ***global variables*** and their values are stored and become available to multiple users on different terminals.

Global Variables

```
^AGE          ^XDOLLAR        ^NAME       ^TOTAL1
```

In the previous lesson, we studied the concept of arrays where a named location can hold multiple values at one time. We can precede an array name with an ^ and create a global array which is stored permanently and becomes available to multiple users. These global arrays are the basis of M files.

Global Arrays

```
^VECARR(1)      ^PATIENT(RECNR)      ^LABELS(NAME)
```

Global names must be consistent with the assigned namespace for the application. Application programs must not KILL entire (i.e., unsubscripted) globals.

VA Programming Standards and Conventions
Global names must be consistent with the assigned namespace.
Application programs must not KILL entire (unsubscripted) globals.

More About M Files

In M, a global array is the structure we use for files. You will find that the words array and file are often used synonymously.

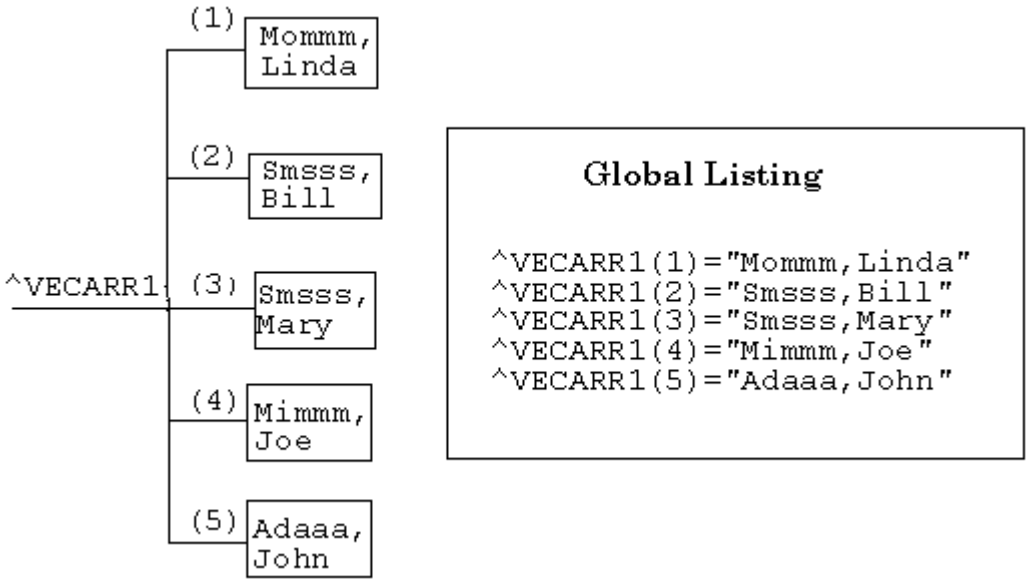
M arrays can have subscripts that are numeric or string values. The restriction is that the sum of the lengths of all evaluated subscripts, plus the number of subscripts, plus the length of the global name must not exceed 127 characters. The length of an individual subscript cannot exceed 63 characters. This restriction is not a physical restriction of the language, but a guideline for portability. An M system is portable if you can write the code on one type of hardware and operating system and move the system to another type of hardware and operating system with little or no modification.

Another feature of M arrays is that, unlike some other computer languages, you don't have to define ahead of time how big your array will be. In languages where arrays must be predefined, we refer to this as a **dense array**: every array element is assigned a storage location even if it is never assigned a value. Since M arrays are not predefined, we refer to them as **sparse arrays** where the only elements that are assigned storage space are the ones that actually have values.

VA Programming Standards and Conventions
A full global reference (the sum of the lengths of all evaluated subscripts, plus the number of subscripts, plus the length of the global name) must not exceed 127 characters. The value of a single subscript must not exceed 63 characters.

The most common way in which an array is represented in M is through the use of the inverted tree diagram. This diagram shows the relationship among data in a file just as a family tree shows the relationship of members of a family.

Inverted Tree Diagram
Using ^VECARR



The name of the array is shown at the left of the diagram. This is also called the ***root***. From the root we have ***branches***. At the ending point of each branch we have a ***node***. Each node is comprised of two parts: the name of the node (name of the array and the subscript) and the content of the node (shown inside the larger box). The content of a node can contain a maximum of 255 characters. However, in the VA Standard, the maximum number of characters a node can contain is 245.

Creating Nodes in a Global File

To create a node, you can use the SET command as you would with a local variable. You cannot, however, have a global variable as the argument of a READ command. If the value of a global variable is to come from the keyboard, you must first READ the value into a local variable and then SET the global variable equal to the local variable as shown: (*Note: What the user typed is double-underlined)*

M Code Sample

```
VECS55  ;
        FOR RECNR=1:1:5 DO ENTER
        QUIT
ENTER   ;
        READ !,"Enter name: ",NAME:DTIME
        IF '$T!(NAME["^") GOTO ENEXIT
        SET ^VECARR1(RECNR)=NAME
ENEXIT  ;
        QUIT

D ^VECS55

Enter name: Mommm,Linda
Enter name: Smsss,Bill
Enter name: Smsss,Mary
Enter name: Mimmm,Joe
Enter name: Adaaa,John

D ^%G

Global ^VECARR1

^VECARR1
^VECARR1(1) = Mommm,Linda
^VECARR1(2) = Smsss,Bill
^VECARR1(3) = Smsss,Mary
^VECARR1(4) = Mimmm,Joe
^VECARR1(5) = Adaaa,John

Global ^<ENTER>
```

The sample program above, prompts the user to enter a name. The name is then set into the global VECARR1. The execution of the program is shown and the contents of the global using the global lister; %G, is also shown.

Using the $ORDER Function to Access Nodes

In Lesson 4, we searched through arrays using the FOR command and subscripts that were consecutive record numbers. However, with string subscripts there is often no predictable order with which we can use the FOR command. Instead, we use an intrinsic function called $ORDER to access nodes in an array.

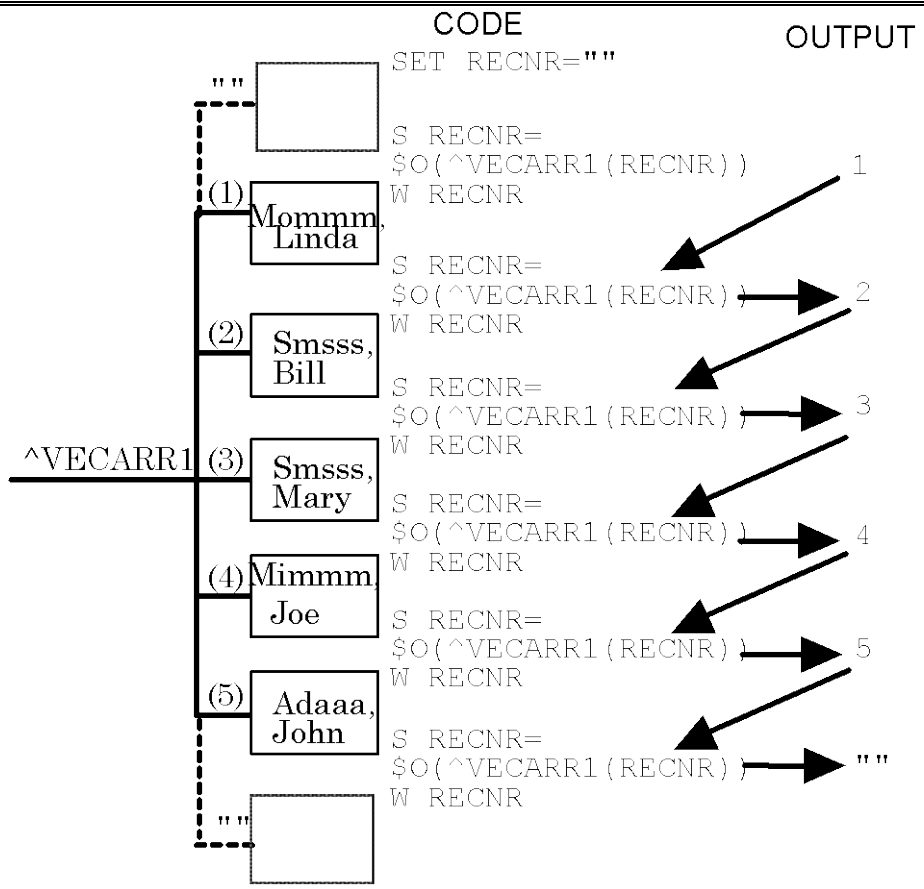
The $ORDER function moves through an array one node at a time starting at the node you specify. In order to access nodes, you will set up a variable to hold each successive subscript. Once $ORDER has accessed a node, its subscript will be in this variable which can then be used to access the contents of the node.

To tell M that you want to start at the beginning of an array, you will SET the subscript variable equal to null ("") before you issue the first $ORDER. When $ORDER has accessed the last node, it will set the subscript variable back to null ("").

Shown below is a typical $ORDER loop designed to access all the nodes in the ^VECARR1 file:

M Code Sample

```
VECS56    ;
          S RECNR=""
          F  S RECNR=$ORDER(^VECARR1(RECNR)) Q:RECNR=""  D PR
          QUIT
          .
          .
PR        ;
          W !,"The content for record number ",RECNR
          W " is ",^VECARR1(RECNR)
          Q
```



The most important thing to remember when interpreting the $ORDER is that M will work first with the expression to the right of the equal sign on the SET. With the $ORDER that means M will use the current value of the variable RECNR to get the next subscript in order. Then M will set that new value into RECNR.

Another M function that has been used in the past but is considered obsolete is $NEXT. $NEXT works in the same way as $ORDER but uses a starting value of -1 and returns a value of -1 when it reaches the last node of the current subscript level.

VA Programming Standards and Conventions
$NEXT shall not be used. Packages must remove all occurrences of $NEXT by December 31, 1992.

Sample System with String Subscripts

The following is a sample system that either loads or prints a file that has string subscripts. Notice that the words are not entered in alphabetical order. When all the data have been entered the user types <ENTER> at the enter word prompt to end the program. *(Note: What the user typed is double-underlined.)*

```
>D ^VECS59

Dictionary System

Option   Function

1        Load dictionary

2       Print dictionary

Enter option number: 1

Enter word: yore
Enter short definition: time long past

Enter word: finis
Enter short definition: the end

Enter word: mart
Enter short definition: market

Enter word: abode
Enter short definition: house or dwelling

Enter word: grow
Enter short definition: to expand

Enter word: neaten
Enter short definition: to put in order

Enter word: chop
Enter short definition: to cut

Enter word:<ENTER>
>
```

The M Code sample below shows how the data that was just entered is actually stored in the global using string subscripts.

M Code Sample

```
TRN,STU>D ^%G

Global ^VECDIC
        VECDIC
^VECDIC
^VECDIC("abode") = house or dwelling
^VECDIC("chop") = to cut
^VECDIC("finis") = the end
^VECDIC("grow") = to expand
^VECDIC("mart") = market
^VECDIC("neaten") = to put in order
^VECDIC("yore") = time long past
Global ^
```

This page shows the same system that was shown on the preceding page, except in this case the user chooses to print the dictionary entries. Notice that the words come out in alphabetical order even though they were not entered that way. *(Note: What the user typed is double-underlined.)*

```
>D ^VECS59

Dictionary System

Option   Function

1        Load dictionary

2       Print dictionary

Enter option number:  2

The word is abode
The definition is house or dwelling

The word is chop
The definition is to cut

The word is finis
The definition is the end

The word is grow
The definition is to expand

The word is mart
The definition is market

The word is neaten
The definition is to put in order

The word is yore
The definition is time long past

Dictionary system

Option   Function

1        Load dictionary

2        Print dictionary

Enter option number:<ENTER>
>
```

Shown below is the program that performs the options as demonstrated on the preceding pages. Notice that the line labels mark the beginning of major functional portions of the program and are somewhat self-explanatory. For example, the label MENU is where the code begins that displays and allows the user to select one of the two options, LOAD or PRINT. Note also the use of $ORDER within that segment beginning with the line label PRINT.

Dictionary System Program

```
VECS59    ;ATL/ACE PROGRAMMER-DICTIONARY SYSTEM ;12/1/90
          ;;1
          ;       VARIABLE LIST
          ;
          ;       OPT  =  Option number as entered by the user
          ;       WORD=   The word being defined
          ;       DEF     =    The definition
          ;       ^VECDIC(     =    The dictionary file
          ;
MENU      ;
          W !!,"Dictionary System"
          W !!,"Option    Function"
          W !!,"1         Load dictionary"
          W !!,"2         Print dictionary"
          R !!!,"Enter option number: ",OPT:DTIME G EXIT:OPT=""
          I '$T!(OPT["^") G EXIT
          I OPT<1!(OPT>2) W !,"Enter 1 or 2" G MENU
OPTIONS   ;
          D LOAD:OPT=1
          D PRINT:OPT=2
          G MENU
EXIT      ;
          K OPT,WORD,DEF Q
LOAD      ;
          R !!,"Enter word: ",WORD:DTIME G LOADEX:WORD=""
          I '$T!(WORD["^") G LOADEX
          R !,"Enter short definition: ",DEF:DTIME
          I '$T!(DEF["^") G LOADEX
          S ^VECDIC(WORD)=DEF
          G LOAD
LOADEX    ;
          Q
PRINT     ;
          S WORD=""
          F  S WORD=$O(^VECDIC(WORD)) Q:WORD=""  D OUTPUT
          Q
OUTPUT    ;
          W !!,"The word is ",WORD
          W !,"The definition is ",^VECDIC(WORD)
          Q
```

Sample System with Numeric Subscripts

Shown below is a similar system but the file has numeric subscripts. Note that the user chose the option to enter data and that the employee numbers are not entered in numerical order. When all the data have been entered the user types <ENTER> at the enter employee name prompt to end the program. *(Note: What the user types is double-underlined.)*

```
>D ^VECS512

Employee System

Option      Function

1           Load employees

2           Print employees

Enter option number: 1

Enter employee number: 56
Enter employee name: Smsss,Ralph

Enter employee number: 1
Enter employee name: Mimmm,Jean

Enter employee number: 13
Enter employee name: Lulll,Harry

Enter employee number: 5
Enter employee name: Jojjj,Mary

Enter employee number: 75
Enter employee name: Adaaa,Adam

Enter employee number:<ENTER>
```

The M Code sample below shows how the data that was just entered is actually stored in the global using numeric subscripts.

M Code Sample

```
TRN,STU>D ^%G

Global ^VECEMP
        VECEMP
^VECEMP
^VECEMP(1) = Mimmm,Jean
^VECEMP(5) = Jojjj,Mary
^VECEMP(13) = Lulll,Harry
^VECEMP(56) = Smsss,Ralph
^VECEMP(75) = Adaaa,Adam
Global ^
```

Shown below, the user selects the print option and lists the current names and employee numbers of employees in the file. Notice that the employee numbers are printed in numeric order even though they were not entered that way. *(Note: What the user typed is double-underlined.)*

Employee System

```
Employee System

Option      Function

1           Load employees

2           Print employees

Enter option number: 2

The employee number is 1
The name is Mimmm,Jean

The employee number is 5
The name is Jojjj,Mary

The employee number is 13
The name is Lulll,Harry

The employee number is 56
The name is Smsss,Ralph

The employee number is 75
The name is Adaaa,Adam

Employee System

Option      Function

1           Load employees

2           Print employees

Enter option number:<ENTER>
>
```

Here is the program for the employee system that performs the options as shown on the preceding pages. Notice that the line labels mark the beginning of major functional portions of the program and are somewhat self-explanatory. For example, the label MENU is where the code begins that displays and allows the user to select one of the two options, LOAD or PRINT. Note also the use of $ORDER within that segment beginning with the line label PRINT.

Employee System Program

```
VECS512  ;ATL/ACE PROGRAMMER-EMPLOYEE SYSTEM ;12/1/90
         ;;1
         ;           VARIABLE LIST
         ;
         ;     OPT       =    Option number as entered by the user
         ;     NAME      =    The employee's name
         ;     NR        =    The employee number
         ;     ^VECEMP(  =    The employee file
MENU     ;
         W !!,"Employee System"
         W !!,"Option    Function"
         W !!,"1         Load employees"
         W !!,"2         Print employees"
         R !!!,"Enter option number: ",OPT:DTIME G EXIT:OPT=""
         I '$T!(OPT["^") G EXIT
         I OPT<1!(OPT>2) W !,"Enter 1 or 2" G MENU
OPTIONS  ;
         D LOAD:OPT=1
         D PRINT:OPT=2
         G MENU
EXIT     ;
         K OPT,NAME,NR
         Q
LOAD     ;
         R !!,"Enter employee number: ",NR:DTIME G LOADEX:NR=""
         I '$T!(NR["^") G LOADEX
         R !,"Enter employee name: ",NAME:DTIME
         I '$T!(NAME["^") G LOADEX
         S ^VECEMP(NR)=NAME
         G LOAD
LOADEX   ;
         Q
PRINT    ;
         S NR=""
         F  S NR=$O(^VECEMP(NR)) Q:NR=""  D OUTPUT
         Q
OUTPUT   ;
         W !!,"The employee number is ",NR
         W !,"The name is ",^VECEMP(NR)
         Q
```

Sample System with Mixed String and Numeric Subscripts

This screen shows the entry of stock numbers and item descriptions using a program to accept and store the information. It is a similar system but with subscripts that are mixed numeric and string values. Note that the user chooses option one and then begins to type the data in response to prompts issued by the program. Notice also that the stock numbers are not entered in any particular order. When all the data have been entered the user types <ENTER> at the stock number prompt to end the program. (*Note: What the user types is double-underlined.)*

```
>D ^VECS517

Inventory System

Option     Function

1          Load stock items

2          Print stock items

Enter option number: 1

Enter stock number: 159
Enter item description: manila file folders 3 cut

Enter stock number: 160
Enter item description: manila file folders 5 cut

Enter stock number: 011
Enter item description: copy paper 20 lb.

Enter stock number: 012
Enter item description: copy paper 15 lb.

Enter stock number: K351
Enter item description: blue stick pens

Enter stock number: K352
Enter item description: black stick pens

Enter stock number: K353
Enter item description: red stick pens

Enter stock number: A B54
Enter item description: telephone message pads

Enter stock number: ABC123
Enter item description: stick-on notes
```

```
Enter stock number: 05793
Enter item description: legal pads

Enter stock number: 309-187
Enter item description: white index cards 3" x 5"

Enter stock number: 309-188
Enter item description: white index cards 4" x 6"

Enter stock number: 11
Enter item description: rubber bands

Enter stock number: 15
Enter item description: paper clips

Enter stock number: 100.5
Enter item description: clip dispenser

Enter stock number: 1000.5
Enter item description: paper fasteners

Enter stock number: 1000.0
Enter item description: 3-hole punch

Enter stock number: 99.10
Enter item description: push pins

Enter stock number: 40.0
Enter item description: binder rings

Enter stock number: 0AM
Enter item description: FAX paper

Enter stock number:<ENTER>
>
```

The M Code Sample below shows how the data that was just entered is actually stored in the global using a combination of string and numeric subscripts.

M Code Sample

```
TRN,STU>D ^%G

Global ^VECINV
        VECINV
^VECINV
^VECINV(11) = rubber bands
^VECINV(15) = paper clips
^VECINV(100.5) = clip dispenser
^VECINV(159) = manila file folders 3 cut
^VECINV(160) = manila file folders 5 cut
^VECINV(1000.5) = paper fasteners
^VECINV("011") = copy paper 20 lb.
^VECINV("012") = copy paper 15 lb.
^VECINV("05793") = legal pads
^VECINV("0AM") = FAX paper
^VECINV("1000.0") = 3 hole punch
^VECINV("309-187") = white index cards 3" x 5"
^VECINV("309-188") = white index cards 4" x 6"
^VECINV("40.0") = binder rings
^VECINV("99.10") = push pins
^VECINV("A B54") = telephone message pads
^VECINV("ABC123") = stick-on notes
^VECINV("K351") = blue stick pens
^VECINV("K352") = black stick pens
^VECINV("K353") = red stick pens
Global ^
```

When the print option is selected, the items are printed in collating sequence order by stock number. However, the items are not arranged in a way that is easy to locate by stock number. The next section will discuss the output ordering when subscripts are a mixture of string and numeric values. *(Note: What the user types is double-underlined.)*

```
>D ^VECS517

Inventory System

Option     Function

1          Load stock items

2          Print stock items

Enter option number: 2

The stock number is 11
The description is rubber bands

The stock number is 15
The description is paper clips

The stock number is 100.5
The description is clip dispenser

The stock number is 159
The description is manila file folders 3 cut

The stock number is 160
The description is manila file folders 5 cut

The stock number is 1000.5
The description is paper fasteners

The stock number is 011
The description is copy paper 20 lb.

The stock number is 012
The description is copy paper 15 lb.

The stock number is 05793
The description is legal pads

The stock number is 0AM
The description is FAX paper
```

```
The stock number is 1000.0
The description is 3-hole punch

The stock number is 309-187
The description is white index cards 3" x 5"

The stock number is 309-188
The description is white index cards 4" x 6"

The stock number is 40.0
The description is binder rings

The stock number is 99.10
The description is push pins

The stock number is A B54
The description is telephone message pads

The stock number is ABC123
The description is stick-on notes

The stock number is K351
The description is blue stick pens

The stock number is K352
The description is black stick pens

The stock number is K353
The description is red stick pens

Inventory System

Option     Function

1          Load stock items

2          Print stock items

Enter option number:<ENTER>
>
```

Shown below is the program for the inventory system. It is similar in structure to the previous two programs for the dictionary system and the employee system.

Inventory System Program

```
VECS517  ;ATL/ACE PROGRAMMER-INVENTORY SYSTEM ;12/1/90
         ;;1
         ;       VARIABLE LIST
         ;
         ;       OPT      =  Option number as entered by the user
         ;       DESCR    =  Description of the inventory item
         ;       NR       =  The item number
         ;       ^VECINV( =  The inventory file
MENU     ;
         W !!,"Inventory System"
         W !!,"Option    Function"
         W !!,"1         Load stock items"
         W !!,"2         Print stock items"
         R !!!,"Enter option number: ",OPT:DTIME G EXIT:OPT=""
         I '$T!(OPT["^") G EXIT
         I OPT<1!(OPT>2) W !,"Enter 1 or 2" G MENU
OPTIONS  ;
         D LOAD:OPT=1
         D PRINT:OPT=2
         G MENU
EXIT     ;
         K OPT,DESCR,NR
         Q
LOAD     ;
         R !!,"Enter stock number: ",NR:DTIME G LOADEX:NR=""
         I '$T!(NR["^") G LOADEX
         R !,"Enter item description: ",DESCR:DTIME
         I '$T!(DESCR["^") G LOADEX
         S ^VECINV(NR)=DESCR
         G LOAD
LOADEX   ;
         Q
PRINT    ;
         S NR=""
         F  S NR=$O(^VECINV(NR)) Q:NR=""  D OUTPUT
         Q
OUTPUT   ;
         W !!,"The stock number is ",NR
         W !,"The description is ",^VECINV(NR)
         Q
```

Canonic Order

You noticed in the inventory system that when the file is printed, the global subscripts (in this case stock numbers) are not ordered in a way to easily find a particular stock number in the list. We will now examine the ordering of global subscripts of both numeric and string values.

Order of Mixed Numeric and String Subscripts

11
15
100.5
159
160
1000.5
011
012
05793
0AM
1000.0
309-187
309-188
40.0
99.10
A B54
ABC123
K351
K352
K353

Examine this list of stock numbers that were used in the previous example. They are arranged in canonic order. Can you determine the rule(s) for canonic ordering by studying this list?

There is in fact a method to this order. Whenever you create a global file, M automatically keeps the subscripts in a sorted order called ***canonic order***. If all your subscripts are numeric, they will be in numeric order from lowest to highest values. If all your subscripts are alphabetic strings, they will be in alphabetic order. When your subscripts are mixed numeric and alphanumeric, canonic ordering determines the arrangement.

With mixed types of subscripts, M goes through several steps to sort them:

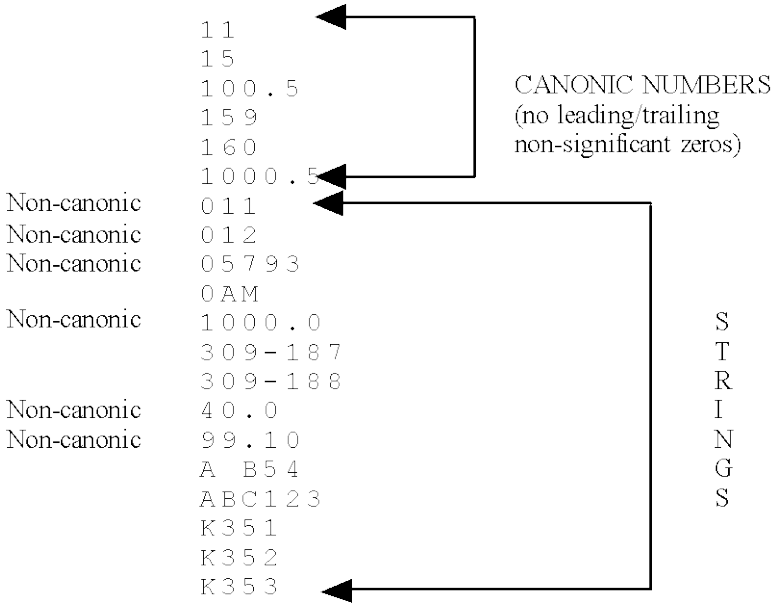
1. Subscripts are separated into two groups -- canonic numbers, and strings that include non-canonic numbers:

> Canonic numbers: Any numeric value that does not have non-significant zeros is considered to be canonic. Canonic numbers are sorted by their numeric values.
>
> Strings and non-canonic numbers: If a numeric value has a leading zero (e.g., 0100) or a trailing zero (e.g., 100.0), these leading and/or trailing zeros are considered to be non-significant. Any number that has non-significant zeros is considered to be non-canonic. Non-canonic numbers are sorted in the same way that alphabetic and/or alphanumeric strings are sorted. M sorts them by comparing characters from left to right.

2. Canonic numbers are sorted first by their numeric values. Then, strings are sorted character by character from left to right. Strings may include non-canonic numbers, alphabetic strings, and/or alphanumeric strings.

As a review of this process, we will look again at the output from the inventory system used in the previous example. Shown below is the same list of stock numbers as before. Canonic numbers are identified and they appear first in the list. The strings are also identified and within the string list the non-canonic numbers are marked. Sorting within the list of strings is based on the ASCII collating sequence with numbers preceding alphabetics. The sorting is performed character by character beginning with the left most character and proceeding to the right.

Order of String Subscripts



$DATA function

Once a file is created, you can check to see if a particular node exists by using the $DATA intrinsic function. The general syntax is:

$DATA(name of the node)

$DATA is used with a command, most often the IF command. Its purpose is to return one of four values:

00 node does not exist at all
01 the node has a value but no descendants (no nodes below it)
10 the node has one or more descendants but has no value
11 the node has one or more descendants and has a value

This digit indicates whether the node has a value

This digit indicates whether the node has one or more descendants

The first digit of the value returned by $DATA specifies whether the node has one or more descendents. The second digit indicates whether the node itself exists.

Shown below are three examples illustrating the usefulness of $DATA.

The first shows its use when a node does not exist. If a non-existent is used in a WRITE statement an error message will be displayed. The $DATA function may be used to test the existence of the node and then take appropriate action.

The second example is for the case when the node does exist. It is printed without error or some action may be taken when $DATA verifies its existence.

The last example is abstracted from a program that will be used later. It shows the use of $DATA for taking one action when the node exists and another when it does not.

You will discover that $DATA is one of the most useful and most used intrinsic functions of M.

Examples using $DATA:

M Code Sample

[Example 1. Record does NOT exist]

```
>W ^VECARR1(7)

<UNDEF>^VECS512:1 W ^VECARR1(7)

>I $D(^VECARR1(7))=0 W !,"Record does NOT exist"

Record does NOT exist

>I $D(^VECARR1(7))=1 W !,"Record DOES exist"
```

[Example 2. Record DOES exist]

```
>W ^VECARR1(1)

Mommm,Linda

>I $D(^VECARR1(1))=1 W !,"Record DOES exist"

Record DOES exist
```

[Example 3. Branch if record does not exist]

```
I $D(^VECS5G(NAME))=0 W !,"Record does not exist"
I  S MISSING=1 G LOOKEXIT
```

Indirection

In the routines we've written so far, the values of variables have contained data. However, in M there is a feature called ***indirection***, which enables us to store all or some of an M statement(s) in a variable. Once M code is stored in a variable (code in the node), the routine can then execute the code in the variable by indirection.

The purpose of indirection is to reduce the amount of redundant or repetitious code in a routine. It can also stream the logical flow of a program and reduce the overall amount of code required to accomplish a task. It is one of the most powerful features of M, but it must be use judiciously. As we will see in a later section there are both advantages and disadvantages to using indirection.

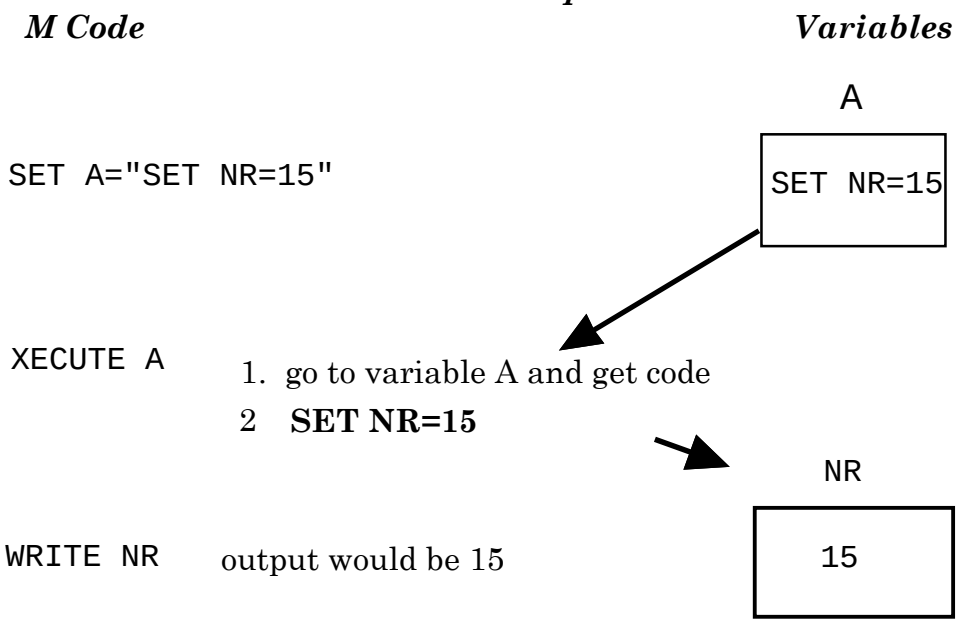
There are five types of indirection. The type of indirection depends upon what part of a M statement is stored in a variable called an indirection variable:

command : a command(s) and all of its arguments are stored in a variable
argument : the argument of a command is stored in a variable
name : the name of a line label or routine is stored in a variable
pattern : a pattern match is stored in a variable
subscript : a subscript is stored in a variable

Command Indirection

How command indirection works is shown on the next page. Generally, one or more M commands plus their arguments is set into the indirection variable. The M command XECUTE is then used with the indirection variable as shown. Here the indirection variable A contains a simple set command. When it is used as an argument to the XECUTE command, the first step is to get the value of the variable A, which is M code. The next step is to perform the code and in this example the variable NR is set to the value of 15. If you use the XECUTE command and the indirection variable does not contain a complete command with argument(s), you will get an error.

M Code Sample

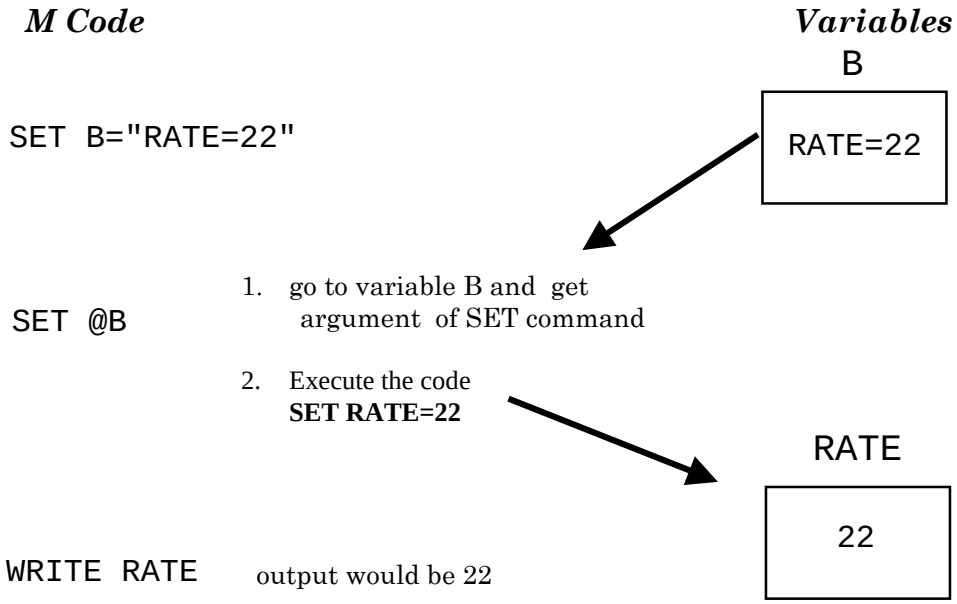


Argument Indirection

Compare this graphic to the graphics where the XECUTE command was illustrated. In the case of the XECUTE, the indirection variable contains complete M code. In this form of indirection, the indirection variable is preceded with the @ symbol to become the argument of a command. It only contains the argument(s) of the particular command to be performed.

The graphic on the next page illustrates this using the SET command. The indirection variable B is given a value of RATE, which equals 22. Next the SET command is performed with the indirection variable. The first step is to obtain the value of the indirection variable, which then becomes the argument of the SET command. In the next step, the SET command is performed to set the value of the variable rate to 22.

M Code Sample



A common use of this form of indirection is to provide the character(s) necessary to perform various screen functions, including clearing the screen:

At the beginning of a routine: SET IOF="#"

The pound sign is the character (#) that clears many terminal screens. If this doesn't work, you will have to find the manual for your terminal and look up the procedures for clearing the screen. Where you want to clear the screen, you insert the statement:
WRITE @IOF

You must be careful not to split the argument of a command so that part of it is in the indirection variable and the remaining part is on the command line, since this will cause an error. In the M code sample below, M will not be able to recognize the second line as a valid argument for the SET command and an error will occur.

M Code Sample

```
SET B="RATE=22"
SET @B-20      <--illegal use of indirection
```

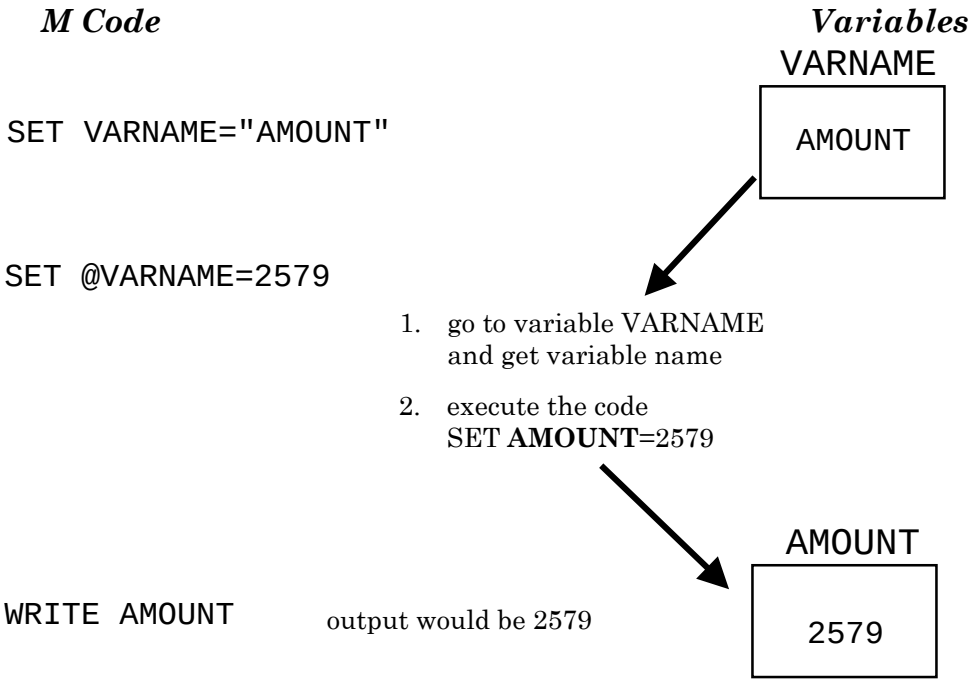
Name Indirection

In name indirection, the indirection variable is again preceded with the @ symbol just as in argument indirection, but in this case, it is used as a variable name, line reference, or routine name. In the example shown below, indirection is used to specify a variable name.

First, a variable name is SET into the variable VARNAME. M recognizes that the SET statement is using name indirection and then gets the name of the variable, in this case, AMOUNT.

Then it executes the command setting the value of AMOUNT to 2579.

M Code Sample



Name Indirection Used for a Line Reference

In the example on the next page, indirection is used to specify a line label for use in a DO statement. The variable label is SET to value of the line label PRINT.

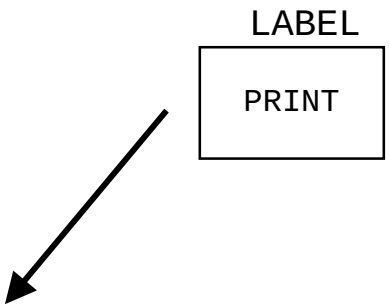
M recognizes that indirection is in use, acquires the value of label, and then executes code that begins at the line labeled PRINT.

M Code Sample

M Code ***Variable***

```
SET LABEL="PRINT"

DO @LABEL
```



1. go to variable LABEL and get line label for the DO
2. execute the code DO **PRINT**

M would DO the line label PRINT

Pattern Indirection

In this form of indirection, the indirection variable is preceded by the @ symbol and is used after the pattern match operator (?) on a conditional statement:

In the example shown on the next page, the variable DATA is SET equal to a string that corresponds to a typical social security number. Then the variable PATTERN is SET equal to the pattern that determines acceptability. The IF statement identifies the variable PATTERN as the variable holding the pattern by use of the ampersand.

The first step is to get the variable PATTERN and use its value as the operand for the pattern match. Then the pattern match is executed. In this case the pattern matches so the IF is false.

M Code Sample

M Code ***Variables***

```
SET DATA="000-222-3333"
```

DATA: 000-222-3333

```
SET PATTERN="3N1""-""2N1""-""4N"
```

PATTERN: 3N1"-"2N1"-"4N

1. Go to variable PATTERN and get pattern match
2. Execute the code
   IF DATA'?3N1"-"2N1"-"4N

```
IF DATA'?@PATTERN WRITE !,"No match"
```

Because the data matches the pattern, there would be no output.

Shown below is a similar example. This pattern match again uses indirection and is determining if the data matches five numbers. In this case it does not.

M Code Sample

M Code ***Variables***

```
SET DATA=444

SET PATTERN="5N"
```

DATA: 444

PATTERN: 5N

1. go to variable PATTERN and get pattern match
2. Execute the code IF DATA'?5N

```
IF DATA'?@PATTERN WRITE !,"No match"
```

Because the data does NOT match the pattern, the phrase "No match" would be output.

Subscript Indirection

This form of indirection is more complex than the other forms. An indirection variable is used to hold the name of the array and any initial subscripts. Later, an indirection variable will be used to hold the last subscript(s). This for of indirection departs from the general syntax and behaves somewhat like a shorthand version of concatenation. For purposes of illustration, two forms of indirection are used in the example shown on the next page.

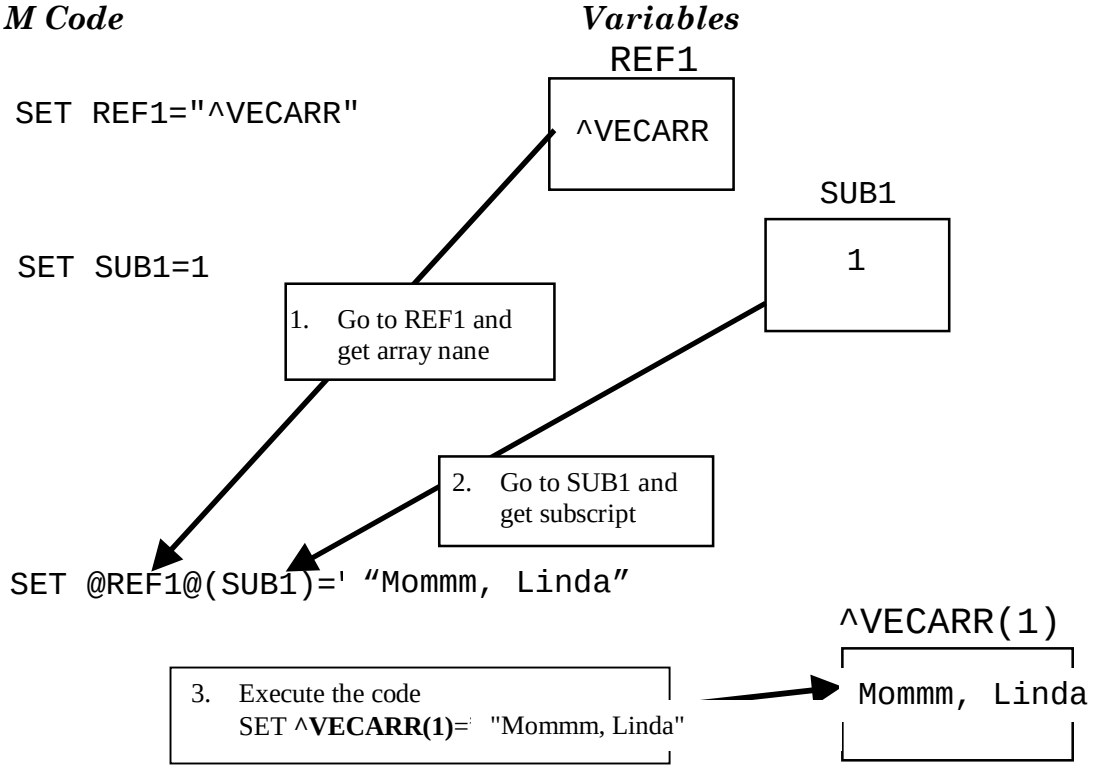
First, SET the values for the array name REF1 and the subscript SUB1. The third SET statement uses name indirection to specify the array name and subscript indirection to specify the subscript. The subscript variable is enclosed in parentheses.

When this SET command is executed it first resolves the name of the array from the variable REF1.

Then the subscript is resolved from the variable SUB1.

Finally, the newly formed command is executed to SET the global node up arrow VECARR1.

M Code Sample



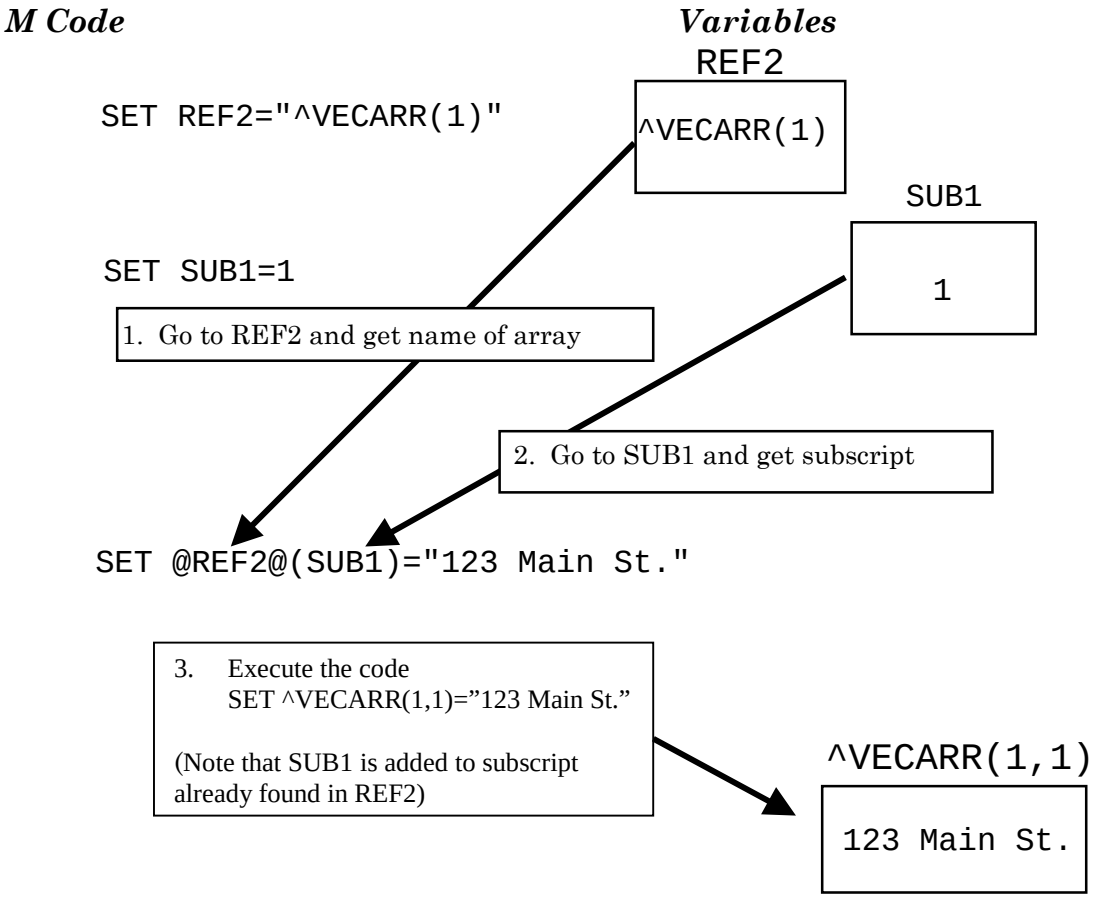
Shown on the next page is another example of subscript indirection. This is very similar to the preceding example, except in this case variable REF2 contains an array name with a top-level subscript.

When the third SET statement is executed, the array name with the subscript is first resolve.

Then the second level subscript is resolved and appended to the first.

Finally, the global node ^VECARR(1,1) is SET.

M Code Sample



Precautions Regarding Indirection

Indirection is a useful and powerful M tool. However, some caution must be exercised in its use.

Since the value of variables can be changed, any code, line references, names, or arguments placed in indirection variables can also be changed. That means that an M routine can modify itself! Also, a maintenance programmer must find where an indirection variable is assigned before they can determine what indirection does, frequently increasing the amount time required to correct problems or make program modifications. When indirection is used, make sure its use is well documented.

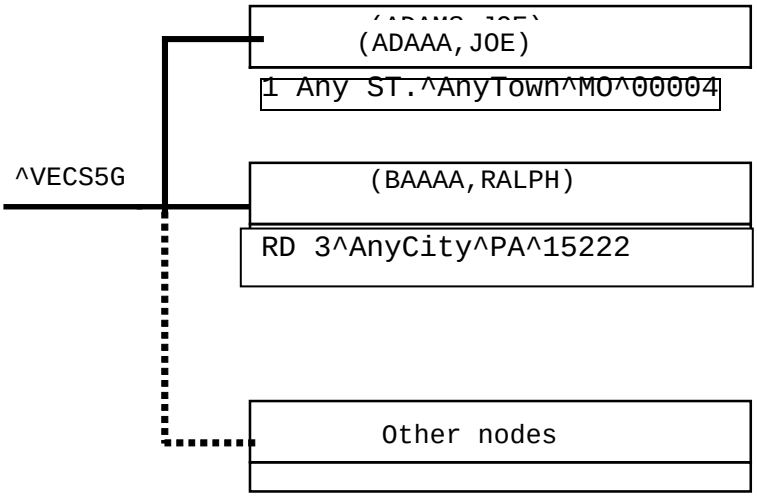
The use of indirection can slow a routine down. M will have to go to the indirection variable first before it can actually execute the intended code.

Building a System

With all the M language components we have learned so far, we are going to build a sample system that will allow us to create, edit, delete, and print file entries for an electronic Mailing List system.

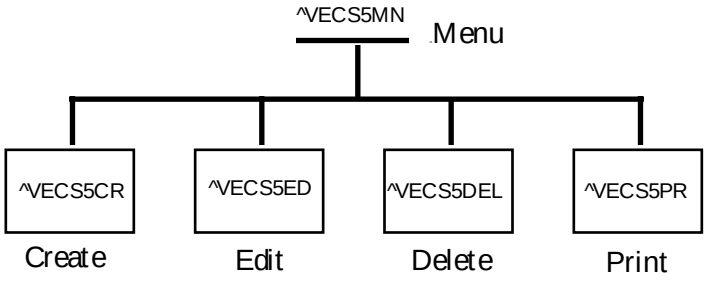
File Structure

Our file structure will use the person's name as the subscript of each node. The content of each node will be the mailing address.



System Structure

The system will be comprised of individual routines that are stored in separate routine files. The graphic schematically outlines the routines that will be used to build the mailing list system. The program code will be listed later, but first a testing sequence will be illustrated.



Standard Features of the System

The table on the following pages shows the interactions required to test each prompt in VECS5MN thoroughly. Some values that are unacceptable have been entered to make sure that validation and "error trapping" are performing as intended.

Please note that the spacing of the interactive example is not as you would see it on the computer. Spacing adjustments were made to fit the sample into the lesson materials. (*What the user typed is double-underlined.)*

The user will be able to create, edit, delete, and print entries. At most prompts, the user can type a "?" to get help.

VA Programming Standards and Conventions
All prompts requesting user input must provide additional help when the user enters a question mark ("?").

Review the testing procedure thoroughly, noting all interactions.

Sample Interaction with Testing Procedures

	Testing
>D ^VECS5MN	
Mailing List System	
Options Function	
1.....Create Mailing List entry 2.....Edit Mailing List entry 3.....Delete Mailing List entry 4.....Print Mailing List entry	
Enter option number: ^	We test each prompt with an ^
>D ^VECS5MN	
Mailing List System	
Options Function	
1.....Create Mailing List entry 2.....Edit Mailing List entry 3.....Delete Mailing List entry 4.....Print Mailing List entry	

```
Enter option number: 0
```

We enter an invalid option number

```
Option must be between 1 and 4
Enter option number:5
```

We enter an invalid option number

```
Option must be between 1 and 4
Enter option number: 1


Enter NAME: 234
```

We make an entry that violates the pattern match

```
Enter the name of the new entry.  Enter last
name, first name
Enter NAME:XXXXXXXXXXXXXXXXXXXXXXXX
Enter the name of the new entry.  Enter last
name, first name
Enter NAME:?
Enter the name of the new entry.  Enter last
name, first name
Enter NAME:MOMMM
Enter the name of the new entry.  Enter last
name, first name

Enter NAME:^


Mailing List System

Options    Function

1......Create Mailing List entry
2......Edit Mailing List entry
3......Delete Mailing List entry
4......Print Mailing List entry

Enter option number: 1


Enter NAME: MOMMM,LINDA
ADDRESS: XXXXXXXXXXXXXXXXXXXXXXXXXXXXX

Address must not exceed 25 characters


ADDRESS: ?

Address must not exceed 25 characters

ADDRESS: ^


Enter NAME: MOMMM,LINDA
```

We make an entry that is too long

We try the ?

We enter only the last name

We always test the ^

We've thoroughly tested NAME so we go to the next field

We make an entry that is too long

We test the ?

We test the ^

```
ADDRESS: 101 ANY ST.
CITY: XXXXXXXXXXXXXXXXXXXXXXXXXX

City must not exceed 20 characters

CITY: ?
City must not exceed 20 characters

CITY: ^

Enter NAME: MOMMM,LINDA
ADDRESS: 101 ANY ST.
CITY: ANYWHERE

STATE: XXXX

State must be a 2 letter abbreviation
STATE: 22
State must be a 2 letter abbreviation

STATE: ?
State must be a 2 letter abbreviation
STATE: ^

Enter NAME: MOMMM,LINDA
ADDRESS: 101 ANY ST.
CITY: ANYWHERE
STATE: TX

ZIP CODE: XXXXXXX

Zip code must be five or nine digits, no
hyphens

ZIP CODE: AAAAA

Zip code must be five or nine digits, no
hyphens

ZIP CODE: ?
Zip code must be five or nine digits, no
hyphens

ZIP CODE: ^
```

We've tested the NAME and ADDRESS so we go on to the CITY; we make an entry that is too long

We test the ?

We test the ^

We go on to the STATE and make an entry that's too long

We make an entry that violates the pattern match

We test the ?

We test the ^

We make an entry in ZIP CODE that is too long

We make an entry that violates the pattern match

We test the ?

We test the ^

```
Enter NAME: MOMMM,LINDA
ADDRESS: 101 ANY ST.
CITY: ANYWHERE
STATE: TX
ZIP CODE: 22222
```

All fields have been tested thoroughly for length, pattern, ?, and ^

```
Enter NAME: JOJJJ,MARY
ADDRESS:  <ENTER>
CITY: <ENTER>
STATE: <ENTER>
ZIP CODE: <ENTER>
```

We now make sure that we can bypass each field

```
Enter NAME: SMSSS,RALPH
ADDRESS: RD 3
CITY: ANYTOWN
STATE: GA
ZIP CODE: 00006
```

We enter an additional record

```
Enter NAME: <ENTER>
```

We press <ENTER> to go back to the menu

```
Mailing List System

Options   Function

1......Create Mailing List entry
2......Edit Mailing List entry
3......Delete Mailing List entry
4......Print Mailing List entry

Enter option number: 2
```

We move to the edit function

```
Enter NAME: 234

Enter the name of the new entry.  Enter last
name, first name

Enter NAME:XXXXXXXXXXXXXXXXXXXXXXXXX
Enter the name of the new entry.  Enter last
name, first name

Enter NAME:?
Enter the name of the new entry.  Enter last
name, first name
```

We go through the same testing as we did for NAME in the Create function

```
Enter NAME:MOMMM
Enter the name of the new entry.  Enter last
name, first name

Enter NAME:^



Mailing List System

Options    Function

1......Create Mailing List entry
2......Edit Mailing List entry
3......Delete Mailing List entry
4......Print Mailing List entry

Enter option number: 2


Enter NAME: MOMMM,MELINDA
Record does not exist




Enter NAME: MOMMM,LINDA
ADDRESS: 101 ANY ST.//
        XXXXXXXXXXXXXXXXXXXXXXXXX
Address must not exceed 25 characters

ADDRESS: 101 ANY ST.//?
Address must not exceed 25 characters

ADDRESS:  101 MAIN ST.//^




Enter NAME: MOMMM,LINDA
ADDRESS: 101 ANY ST.//202 YOUR ST.


CITY: ANYWHERE//XXXXXXXXXXXXXXXXXXXX
City must not exceed 20 characters
```

Because we cannot edit an entry that doesn't exist, we test it

We test the ADDRESS field as we did in Create

We test the CITY field as we did in Create

```
CITY: ANYWHERE//?
City must not exceed 20 characters
CITY: ANYWHERE//^


Enter NAME: MOMMM,LINDA
ADDRESS: 202 ANY ST.//<ENTER>
CITY: ANYWHERE//NOPLACE
STATE: TX//XXXX
State must be a 2 letter abbreviation

STATE: TX//22
State must be a 2 letter abbreviation

STATE: TX//?
State must be a 2 letter abbreviation

STATE: TX//^


Enter NAME: MOMMM,LINDA
ADDRESS: 202 ANY ST.//<ENTER>
CITY: NOPLACE//<ENTER>
STATE: TX//PA
ZIP CODE: 00002//XXXXXXX
Zip code must be five or nine digits, no
hyphens

ZIP CODE: 00002//AAAAA
Zip code must be five or nine digits, no
hyphens

ZIP CODE: 00002//?
Zip code must be five or nine digits, no
hyphens

ZIP CODE: 00002//^


Enter NAME: MOMMM,LINDA
ADDRESS: 202 YOUR ST.//<ENTER>
CITY: NOPLACE//<ENTER>
STATE: PA//<ENTER>
ZIP CODE: 00002//<ENTER>

Enter NAME: <ENTER>
```

We test the STATE field as we did in Create

We test the ZIP CODE field as we did in Create

We make sure we can bypass each of the fields

```
Mailing List System

Options    Function

1......Create Mailing List entry
2......Edit Mailing List entry
3......Delete Mailing List entry
4......Print Mailing List entry

Enter option number: 3
```

We enter the Delete function

```
Enter NAME: MOMMM,MELINDA
Record does not exist
```

We enter a NAME that isn't there

```
Enter NAME: 234
Enter the name of the new entry.  Enter last
name, first name

Enter NAME: XXXXXXXXXXXXXXXXXXXXXXXXXX
Enter the name of the new entry.  Enter last
name, first name

Enter NAME: ?
Enter the name of the new entry.  Enter last
name, first name

Enter NAME: MOMMM
Enter the name of the new entry.  Enter last
name, first name

Enter NAME: ^


Mailing List System

Options    Function

1......Create Mailing List entry
2......Edit Mailing List entry
3......Delete Mailing List entry
4......Print Mailing List entry

Enter option number: 3


Enter NAME: MOMMM,LINDA
```

We test the NAME prompt as we did above

```
Are you sure? NO//
Name not deleted!
```

We make sure we can bypass deleting a record we've selected

```
Enter NAME: MOMMM,LINDA
Are you sure? NO//YES
Name deleted!
```

We make sure we can delete a record

```
Enter NAME: MOMMM,LINDA
Record does not exist
```

We make sure the record has actually been deleted

```
Enter NAME: SMSSS,RALPH

Are you sure? NO//^
```

We test the ^ on this prompt

```
Enter NAME: <ENTER>



Mailing List System

Options    Function

1......Create Mailing List entry
2......Edit Mailing List entry
3......Delete Mailing List entry
4......Print Mailing List entry

Enter option number: 4
```

We enter the Print function

```
          Mailing List File Listing

JOJJJ,MARY
```

Here's the record we entered without any address information

```
SMSSS,RALPH
RD 3
ANYTOWN
GA
00006

Press Enter to go back to the menu: <ENTER>
```

```
Mailing List System

Options    Function

1......Create Mailing List entry
2......Edit Mailing List entry
3......Delete Mailing List entry
4......Print Mailing List entry

Enter option number: <ENTER>
>
```

VECS5MN--The Menu Routine

The following pages show the program for displaying the options the user may select and which prompt the user to make a choice. The variables IOF and DTIME are assumed to have been SET before entering this routine and those that follow.

```
VECS5MN  ;ATL/ACE PROGRAMMER-MAILING LIST SYSTEM ;12/1/90
         ;;1
         ;VARIABLE LIST
         ;
         ;OPT      = Option number as entered by user
         ;IOF      = Control characters to clear screen (not to
         ;           be KILLed)
         ;DTIME  =  Delay time for READs (not to be KILLed)
         ;II       = Loop counter to display menu text
         ;
MAIN     ;
         D DISMENU
         I OPT=""!(OPT["^") G EXIT
         D ^VECS5CR:OPT=1,^VECS5ED:OPT=2,^VECS5DEL:OPT=3,^VECS5PR:OPT=4
         G MAIN
DISMENU  ;
         W @IOF
         W !!!,"Mailing List System"
         W !!,"Options    Functions"
         F II=1:1:4 D DISMENU2
DISMENU1 ;
         R !!,"Enter option number: ",OPT:DTIME
         I '$T!(OPT["^")!(OPT="") Q
         I (OPT<1)!(OPT>4)!(OPT="?") W !!,"Option must be "
         I  W "between 1 and 4" G DISMENU1
         Q
DISMENU2 ;
         W !!,II,$P($T(OPTIONS+II),";",3)
         Q
```

```
EXIT     ;
         K II,OPT
         Q
OPTIONS  ;
         ;;.....Create Mailing List entry
         ;;.....Edit Mailing List entry
         ;;.....Delete Mailing List entry
         ;;.....Print Mailing List entry
```

M code for the Sample System

The following pages show our complete system.

Shown below is the program for creating and adding a new entry to the mailing list database.

VECS5CR--The Create Routine

```
VECS5CR  ;ATL/ACE PROGRAMMER-MAILING LIST SYSTEM ;12/1/90
         ;;1
         ;VARIABLE LIST
         ;
         ;NAME    =  Individual's name
         ;ADDR    =  Address
         ;CITY    =  City
         ;STATE   =  State
         ;ZIP     =  Zip code
         ;
START    ;
         W @IOF
         W !!,?20,"Create Mailing List entry"
NAME     ;
         R !!,"Enter NAME : ",NAME:DTIME I '$T!(NAME["^")!(NAME="") G
EXIT
         I NAME'?1U.AP1","1U.AP!($L(NAME)>30)!(NAME["?")
         I  W !,"Enter the name of the new entry. "
         I  W "Enter last name,first name",! G NAME
         D CREATE,STORE G NAME
EXIT     ;
         K NAME,ADDR,CITY,STATE,ZIP
         Q
CREATE   ;
         S (ADDR,CITY,STATE,ZIP)=""
ADDR     ;
         R !,"ADDRESS : ",ADDR:DTIME I '$T!(ADDR["^") G CREXIT
         I ADDR="" G CITY
         I ADDR'?.ANP!($L(ADDR)>25)!(ADDR["?")
         I  W !,"Address must not exceed 25 characters",! G ADDR
```

```
CITY    ;
        R !,"CITY : ",CITY:DTIME I '$T!(CITY["^") G CREXIT
        I CITY="" G STATE
        I CITY'?.ANP!($L(CITY)>20)!(CITY["?")
        I  W !,"City must not exceed 20 characters" G CITY
STATE   ;
        R !,"STATE : ",STATE:DTIME I '$T!(STATE["^") G CREXIT
        I STATE="" G ZIP
        I STATE'?2U!(STATE["?") W !,"State must be a 2 letter"
        I  W " abbreviation",! G STATE
ZIP     ;
        R !,"ZIP CODE : ",ZIP:DTIME I '$T!(ZIP["^") G CREXIT
        I ZIP="" G CREXIT
        I ZIP'?5N&(ZIP'?9N)!(ZIP["?") W !,"Zip code must be five"
        I  W " or nine digits, no hyphens" G ZIP
CREXIT  ;
        Q
STORE   ;
        S ^VECS5G(NAME)=ADDR_"^"_CITY_"^"_STATE_"^"_ZIP
        Q
```

VECS5ED--The Edit Routine

Shown below is the program for editing an existing entry in the mailing list database.

```
VECS5ED ;ATL/ACE PROGRAMMER-MAILING LIST SYSTEM ;12/1/90
        ;;1
        ;  VARIABLE LIST
        ;  NAME, ADDR, CITY, STATE, ZIP = same as VECS5CR
        ;  MISSING   =    set to 1 if NAME does not exist
        ;  RECORD=   contains record copied from ^VECS5G
        ;  INDATA =   user's response before it is validated
START   ;
        W @IOF
        W !!,?20,"Edit Mailing List entry"
NAME    ;
        S MISSING=0
        R !!,"Enter NAME : ",NAME:DTIME I '$T!(NAME["^")!(NAME="") G EXIT
        I NAME'?1U.A1","1U.AP!($L(NAME)>30)!(NAME["?")
        I  W !,"Enter the name of the new entry."
        I  W "  Enter last name,first name",! G NAME
        D LOOKUP D STORE:MISSING'=1 G NAME
EXIT    ;
        K NAME,ADDR,CITY,STATE,ZIP,MISSING,RECORD,INDATA
        Q
```

```
LOOKUP  ;
        I $D(^VECS5G(NAME))=0 W !,"Record does not exist"
        I  S MISSING=1 G LOOKEXIT
        S RECORD=^VECS5G(NAME)
        S ADDR=$P(RECORD,"^",1)
        S CITY=$P(RECORD,"^",2)
        S STATE=$P(RECORD,"^",3)
        S ZIP=$P(RECORD,"^",4)
ADDR    ;
        W !,"ADDRESS : ",$S(ADDR="":"",1:ADDR_"//") R INDATA:DTIME
        I '$T!(INDATA["^") G LOOKEXIT
        I INDATA="" G CITY
        I INDATA'?.ANP!($L(INDATA)>25)!(INDATA["?") W !,"Address must not "
        I  W "exceed 25 characters",! G ADDR
        S ADDR=INDATA
CITY    ;
        W !,"CITY : ",$S(CITY="":"",1:CITY_"//") R INDATA:DTIME
        I '$T!(INDATA["^") G LOOKEXIT
        I INDATA="" G STATE
        I INDATA'?.ANP!($L(INDATA)>20)!(INDATA["?") W !,"City must not exceed"
        I  W " 20 characters" G CITY
        S CITY=INDATA
STATE   ;
        W !,"STATE : ",$S(STATE="":"",1:STATE_"//") R INDATA:DTIME
        I '$T!(INDATA["^") G LOOKEXIT
        I INDATA="" G ZIP
        I INDATA'?2U!(INDATA["?") W !,"State must be a 2 letter"
        I  W " abbreviation",! G STATE
        S STATE=INDATA
ZIP     ;
        W !,"ZIP CODE : ",$S(ZIP="":"",1:ZIP_"//") R INDATA:DTIME
        I '$T!(INDATA["^") G LOOKEXIT
        I INDATA="" G LOOKEXIT
        I INDATA'?5N&(INDATA'?9N)!(INDATA["?") W !,"Zip code must be five or"
        I  W " nine digits, no hyphens" G ZIP
        S ZIP=INDATA
LOOKEXIT ;
        Q
STORE   ;
        S ^VECS5G(NAME)=ADDR_"^"_CITY_"^"_STATE_"^"_ZIP
        Q
```

VECS5DEL--The Delete Routine

Shown below is the program that deletes an existing entry in the mailing list database.

```
VECS5DEL ;ATL/ACE PROGRAMMER-MAILING LIST SYSTEM ;12/1/90
        ;;1
        ; VARIABLE LIST
        ;
        ; NAME      =    Individual's name
        ; ANS       =    User's response:
        ;                  <ENTER>, YES, or Y if they want to
        ;                  delete record
        ;                  any other response if they don't
        ;                  want to delete
        ; MISSING   =    Set to 1 if NAME does not exist
START   ;
        W @IOF
        W !!,?20,"Delete Mailing List entry"
NAME    ;
        S MISSING=0
        R !!,"Enter NAME : ",NAME:DTIME I '$T!(NAME["^")!(NAME="") G EXIT
        I NAME'?1U.A1","1U.AP!($L(NAME)>30)!(NAME["?")
        I  W !,"Enter the name of the new entry."
        I  W "  Enter last name,first name",! G NAME
        D LOOKUP D DELETE:MISSING'=1 G NAME
EXIT    ;
        K NAME,ANS,MISSING
        Q
LOOKUP  ;
        I $D(^VECS5G(NAME))=0 W !,"Record does not exist"
        I  S MISSING=1
        Q
DELETE  ;
        R !!,"Are you sure? NO//",ANS:DTIME
        I '$T!(ANS["^") G DELEXIT
        I (ANS="Y")!(ANS="YES") K ^VECS5G(NAME)
        I  W !,"Name deleted!"
        E  W !,"Name not deleted!"
DELEXIT ;
        Q
```

After entering, editing, and deleting a record, the following entries for the Mailing List System are stored.

M Code Sample

```
TRN,STU>D ^%G

Global ^VECS5G
         VECS5G
^VECS5G
^VECS5G("JOJJJ,MARY") = ^^^
^VECS5G("SMSSS,RALPH") = RD 3^ANYTOWN^GA^00006
Global ^
```

VECS5PR--The Print Routine

Shown below is the program for printing all the entries in the mailing list database.

```
VECS5PR ;ATL/ACE PROGRAMMER-MAILING LIST SYSTEM ;12/1/90
        ;;1
        ;  VARIABLE LIST
        ;
        ;  NAME       =   Individual's name
        ;  RECORD   =   copy of ^VECS5G(NAME) node
        ;  ANS        =   user's response to "Press Enter..." prompt
        ;
START   ;
        D HEADING
        D ORDER
        R !!,"Press Enter to go back to the menu",ANS:DTIME
EXIT    ;
        K NAME,RECORD,ANS
        Q
HEADING ;
        W @IOF
        W !!,?30,"Mailing List File Listing"
        Q
ORDER   ;
        S NAME=""
        F  S NAME=$O(^VECS5G(NAME)) Q:NAME=""  D DISPLAY
        Q
DISPLAY ;
        W !!,NAME
        I $D(^VECS5G(NAME))=0 Q
        S RECORD=^VECS5G(NAME)
        W !,$P(RECORD,"^",1)
        W !,$P(RECORD,"^",2)
        W !,$P(RECORD,"^",3)
        W !,$P(RECORD,"^",4)
        Q
```

Limitations of the Sample System

The following are some of the limitations you will find with the sample system:

1. There is a lot of redundant code. For example, the create and edit routines use the same validation (pattern matches) for each field.

2. You cannot add a duplicate name. If a name already exists and you make another entry for the same name, the second entry overwrites the first one.

3. You can print the data only in order by name.

4. The menu routine is not "generic." That is, to add a menu item, you must change several lines of code. In the next lesson, we will develop a "generic" menu routine that will require only one line change to add a menu option.

5. In the edit routine, you can't delete a field entry.

All of these limitations will be addressed and resolved in future lessons.

Assignment **(Optional)**

In General:

1. Follow the VA Programming Conventions as illustrated in this lesson.

2. Use only the techniques, commands, and features described in this lesson or the previous lessons.

3. Save the routine(s) under a namespace assigned to you on a test system by your local facility. For this lesson, use the following suffix pattern:

   ^VEC5A for the menu routine
   ^VEC5B for the create routine
   ^VEC5C for the edit routine
   ^VEC5D for the delete routine
   ^VEC5E for the print routine

Important

You will need to create a global file. In your routine, use the following suffix pattern:

^VEC5G

Mandatory Specifications

You are to design and implement a personnel system *following the Sample System in this lesson*. The global file must contain the following fields (each field should be validated according to the format below):

Field	***Format***
Name	Last name,First name You must use the Name as the subscript; name must be entered in all CAPS with no space before or after the comma.
Address	No more than 25 characters
City	No more than 20 characters
State	2-letter code
Zip	5 numeric digits
Social security number	nnn-nn-nnnn (use hyphens only)

NOTE: Please make sure that your system prompts for the above fields in the order shown. Also, follow each prompt with a space, a colon (:) and a space. For example, your prompt for the name field might be:

Enter Name|_|:|_| *where |_| indicates a space*

Your system should have the following functions:

- Menu which "drives" the entire system
- Create routine for new entries
- Edit routine to edit entries. Make sure to show any prior values (as default responses)
- Delete routine to delete entries. Make sure you ask the user to confirm the deletion
- Print routine to print the file. Design the output in whatever format you choose

All prompts should be timed and escapable.
The user should be able to type a ? on all prompts to get help.