

VA M PROGRAMMING

Intermediate



M

VA M Programming Intermediate

Lesson 7

Index Files

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

VA M PROGRAMMING

Intermediate

Table of Contents

Guide to Symbols Used	1
Optional Reading References	1
References	1
Document Update Note	1
Objectives	2
Enhancements to our Mailing List System	2
Extended File Structure	3
Index Nodes	3
M Code Sample	5
M Code Sample	5
Creating Index Nodes	6
M Code Sample	6
Using an Index Node	6
M Code Sample	6
M Code Sample	6
M Code Sample	7
FOR Loop with Index Nodes	7
\$ORDERing Through the Data Nodes	7
M Code Sample	9
\$ORDERing Through the Index Nodes	10
M Code Sample	10
M Code Sample	12
M Code Sample	12
Reducing Redundancy	13
New Create Routine	13
Adding Duplicate Names	13
M Code Sample	13
M Code Sample	14
Pseudocode for VECS7CRI, the Create Routine	14
M Code Sample for VECS7CRI, the Create Routine	16
New Lookup-Names-to-Edit Routine	17
Selecting from Duplicate Names	18
M Code Sample	18
M Code Sample	18
Pseudocode for VECS7EDI, the Lookup-Names-to-Edit Routine	19

VA M PROGRAMMING

Intermediate

M Code for VECS7EDI, the Lookup-Names-to-Edit Routine	20
New Edit-Fields Routine	22
Deleting Records	22
M Code Sample	22
Deleting Fields	23
M Code Sample	23
Lesson 6 M Code Sample	23
M Code for the Edit-Fields Routine	24
M Code Sample	24
Decision Table of the Edit-Fields Routine	27
Routines That Required Minor Modifications	28
The Menu Routine	28
The Print Routine	29
M Code Sample	29
Naked References	30
The Naked State	30
The Naked Reference	31
M Execution of Naked References:	32
M Code Sample	33
Naked Indicator	33
Naked References in the Sample System	34
M Code Sample	34
M Code Sample	35
ANSI M Standard	35
Components of the Language Standard	36
M Commands	36
M Functions	36
M Special Variables	36
VA Written Utility %INDEX	37
M Code Sample	37
M Code Sample	38
M Code Sample	38
M Code Sample	38
M Code Sample	39
M Code Sample	39
M Code Sample	41
Assignment (optional)	44
In General	44

VA M PROGRAMMING

Intermediate

Mandatory Specifications

45

VA M PROGRAMMING

Intermediate

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

VA M PROGRAMMING

Intermediate

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 put a complete system together using the following commands and techniques:

- Index files: creating and using
- Reducing redundancy in the code
- Allowing for duplicate names; showing duplicate names
- Having the user enter the @ to delete records and field entries
- %INDEX for information about routines
- Naked references

Enhancements to our Mailing List System

In this lesson, we will enhance our Mailing List system in several additional ways:

1. We will reduce the amount of redundant code by creating an edit routine that contains all the validation and editing features that will be used by both the create and edit functions.
2. We will add to the edit function the ability to delete an entire record or a selected field.
3. In the create function, when attempting to add a record for which an entry with the same name already exists, we will issue a message. The message will state that there is already a record with the same name, but if you want to add it anyway, enter it again with quotes around it.
4. In the edit function, if you enter or select a name for which there are multiple entries (duplicate names) a list of these duplicates along with the phone number will be displayed, so that you can select the right entry to edit.
5. We have extended the file structure to include an index by name that will allow us to access records faster and more efficiently.

VA M PROGRAMMING

Intermediate

Extended File Structure

Index Nodes

As an illustration, If you want to find information about **globals** in your reference book, you would use the following procedure:

1. Turn to the book's index, which lists topics in alphabetical order.
2. Scan through the index until you have found the word globals.
3. Go to the first page listed beside globals.
4. Read all of the information related to globals.
5. Go back to the index and go to the next page listed beside globals.
6. Repeat Steps 4 and 5 until there are no more references.

To summarize, the text of the book itself is organized in one order (usually logically by chapters) and the index is organized in another order (usually alphabetical order by topic). The index provides an alternate way to access the original text.

Index files in M are very much like an index of a book. Our ^VECS7G global file, up to this point, contains two types of nodes:

^VECS7G(0)	the header node
^VECS7G(record number,0)	the data nodes

The header node, which has only the one subscript, zero,
and the data nodes, which have two subscripts, the record number and zero.

The problem with this structure is that if we want to find a name of an individual, we have to use \$ORDER, beginning either at the first or last data node, and step through the data nodes until we find a match. This would roughly be equivalent to scanning every page of a book looking for the word "*globals*."

VA M PROGRAMMING

Intermediate

Lets construct an index node that will have name as a subscript as well as the record number:

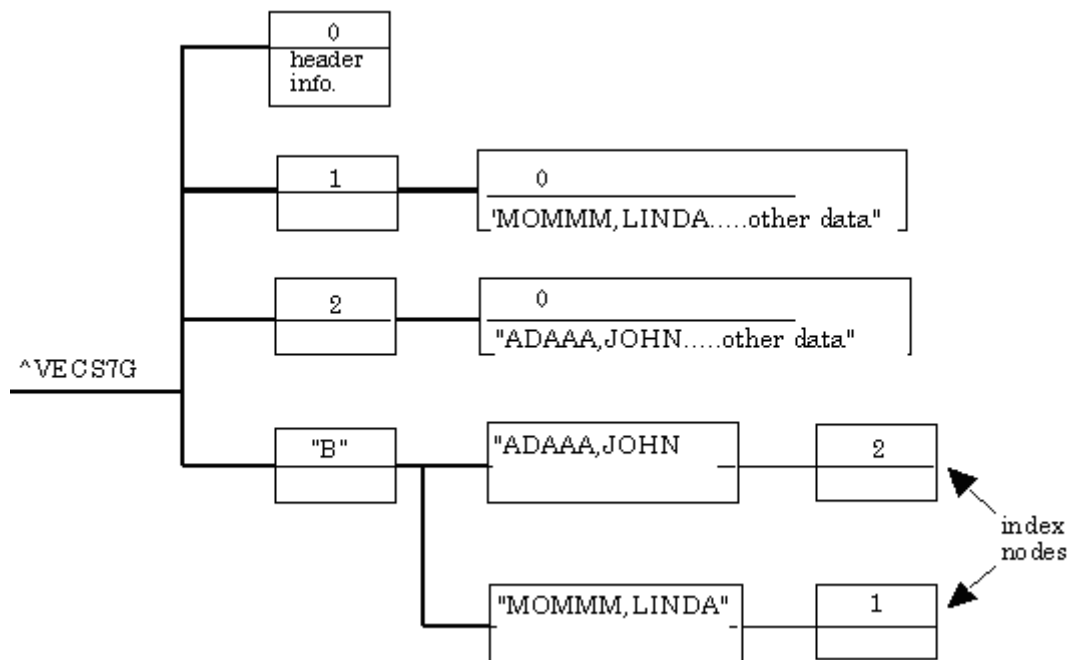
```
^VECS7G("B",name,record number)=""
```

We will call this the "B" index. The first level subscript "B" is chosen to ensure that the index nodes are at the end of the ^VECS7G file (remember with canonic ordering, alphas will follow numbers). This node has no value; it is set to null. It's only purpose is to link the name with the record number. Once we have the record number, we can access the data node directly to obtain the rest of the information associated stored with that name.



VA Programming Standards and Conventions

All globals except for ^UTILITY and ^TMP must be VA FileMan-compatible.



^VECS7G(0)=file name^file number^last record accessed^number of entries

^VECS7G(record number,0)=name^address^city^state^zip^phone^last contact date

```
^VECS7G("B",name,record number)=""
```

VA M PROGRAMMING

Intermediate

This is what the file would look like if we listed the global.

M Code Sample

```
^VECS7G(0) = MAILING LIST^1^2^2
^VECS7G(1,0) = MOMMM,LINDA^123 Main St.^Anywhere^PA^11111^1112223333^2900101
^VECS7G(2,0) = ADAAA,JOHN^Box 103^Nowhere^GA^22222^9998887777^2910214
^VECS7G("B", "ADAAA,JOHN", 2) =
^VECS7G("B", "MOMMM,LINDA", 1) =
```

A schematic of the new global structure including the index nodes with the first level B subscript is shown on the previous page. The structure and contents of the header node and the data nodes are also shown.

A listing of the global with fictitious data is shown above in the M code sample. Note that the data for John Adams is located at record number 2, the B index for him is listed first.

If the file contained more data, it might appear as the listing shown below. Note the index nodes at the end of the data records. Note also that there are several entries for a bill smith, all of which are different persons and reference different records.

M Code Sample

```
^VECS7G(0) = MAILING LIST^1^1301^11
^VECS7G(101,0) = SMSSSS,BILL^123 ANY ST.^ANYWHERE^PA^11111
^VECS7G(301,0) = ADAAA,JOHN^3 ANY ST.^ANYTOWN^GA^34565
^VECS7G(401,0) = SMSSSS,BILL^RD 4^ANYWHERE^PA^11111
^VECS7G(501,0) = BABBB,RALPH^55 ANYWAY^ANYTOWN^NY^34576
^VECS7G(601,0) = SMSSSS,BILL^RD 10^ANYWHERE^PA^11111
^VECS7G(701,0) = CACCC,BILL^1 ANY AVE.^ANYTOWN^OR^22222
^VECS7G(801,0) = FIFFF,MICKEY^ANY LANE^ANYTOWN^TN^23456
^VECS7G(901,0) = SMSSSS,BILL^RD 6^ANYWHERE^PA^11111
^VECS7G(1001,0) = TATTT,MARY^11 ANY ST^ANYTOWN^PA^15601
^VECS7G(1101,0) = SMSSSS,BILL^123 ANY ST.^ANYWHERE^PA^11111
^VECS7G(1301,0) = WAWWW,RON^RD 6^ANYSIDE^GA^33333
^VECS7G("B", "ADAAA,JOHN", 301) =
^VECS7G("B", "BABBB,RALPH", 501) =
^VECS7G("B", "CACCC,BILL", 701) =
^VECS7G("B", "FIFFF,MICKEY", 801) =
^VECS7G("B", "SMSSSS,BILL", 101) =
^VECS7G("B", "SMSSSS,BILL", 401) =
^VECS7G("B", "SMSSSS,BILL", 601) =
^VECS7G("B", "SMSSSS,BILL", 901) =
^VECS7G("B", "SMSSSS,BILL", 1101) =
^VECS7G("B", "TATTT,MARY", 1001) =
^VECS7G("B", "WAWWW,RON", 1301) =
```

VA M PROGRAMMING

Intermediate

Creating Index Nodes

The following code should look familiar. It is from the previous create routine. The code to add an index node is highlighted in the M code sample.

M Code Sample

```
ADD ;Create a new record
L +^VECS7G(0):5 I '$T W !,"File is busy, try again later." S BUSY=1 Q
I '$D(^VECS7G(0)) S ^VECS7G(0)="MAILING LIST^1^0^0"
F RECNR=$P(^VECS7G(0),"^",3)+1:1 I '$D(^VECS7G(RECNR)) Q
S $P(^VECS7G(0),"^",3,4)=RECNR_"^"_$P(^VECS7G(0),"^",4)+1)
S ^VECS7G(RECNR,0)=NAME,^VECS7G("B",NAME,RECNR)=" L -^VECS7G(0)
W !,"Record # ",RECNR,!
Q
```



Using an Index Node

We can use the index node to determine if there already is an entry with the name entered by the user. The first code sample uses \$DATA to determine if the name entered by the user is in the B index.





M Code Sample

```
I $D(^VECS7G("B",NAME)) W !,"Record with this name "
I W "already exists, if you wish to add, enter name "
I W "enclosed",!," in quotes",! S XDUP=1 Q
```

Lets examine the code we used in Lesson 6 to do this search.

There are lines of code that follow that must be typed on one statement but are too long to fit on one line on the screen. The arrow symbols  and  will be used throughout this lesson to indicate that the code should be read as if it were on one line.

M Code Sample

```
F RECNR=0:0 S RECNR=$O(^VECS7G(RECNR)) Q:RECNR'>0 I 
$D(^VECS7G(RECNR,0)), $P(^VECS7G(RECNR,0),"^",1)=NAME 
S RECORD=^VECS7G(RECNR,0) Q
```

The \$ORDER in this sample will start at the beginning of the file and continue until the first match is found or the end of the file is reached. If the file contained 10,000 records and there were no matches, we would have to check all 10,000 records anyway!

VA M PROGRAMMING

Intermediate

Now, we will use the index to look for a name match. Every time we find a match, the MATCH subroutine will be called to save the individual's record number. We'll see more about this in detail later.

M Code Sample

```
F RECNR=0:0 S RECNR=$O(^VECS7G("B",NAME,RECNR)) Q:RECNR'>0 D MATCH
```

In this line of code, we begin the \$ORDER at the node in the "B" index with the name entered by the user. We end the \$ORDER when the name no longer matches. For every match, the MATCH subroutine is called. Therefore, if the file contained 10,000 individuals but only five Bill Smiths, we would access only five entries in the "B" index, find five matches and store five record numbers. It would not be necessary to examine all 10,000 data nodes to find the appropriate Bill Smith as was required by the technique shown on the preceding page.

FOR Loop with Index Nodes

\$ORDERing Through the Data Nodes

You may have noticed in the preceding screen that the FOR loop looks different than the first time we saw it in Lesson 5:

Lesson 5 M Code Sample (using the variable RECNR):

```
S RECNR=" "
F X=1:1 S RECNR=$ORDER(^VECS5G(RECNR)) Q:RECNR=" " D PR
```

Lesson 6 and Lesson 7 M Code Sample (using the variable RECNR):

```
F RECNR=0:0 S RECNR=$ORDER(^VECS7G(RECNR)) Q:RECNR'>0 D PR
```

The goal of both FOR loops is to \$ORDER through the data nodes only. In Lesson 5 we did not have index nodes, so we could QUIT when RECNR="". However, in this lesson we have added index nodes at the end of the data nodes. If we used the Lesson 5 code with Q:RECNR="", we would not only access the data nodes but would continue into and through the index nodes.

The quit logic of the Lesson 6 and 7 FOR loop prevents entry into the index nodes by using Q:RECNR'>0.

VA M PROGRAMMING

Intermediate

In Lesson 6 and 7 the data node used RECNR. The ordering through the data node begins with the first record number after zero and stops when the record number is greater than zero, as is the case when the value B is returned when the first index node is encountered. The numeric interpretation of B is zero and since zero is not greater than zero, the QUIT command is executed.

Here's is an example of each structure:

Lesson 5: ^VECSG("Mommm, Linda")=address^city^state^zip

The \$ORDER will stop when there are no more nodes (NAME="") at that time.

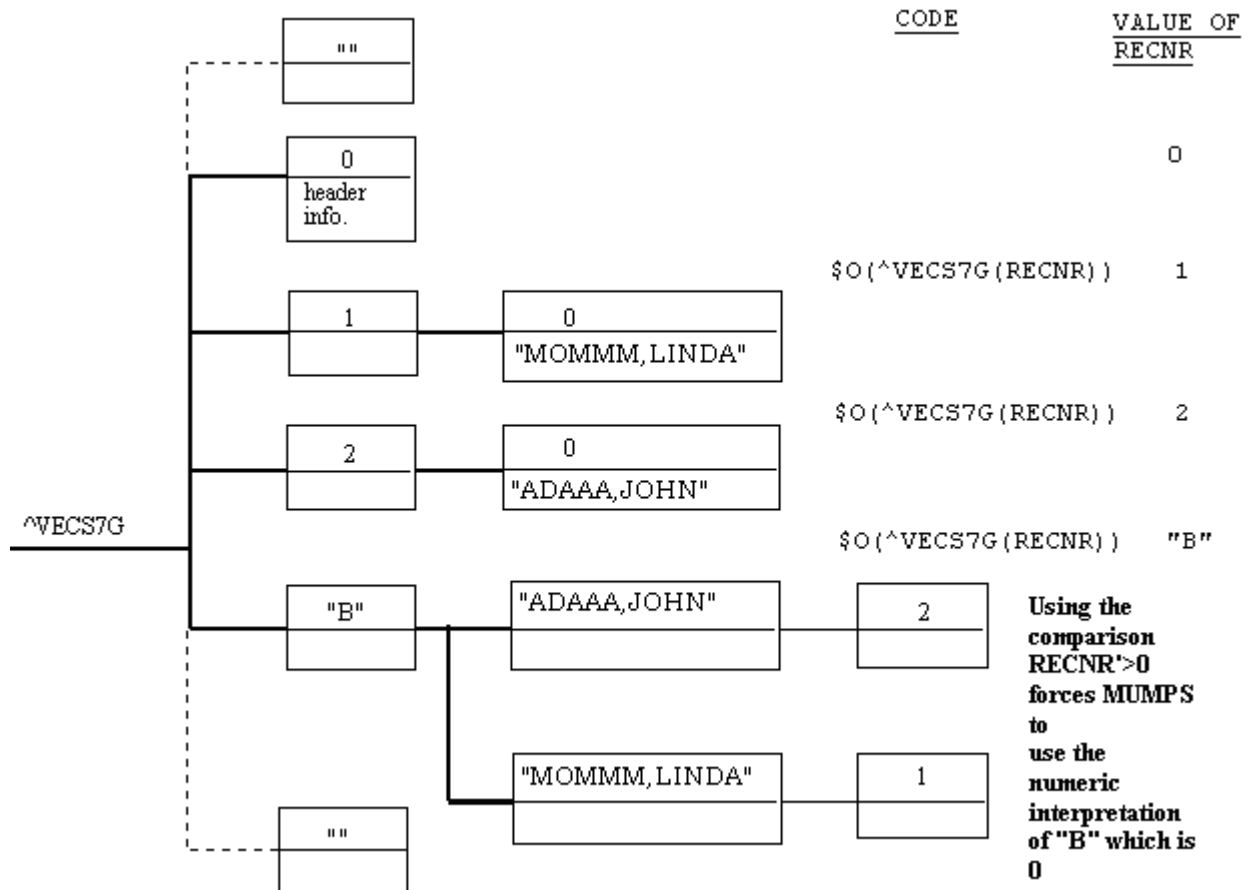
Lesson 6 & 7: ^VECSG(1)=Mommm, Linda^address^city^state^zip
 ^VECSG("B","Mommm, Linda",1)=""

We want to stop when the "B" subscript is returned. Therefore, we change the M code to say QUIT when RECNR is not greater than 0. The variable RECNR will be equal to "B" when the index node is encountered. The numeric interpretation of B is 0, therefore, 0 is not greater than 0 and the QUIT command is executed.

VA M PROGRAMMING

Intermediate

Shown below is a diagram of the file with data nodes and B index nodes. At the bottom of the diagram is a line of M code that could be used to order through the data nodes. In the rightmost column of the diagram, the value of the variable RECORD NUMBER is given for each iteration of the \$ORDER function as it processes the data nodes.



M Code Sample

```
F RECNR=0:0 S RECNR=$ORDER(^VECS7G(RECNR)) Q:RECNR'>0 D PR
```

The initial value is null and increments by one as it steps to each data node. When the \$ORDER steps beyond the last record, the value of RECORD NUMBER becomes capital B because the node after the last data node is the first entry in the B index.

The QUIT in the FOR loop is conditioned on an arithmetic expression; therefore, RECORD NUMBER which is now B, is interpreted as zero. Since zero is not greater than zero, the QUIT is executed and the FOR loop is exited.

VA M PROGRAMMING

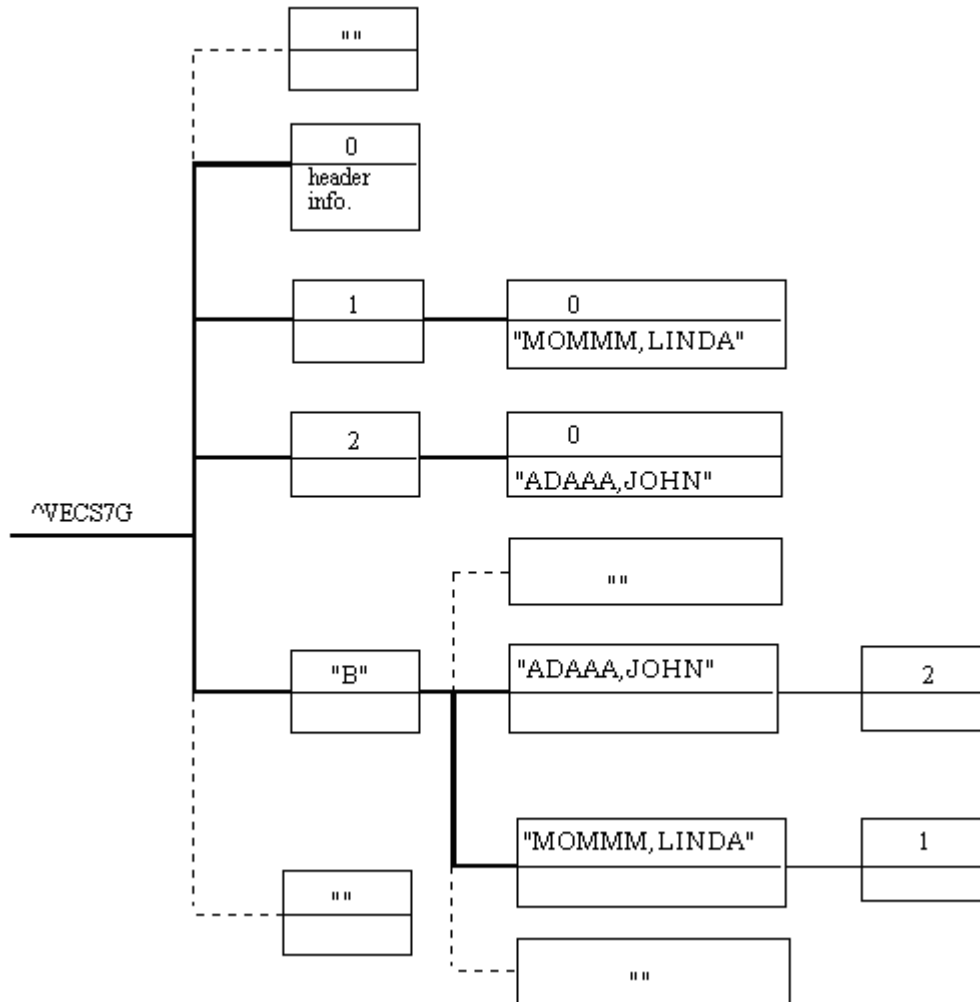
Intermediate

\$ORDERing Through the Index Nodes

Let's look at a practical use of the "B" index. When we wish to print all the records in the file in order by name, we can use the "B" index nodes since they are stored in alphabetical order by the second subscript, NAME. Using the diagram of ^VECS7G to illustrate the process, we might write the following FOR loop.

M Code Sample

```
SET NAME=" "  
FOR S NAME=$O(^VECS7G("B",NAME)) Q:NAME=" " DO PR
```

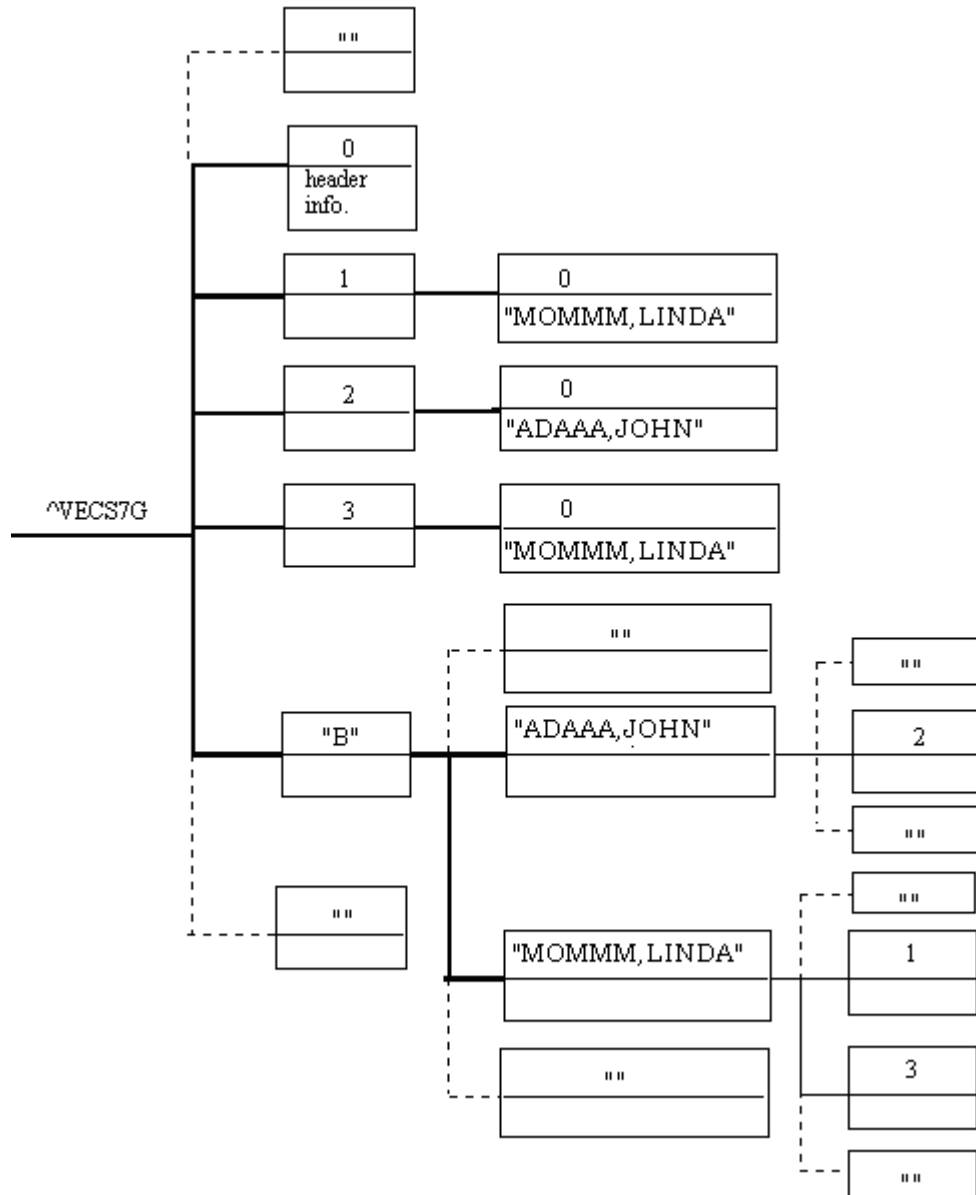


If the PR subroutine just printed the values of the field NAME, the output from the above FOR loop would be:

```
ADAAA,JOHN  
MOMMM,LINDA
```

Intermediate

What was done on the previous page would work only if there were no duplicate names in the file. We have added another MOMMM,LINDA record to our file. The schematic below now includes detail for the RECORD NUMBER in the B index. This will be needed to correctly process the file when duplicates are present. This will be discussed see in pages to follow.



VA M PROGRAMMING

Intermediate

If we used the FOR loop we developed previously, we would get exactly the same output from this file as we did with the one with no duplicates! The M code from one of the previous examples is repeated below.

M Code Sample

```
SET NAME=" "  
FOR S NAME=$O(^VECS7G("B",NAME)) Q:NAME=" " DO PR
```

The problem is that we would not be able to tell that there are duplicate records. As you looked at the diagram on the preceding page, you saw that even though we have one "B" index node with "MOMMM,LINDA" at the second level, there are two third-level nodes containing the record numbers of the duplicate records. We have to get to the third level to even recognize that duplicates are present and to access the unique record numbers for each of the duplicates. With the record number we gain access to the other fields in the data nodes.

Turn back to the previous page if you wish to view the diagram again.

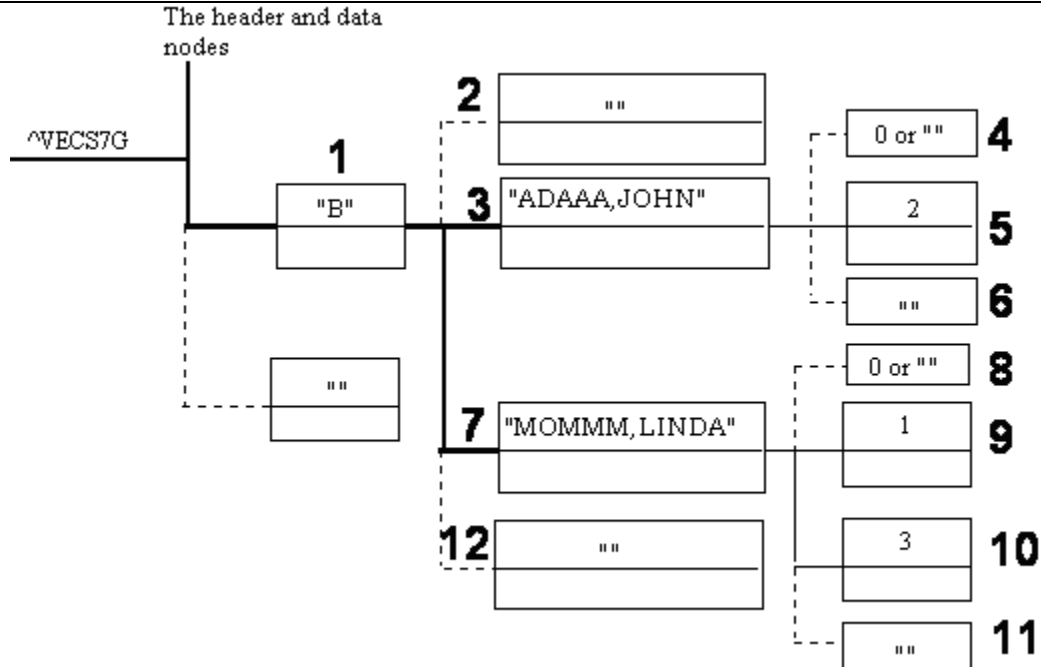
The M code below is necessary to access the record numbers at the third level of the "B" index. The diagram on the next page shows only the "B" index branch. Notice that there are imaginary nodes at the beginning and end of the second level. Also, under each name, there are imaginary nodes at the beginning and end of the third level. The numbers on the diagram show the order in which M will "visit" each node.

M Code Sample

```
SECOND ;  
    S NAME=" "  
    F S NAME=$O(^VECS7G("B",NAME)) Q:NAME=" " D THIRD  
    Q  
THIRD ;  
    F RECNR=0:0 S RECNR=$O(^VECS7G("B",NAME,RECNR)) Q:RECNR=" " DO DISPLAY  
    Q  
DISPLAY ;  
    W !,NAME,?30,RECNR  
    ; You can use the RECNR to access the data node  
    Q
```

VA M PROGRAMMING

Intermediate



Reducing Redundancy

New Create Routine

One of the difficulties with the create routine developed in the previous lesson is that duplicate names could be added accidentally; the second occurrence of the same name would be stored in a separate node whether or not that was the intent.

In this lesson, we will add two features to the create routine that will address this problem.

Adding Duplicate Names

The first new feature to be added is a check to determine if the name entered already exists. This will be accomplished by comparing the entry to the "B" index of the file to see if the name is present in the second level subscript. The following code will do this. If the name is found in the B index, a message is printed and a variable is SET for later use.

M Code Sample

```
I $D(^VECS7G("B",NAME)) W !,"Record with this name "  
I W "already exists, if you wish to add, enter name "  
I W "enclosed",!," in quotes",! S XDUP=1 Q
```

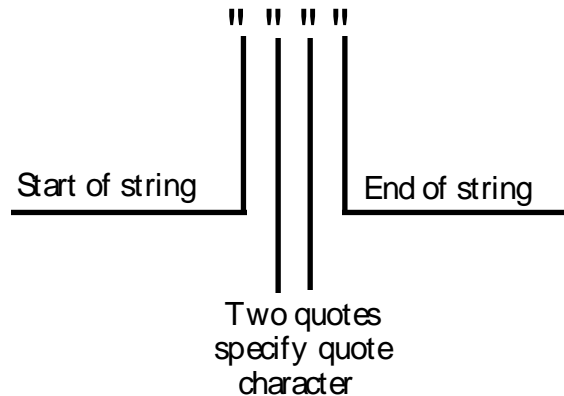
VA M PROGRAMMING

Intermediate

The second feature to be added will permit the user to enter a duplicate name if quotes (") are placed around the name. The M code in the sample below checks for the existence of quotes in the name entered. The diagram explains why four quotes are necessary to get M to look for one in the NAME string.

M Code Sample

```
I NAME[ " " " " S NAME=$P(NAME, " " " ", 2) G CHKEXIT
```



In order for M to recognize that a quote is to be included as a character in the rest of the string, it has to know that the quote is not meant to signify either the beginning or the end of the string. The language uses two quotes to place a single quote somewhere within a string. In other words, the first quote signifies the beginning of the string. The next two quotes specify one quote character. The fourth quote signifies the end of the string.

Pseudocode for VECS7CRI, the Create Routine

The next page shows the pseudocode for the create routine. The main part of the program is steps one through six. Step seven is the subroutine to add a name to the file creating both the data node and the B index node. Step eight is the subroutine that checks for the existence of a duplicate name.

You will want to review this pseudocode carefully before proceeding to the pages that show the complete M code.

VA M PROGRAMMING

Intermediate

<i>Line Labels</i>	<i>Step</i>
START+1 and +2 ASK+1	1. Clear the screen and write a heading.
ASK+2	2. Prompt the user to enter the name. If the user timed out or entered an ^ or pressed Enter, go to the Exit paragraph (Step 6).
ASK+3	3. Initialize a flag that will be used to indicate that the entry is a duplicate name (XDUP). Initialize a flag that will indicate that the name is valid (OK). Do the CHECK Subroutine (Step 8). If the entry was a duplicate or the name was not valid, go back to Step 2.
ASK+4 and +5 EXIT+1 and +2	4. Set a flag that will be used to indicate that the file is already locked (BUSY). Do the ADD Subroutine (Step 7). If the file is already locked, go back to Step 2.
	5. Do the Edit-fields Routine (which is a separate routine). Go back to Step 2.
	6. The Exit paragraph: KILL any variables and QUIT.
	7. The ADD Subroutine:
ADD+1	a. Lock the header node so no one else can edit it at the same time. If the file is already locked, wait five seconds. If you timeout, set the BUSY flag and QUIT.
ADD+7	b. If the file does not exist, create the header record with the file name (MAILING LIST), file number (1), 0 last record, 0 nr. of entries
ADD+8	c. Find the next available record number (same as Lesson 6)
ADD+9	d. Update the header with the new record number as the last record number (piece 3) and add 1 to the number of entries (piece 4)
ADD+10	e. Put the name that was entered into the node for this record number, create a "B" index node for this record number; unlock the file
ADD+11 & +12	f. Write the record number that has been assigned and QUIT
	8. The CHECK Subroutine:
CHECK+1	a. If the name that the user entered contains quotes ("), the user wishes to add a name that already exists in the file. Remove the quotes and store the result back into the name. Go to Step 8c
CHECK+2 Thru +4 CHKEXIT+1 thru +3 CHKEXIT+4	b. If a cross-reference node exists for this record, issue a message to the user. Set the XDUP flag to 1 to indicate that the name is a duplicate
	c. If the name is not the correct pattern or is too long or the user typed a ?, issue a helpful message. Set the OK flag to 0 to indicate that the name is not okay
	d. QUIT

VA M PROGRAMMING

Intermediate

M Code Sample for VECS7CRI, the Create Routine

This page displays the M code for the create routine and includes the two new added features. You may want to take some time to study this program before proceeding. If there is code that you do not understand, take time to return to the pseudocode for the create routine, on the previous page.

```
VECS7CRI ;ATL/ACE PROGRAMMER--MAILING LIST SYSTEM; 12/1/90
; ;1
;
; VARIABLE LIST
;
; NAME.....Name as entered by the user
; RECNR..... Record number
; DTIME.....Delay time (not to be KILLed)
; XDUP.....flag used to show that name entered is
;           a duplicate (XDUP=1 means that record
;           is a duplicate)
; BUSY.....flag used to show that file is busy
;           (BUSY=1 means the file is busy)
; OK.....flag used to show that the NAME is valid
;           (OK=1 means NAME is valid)
START ;
W @IOF
W !!,?20,"Create Mailing List Entry"
ASK ;
R !!,"Enter NAME: ",NAME:DTIME I '$T!(NAME["^")!(NAME="") G EXIT
S OK=1,XDUP=0 D CHECK G:'OK!(XDUP) ASK
S BUSY=0 D ADD G:BUSY ASK
D ^VECS7FLD
G ASK
EXIT ;
K OK,BUSY,XDUP,NAME,RECNR
Q
ADD ;Create a new record
L +^VECS7G(0):5 I '$T W !,"File is busy, try again later." S BUSY=1 Q
; There are a few changes in this paragraph other than
; adding the code to create the "B" index. A few of
; the global references below use the naked reference
; which is discussed later in this lesson
I '$D(^VECS7G(0)) S ^(0)="MAILING LIST^1^0^0"
F RECNR=$P(^VECS7G(0),"^",3)+1:1 I '$D(^RECNR)) Q
S $P(^VECS7G(0),"^",3,4)=RECNR_"^"_$P(^0,"^",4)+1
S ^VECS7G(RECNR,0)=NAME,^VECS7G("B",NAME,RECNR)=" L -^VECS7G(0)
W !,"Record # ",RECNR,!
Q
```

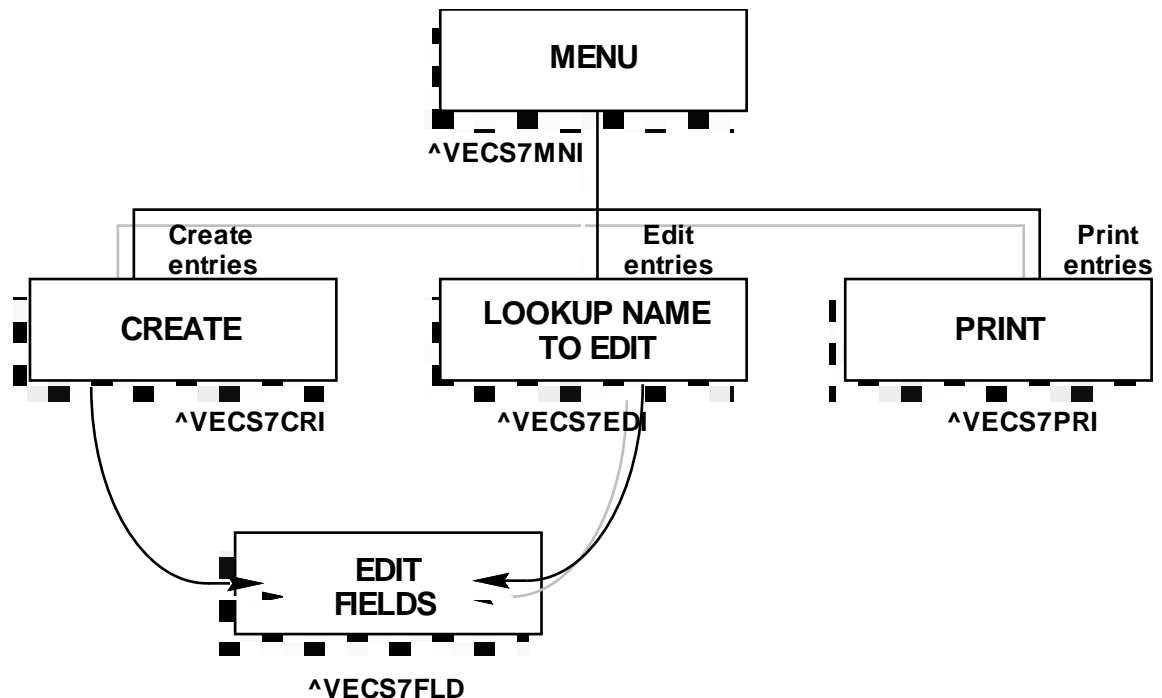
VA M PROGRAMMING

Intermediate

```
CHECK      ;
            I NAME[""] S NAME=$P(NAME,"",2) G CHKEXIT
            I $D(^VECS7G("B",NAME)) W !,"Record with this name "
            I W "already exists, if you wish to add, enter name "
            I W "enclosed",!," in quotes",! S XDUP=1 Q
CHKEXIT    ;
            I NAME'?1U.U1", "1U.UP!($L(NAME)>30)!(NAME["?")
            I W !,"Enter the name of the desired entry.  "
            I W "Enter last name,first name.",! S OK=0
            Q
```

New Lookup-Names-to-Edit Routine

In Lesson 5, there were separate Create, Delete and Edit routines. As was mentioned in the first section of this lesson, we will reduce code redundancy by putting the validation code in a separate routine, which will be called Edit-fields, and it will be accessed by both the Create and Edit functions from the menu. The Delete function will be included in this new routine and will no longer be directly accessible from the menu. Also the old Edit routine will be replaced with a new routine that will be called Lookup Names to Edit. The schematic below outlines the general structure of the system.



VA M PROGRAMMING

Intermediate

Selecting from Duplicate Names

As we mentioned, in the Create routine we have added the ability to add duplicate records. We have also added the ability to have the system list any duplicate names and allow us to select the appropriate one to edit. For example, if the data file contained 5 records for different Bill Smsss, the user would see a list of the five names, with line numbers (phone numbers have been added to help differentiate between the records), as shown in the first M code sample. The user is asked to choose an entry by entering the line number.

If there were more than five Bill Smsss in the file, the user would see the first five listed and be asked to make a selection, as shown in the second M code sample. If the user chose to press enter, not selecting one of the first five entries, the system would then displays the next group of up to five more names. Again, the system would ask the user to choose an entry.

M Code Sample

```
Select MAILING LIST NAME: SMSSS,BILL
1)      SMSSS,BILL          666-111-1111
2)      SMSSS,BILL          666-333-3333
3)      SMSSS,BILL          666-222-2222
4)      SMSSS,BILL          666-555-5555
5)      SMSSS,BILL          666-777-7777
CHOOSE 1 - 5:
```

If there were more than five Bill Smsss in the file, the user would see:

M Code Sample

```
Select MAILING LIST NAME: SMSSS,BILL
1)      SMSSS,BILL          666-111-1111
2)      SMSSS,BILL          666-333-3333
3)      SMSSS,BILL          666-222-2222
4)      SMSSS,BILL          666-555-5555
5)      SMSSS,BILL          666-777-7777
CHOOSE 1 - 5:
```

If a record is not selected and the user presses <ENTER>, the system displays:

```
6)      SMSSS,BILL          666-999-9999
CHOOSE 1 - 6:
```

VA M PROGRAMMING

Intermediate

Pseudocode for VECS7EDI, the Lookup-Names-to-Edit Routine

This page and the next show the pseudocode for the new Lookup Names to Edit routine. Please review these pages carefully before proceeding to the M code for the routine.

<i>Line Labels</i>	<i>Step</i>
START+1 and +2	1. Clear the screen and display a heading.
ASK+1 and +2	2. Prompt the user to enter the name. If the user timed out or entered an ^ or pressed Enter, go to the Exit paragraph (Step 5).
ASK+3 thru +5	3. If the name is not the correct pattern or is too long or the user typed a ?, issue a helpful message. Go back to Step 2.
ASK+6	4. Do the LOOKUP Subroutine (Step 6). Do the Edit-fields Routine (which is a separate routine) if the user chose a valid record to edit (the record number is greater than 0). Go back to Step 2.
EXIT+1 and +2	5. The Exit paragraph: KILL any variables and QUIT.
LOOKUP+1	6. The LOOKUP subroutine:
LOOKUP+2	a. Initialize the count of the number of records that match the name entered by the user
LOOKUP+3	b. On a FOR loop, go through the "B" index file using the name the user entered and searching by the record number. QUIT the FOR loop when there are no more entries for that name (RECNr>0). Do the MATCH Subroutine (Step 7)
LOOKUP+4	c. Initialize the variable that will hold the user's choice from a list of duplicates
LOOKUP+5	d. If there was only one match of the name, set the NAMES subscript to 1
LOOKUP+6	e. If there was more than one match, do the CHOOSE Subroutine, Step 8
LOOKUP+7	f. If there is an entry in the NAMES array for the choice the user made, transfer the NAMES array record number entry to a variable (RECNr)
LOOKUP+8	g. If there was no match, ring the bell and display ??
	h. QUIT
	7. The MATCH Subroutine:
MATCH+1	a. Set the new NAMES array subscript COUNT equal to the last value COUNT + 1
MATCH+2	b. Put the record number into the first level node of NAMES, put the data into a second level node
MATCH+3	c. QUIT
	8. The CHOOSE Subroutine:



VA M PROGRAMMING

Intermediate

CHOOSE+1	a.	On a FOR loop that goes through the NAMES array, write a selection number, name, and phone number. If this is the end of a group of 5 names or we're at the end of the NAMES array, DO the REPLY Subroutine (Step 9). QUIT the FOR loop if the user enters a selection number
CHOOSE+2	b.	QUIT
	9.	The REPLY Subroutine:
REPLY+1 thru +3	a.	Prompt the user to enter a selection number. If the selection entry is ^ or we timed out, go to the Reply exit, Step 9c.
REPLY+4 and +5	b.	If the user types a selection number that does not exist, issue an error message and go back to Step 9a
CHEXIT+1	c.	QUIT

M Code for VECS7EDI, the Lookup-Names-to-Edit Routine

This page and the next page shows the M code for the new Lookup Names to Edit routine. Study it carefully and note the comments in the box regarding the use of the CONTAINS operator. If there is M code that you do not understand, please take time to return to the previous pages and review the pseudocode.

There are lines of code that follow that must be typed on one statement but are too long to fit on one line on the screen. The arrow symbols  and  will be used to indicate that the code should be read as if it were on one line.

```
VECS7EDI           ;ATL/ACE PROGRAMMER--MAILING LIST SYSTEM;
12/1/90
    ; ; 1
    ; VARIABLE LIST
    ;
    ; NAME....Name entered by user
    ; RECNR...Record number
    ; COUNT...Number of matches to name user entered
    ; CHOICE..If there were duplicate names, number
    ;           of choice user entered
    ; NAMES().Array to hold duplicate names
    ; CH.....Loop counter to number selections
    ;
START      ;
W @IOF
W !,?20,"Edit Mailing List Entries"
```

VA M PROGRAMMING

Intermediate

```

ASK      ;
        R !!, "Select MAILING LIST NAME: ", NAME: DTIME
        I '$T!(NAME["^")!(NAME="") G EXIT
        I NAME'?1U.U1", "1U.UP!($L(NAME)>30)!(NAME["?")
        I W !, "Enter the name of the desired entry.  "
        I W "Enter last name, first name.", ! G ASK
        D LOOKUP, ^VECS7FLD: RECNR>0 G ASK

EXIT     ;
        K NAME, CHOICE, CH, COUNT, NAMES, RECNR
        Q

LOOKUP   ;
        S COUNT=0
        F RECNR=0:0 S RECNR=$O(^VECS7G("B", NAME, RECNR))
        Q: RECNR'>0 DO MATCH
        S CHOICE=""
        I COUNT=1 S CHOICE=1
        I COUNT>1 D CHOOSE
        I $D(NAMES(+CHOICE)) S RECNR=NAMES(CHOICE)
        I RECNR'>0 W $C(7), "  ??", !
        Q

MATCH    ;
        S COUNT=COUNT+1
        S NAMES(COUNT)=RECNR, NAMES(COUNT, 0)=^VECS7G(RECNR, 0)
        Q

CHOOSE   ;
        F CH=1:1:COUNT W !, CH, " ) ", $P(NAMES(CH, 0), "^", 1)
        , ?40, $P(NAMES(CH, 0), "^", 6) I CH#5=0!
        (CH=COUNT) D REPLY Q: CHOICE'=""
        Q

REPLY    ;
        W !, "CHOOSE 1 - ", CH, " or press RETURN to continue: "
        R CHOICE: DTIME

```

```

; We want to see if the user entered an ^ or pressed
; the Enter ; key. To understand this "trick" we
; must remember that a null is contained in
; EVERYTHING (even an "^"). That means, that
; ("^[CHOICE) is the same thing as
; (CHOICE["^")!(CHOICE="").
;

```

```

        I '$T!("[^"[CHOICE) G CHEXIT
        ;
        I '$D(NAMES(CHOICE)) W !, ?10, "Must be a number "
        I W "from 1 through ", CH, !, $C(7) G REPLY

CHEXIT   ;
        Q

```

VA M PROGRAMMING

Intermediate

New Edit-Fields Routine

The new Edit-fields routine is basically the old Edit routine without the code to look up a record and with the additions of code to delete a record or a field.

Deleting Records

We will now look at specific segments of the new Edit-fields routine. The first segment to be examined is the code to delete a record. The user will be instructed to enter an @ (at sign) at the NAME prompt to delete an entire record. Shown below is the segment of the code that will accept input at the Name prompt and delete the record if the an "at sign" (@)is typed.

M Code Sample

```
NAME      ;
W !, "NAME:  ", $S(NAME="": "", 1:NAME_"/ /") R INDATA:DTIME
I '$T!(INDATA["^") G EDEXIT
G:INDATA="" ADDR
I INDATA'="@", INDATA'?1U.U1", "1U.UP!($L(INDATA)>30)!(INDATA["?")
I W !, "Enter the name of the desired entry.  "
I W "Enter last name, first name.", ! G NAME
I INDATA="@ " S OK=0 D DELETE G EXIT:OK, NAME
S NAME=INDATA
.
.
.
DELETE    ;
L +^VECS7G(0):5
I '$T W !, "File is busy, try again later" Q
W !, "Sure you want to delete the entire '", NAME, "' MAILING LIST ? "
R ANS:DTIME I '$T!(ANS["^")!(ANS="") G DELEXIT
I (ANS'="Y"), (ANS'="YES") W !, " ... not deleted !", $C(7), ! G DELEXIT
K ^VECS7G(RECNR), ^VECS7G("B", NAME, RECNR)
S $P(^VECS7G(0), "^", 4)=$P(^VECS7G(0), "^", 4)-1 L -^VECS7G(0)
W !, "  '", NAME, "' DELETED !", ! S OK=1
DELEXIT ;
Q
```

VA M PROGRAMMING

Intermediate

Deleting Fields

The new Edit-fields routine will also allow deleting of one or more fields in a selected record. The user will be instructed to enter the @ for any field prompt to delete that field. The first code sample shows the code necessary to delete the address field.

M Code Sample

```
ADDR      ;
          W !,"ADDRESS: ", $$S(ADDR="": "", 1:ADDR_"/") R INDATA:DTIME
          I '$T!(INDATA["^") G EDEXIT
          G:INDATA="" CITY
          I INDATA'="@",INDATA'?.ANP!($L(INDATA)>25)!(INDATA["?")
          I W !,"Address must not exceed 25 characters.",!,$C(7) G ADDR
          I INDATA="@ " D DELFLD I INDATA'="" G ADDR
          S ADDR=INDATA
          .
          .
          .
DELFLD    ;
          R !,"Sure you wish to delete ? ",ANS:DTIME
          I '$T!(ANS["^")!(ANS="") GOTO DELFEXIT
          I (ANS="Y")!(ANS="YES") S INDATA=""
          I W " ... <FIELD DELETED>"
          E W " ... <FIELD NOT DELETED>"
DELFEXIT  ;
          QUIT
```

NOTE: Notice that the validation segments in the M code above, such as that beginning at the line label ADDR, are different in structure than in the M code below, that was contained in Lesson 6. Can you find the difference(s)?

Lesson 6 M Code Sample

```
ADDR      ;
          W !,"ADDRESS: ", $$S(ADDR="": "", 1:ADDR_"/") R INDATA:DTIME
          I '$T!(INDATA["^") G EDEXIT
          I INDATA'="" ,INDATA'?.ANP!($L(INDATA)>25)!(INDATA["?")
          I W !,"Address must not exceed 25 characters.",!,$C(7) G ADDR
          S:INDATA'="" ADDR=INDATA
```

VA M PROGRAMMING

Intermediate





VA Programming Standards and Conventions

It is a VA Programming Standard that if the user wishes to delete the value of a data element or a record, entering the at-sign (@) must cause deletion, if deletion is permitted.

M Code for the Edit-Fields Routine

The next few pages present the entire M code for the new Edit-fields routine. Note that the pseudocode for this new routine is not shown in this lesson since it is very lengthy and is similar to the Edit subroutine of the old Edit routine presented in Lesson 5.

There are lines of code that follow that must be typed on one statement but are too long to fit on one line on the screen. The symbols  and  will be used to indicate that the code should be read as if it were on one line.

M Code Sample

```
VECS7FLD ;ATL/ACE PROGRAMMER--MAILING LIST SYSTEM; 12/1/90
;;1
; VARIABLE LIST
; RECNR.....Record number
; RECORD.....copy of data node
; NAME.....Name as stored in the record
; OLDNAME.....Original name as stored in the record
; ADDR.....Address
; CITY.....City
; STATE.....State
; ZIP.....Zip code
; PHONE.....Phone number
; LCD.....Internal form of last contact date
; XDATE.....Display form of date as stored in node
; ANS.....User's answer to delete? questions
; INDATA.....Holds users input before validation
; OK.....Flag to designate if user is deleting
;           record (OK=1 says record is to be
;           deleted)
```

VA M PROGRAMMING

Intermediate

```

START      ;
           ;Edit data fields
L +^VECS7G(RECNR,0):5
I '$T W !,"File is busy, try again later" Q
S RECORD=^VECS7G(RECNR,0)
S (OLDNAME,NAME)=$P(RECORD,"^",1),ADDR=$P(RECORD,"^",2)
S CITY=$P(RECORD,"^",3)
S STATE=$P(RECORD,"^",4),ZIP=$P(RECORD,"^",5)
S PHONE=$P(RECORD,"^",6)
S LCD=$P(RECORD,"^",7)
I LCD="" S XDATE=$E(LCD,4,5)_"_"$E(LCD,6,7)_"_"($E(LCD,1,3)+1700)
I LCD="" S XDATE=""

NAME      ;
W !,"NAME: ", $S(NAME="": "", 1:NAME_"/") R INDATA:DTIME
I '$T!(INDATA["^") G EDEXIT
G:INDATA="" ADDR
I INDATA'="@",INDATA'?1U.U1",1U.UP!($L(INDATA)>30)!(INDATA["?")
I W !,"Enter the name of the desired entry. "
I W "Enter last name,first name.",! G NAME
I INDATA="@ " S OK=0 D DELETE G EXIT:OK,NAME
S NAME=INDATA

ADDR      ;
W !,"ADDRESS: ", $S(ADDR="": "", 1:ADDR_"/") R INDATA:DTIME
I '$T!(INDATA["^") G EDEXIT
G:INDATA="" CITY
I INDATA'="@",INDATA'? .ANP!($L(INDATA)>25)!(INDATA["?")
I W !,"Address must not exceed 25 characters.",!,$C(7) G ADDR
I INDATA="@ " D DELFLD I INDATA'="" G ADDR
S ADDR=INDATA

CITY      ;
W !,"CITY: ", $S(CITY="": "", 1:CITY_"/") R INDATA:DTIME
I '$T!(INDATA["^") G EDEXIT
G:INDATA="" STATE
I INDATA'="@",INDATA'? .ANP!($L(INDATA)>20)!(INDATA["?")
I W !,"City must not exceed 20 characters.",!,$C(7) G CITY
I INDATA="@ " D DELFLD I INDATA'="" G CITY
S CITY=INDATA

STATE     ;
W !,"STATE: ", $S(STATE="": "", 1:STATE_"/") R INDATA:DTIME
I '$T!(INDATA["^") G EDEXIT
G:INDATA="" ZIP
I INDATA'="@",INDATA'?2U W !,"State must be a 2 character "
I W "abbreviation.",!,$C(7) G STATE
I INDATA="@ " D DELFLD I INDATA'="" G STATE
S STATE=INDATA

ZIP       ;
W !,"ZIP: ", $S(ZIP="": "", 1:ZIP_"/") R INDATA:DTIME
I '$T!(INDATA["^") G EDEXIT
G:INDATA="" PHONE
I INDATA'="@",INDATA'?5N&(INDATA'?9N) W !,"Zip must be five"
I W "or nine digits, no hyphen.",!,$C(7) G ZIP
I INDATA="@ " D DELFLD I INDATA'="" G ZIP
S ZIP=INDATA

```

VA M PROGRAMMING

Intermediate

```

PHONE      ;
W !,"PHONE: ",,$S(PHONE="":",1:PHONE_"/") R INDATA:DTIME
I '$T!(INDATA["^") G EDEXIT
G:INDATA="" LCD
I INDATA'="@",INDATA'?3N1"-3N1"-4N&(INDATA'?3N1"-4N)
I W !,"Phone must be entered with or without area code,"
I W "using hyphens.",!,$C(7) G PHONE
I INDATA="@ " D DELFLD I INDATA'="" G PHONE
S PHONE=INDATA

LCD        ;
W !,"LAST CONTACT DATE: ",,$S(XDATE="":",1:XDATE_"/")
R INDATA:DTIME
I '$T!(INDATA["^") G EDEXIT
G:INDATA="" DATE
I INDATA'="@",INDATA'?2N1"/2N1"/4N W !,"Enter a date in the "
I W "format of MM/DD/YYYY, where MM & DD are each two "
I W "digits and YYYY is 4 digits.",!,$C(7) G LCD
I INDATA="@ " D DELFLD G LCD:INDATA'="" I INDATA="" S LCD="" G EDEXIT

DATE       ;
S:INDATA'="" LCD=($E(INDATA,7,10)-1700)_$E(INDATA,1,2)_$E(INDATA,4,5)

EDEXIT     ;
S $P(^VECS7G(RECNR,0),"^",1,7)=NAME_"^"_ADDR_"^"_CITY_"^"_
STATE_"^"_ZIP_"^"_PHONE_"^"_LCD
I OLDNAME'=NAME K ^VECS7G("B",OLDNAME,RECNR)
I S ^VECS7G("B",NAME,RECNR)=" "
L -^VECS7G(RECNR,0)

EXIT        ;
K NAME,ADDR,CITY,STATE,ZIP,PHONE,LCD,XDATE,ANS,INDATA
K OK,OLDNAME,RECNR,RECORD
Q

DELFLD     ;
R !,"Sure you wish to delete ? ",ANS:DTIME
I '$T!(ANS["^")!(ANS="") GOTO DELFEXIT
I (ANS="Y")!(ANS="YES") S INDATA=""
I W " ... <FIELD DELETED>"
E W " ... <FIELD NOT DELETED>"

DELFEXIT   ;
QUIT

DELETE     ;
L +^VECS7G(0):5
I '$T W !,"File is busy, try again later" Q
W !,"Sure you want to delete the entire '",NAME,'" MAILING LIST ? "
R ANS:DTIME I '$T!(ANS["^")!(ANS="") G DELEXIT
I (ANS!="Y"),(ANS!="YES") W !," ... not deleted !",$C(7),! G DELEXIT
K ^VECS7G(RECNR),^VECS7G("B",NAME,RECNR)
S $P(^VECS7G(0),"^",4)=$P(^VECS7G(0),"^",4)-1 L -^VECS7G(0)
W !," '",NAME,'" DELETED !",! S OK=1

DELEXIT    ;
Q

```

VA M PROGRAMMING

Intermediate

Decision Table of the Edit-Fields Routine

Shown below is the decision table used for setting the last contact date (LCD) and storing it in the record. It shows the possible combinations of user input for the contact date (INDATA) and of the current stored value (XDATE). The text in each square shows the action taken for the particular combination of values.

Study this table carefully, for similar tables could be constructed for the other variables stored in the record. The actions taken for the other variables might be somewhat different from those given here for the "last contact date." You may want return to the previous pages and review the code to see how this table was implemented.

		Values of XDATE (current stored value)	
		XDATE="" no current value	XDATE'="" current value
Values of INDATA (user's entry)	INDATA="" (no change entered)	no change	no change
	INDATA'="" (new value entered)	LCD=INDATA no current value, new date stored	LCD=INDATA current value replaced with new date

VA M PROGRAMMING

Intermediate

Routines That Required Minor Modifications

The Menu Routine

The only changes necessary for this routine are to the OPTIONS paragraph, as indicated in the M Code Sample.

M Code Sample

```
VECS7MNI ;ATL/ACE PROGRAMMER--MAILING LIST SYSTEM; 12/1/90
; ;1
;VARIABLE LIST
;
;SPECIAL VARIABLES USED IN VA:
;DTIME.....Delay time (not to be KILLed)
;IOF.....contains the codes to start output
;          at the top of a page or screen
;APPLICATION VARIABLES:
;OPT.....Option number chosen by the user
;ROUTINE...Routine name for option chosen
;ITEMS.....Number of items in menu
;II.....Menu loop counter
;XDESCR.....Text for menu option

START ;
MAIN ;
D DISMENU
I OPT="!"(OPT["^") G EXIT
S ROUTINE=$P($T(OPTIONS+OPT),"^",2,3)
D @ROUTINE G MAIN

EXIT ;
K OPT,ROUTINE,ITEMS,XDESCR,II
Q

DISMENU ;
W @IOF
W !!!,?20,"MAILING LIST SYSTEM"
W !!,"OPTIONS FUNCTIONS"
F II=1:1 D DISMENU2 Q:XDESCR=""
S ITEMS=II-1

DISMENU1 ;
R !!,"ENTER OPTION NUMBER: ",OPT:DTIME
Q:OPT="!"('$T)!(OPT["^")
I (OPT<1)!(OPT>ITEMS) W !!,"OPTION NUMBER MUST BE "
I W "BETWEEN 1 AND ",ITEMS
I R !!,"Press RETURN to continue",OPT:DTIME G DISMENU1
Q

DISMENU2 ;
S XDESCR=$P($T(OPTIONS+II),"^",2)
I XDESCR'="" W !!,II,$P(XDESCR,"^",1)
Q

OPTIONS;
; ;.....CREATE MAILING LIST ENTRY^^VECS7CRI
; ;.....EDIT MAILING LIST ENTRY^^VECS7EDI
; ;.....PRINT MAILING LIST ENTRIES^^VECS7PRI
```

VA M PROGRAMMING

Intermediate

The Print Routine

The only change required for the Print routine is to change the name of the routine for Lesson 7.

M Code Sample

```
VECS7PRI ;ATL/ACE PROGRAMMER--MAILING LIST SYSTEM; 12/1/90
; ;1
; VARIABLE LIST
; RECNr.....Record number assigned in Create
; PAGE.....Page number counter
; IN.....holds response to device number
; ANS.....holds response to "Press enter to continue.."
; NODE.....contents of ^VECS7G(RECNr,0) node
; OUT.....holds response: during display of
;           multiple pages of output, user can press
;           Enter to continue or ^ to stop display
; POP.....holds response: when user is asked to
;           enter a device number, they can enter
;           a number, press Enter, or ^ to quit. If
;           they enter ^, POP gets set to 1
; XDATE.....Holds the display format of the date
; LCD.....Holds the internal format of the date
;
START ;
S PAGE=1,OUT=""
D OPEN G EXIT:POP
U IO
D HEADING
D ORDER
D CLOSE
R !,"Press Enter to go back to the menu",ANS:DTIME
EXIT ;
K RECNr,PAGE,IN,ANS,NODE,OUT,POP,LCD,XDATE
Q
HEADING ;
I PAGE>1,($E(IOST,1,2)="C-")
I R !,"Press Enter to continue or ^ to exit",OUT:DTIME Q:'$T!(OUT["^")
W @IOF
W !!,?30,"MAILING LIST FILE LISTING",?60,"PAGE ",PAGE
S PAGE=PAGE+1
Q
ORDER ;
F RECNr=0:0 S RECNr=$O(^VECS7G(RECNr)) Q:RECNr'>0 D DISPLAY Q:OUT["^"
Q
DISPLAY ;LINE 1
Q:'$D(^VECS7G(RECNr,0))
S NODE=^VECS7G(RECNr,0)
W !!!,$P(NODE,"^",1)
W ?40,"PHONE: ",$P(NODE,"^",6)
LINE2 ;LINE 2
W !,$P(NODE,"^",2)
```

VA M PROGRAMMING

Intermediate

```
LINE3      ;LINE 3
           W !,$P(NODE,"^",3),?$X+2
           W $P(NODE,"^",4),?$X+2
           W $P(NODE,"^",5)
           ; WE WANT THE DATE PRINTED IN DISPLAY FORMAT MM/DD/YYYY
           ; NOT INTERNAL FORMAT YYYYMMDD
           S LCD=$P(NODE,"^",7)
           I LCD="" S XDATE=""
           I LCD'="" S XDATE=$E(LCD,4,5)_"_"$E(LCD,6,7)_"_"($E(LCD,1,3)+1700)
           W ?40,XDATE
           D:$Y+2>IOSL HEADING
           Q
OPEN        ;
           S %ZIS="M" D ^%ZIS
           Q
           ;
CLOSE       ;
           D ^%ZISC
           Q
```

Naked References

A shorthand way of referencing globals is using what is called a ***naked reference*** (or naked syntax). Basically, this means that once a global node has been referenced, it is then possible to make subsequent references to the same global and subscript level using a shortened syntax, omitting the global name and some of the subscripts.

There are two components to the naked reference concept:

- The ***naked indicator***, sometimes called the *naked state*, and
- The ***naked reference*** (also called the *naked syntax*)

The Naked State

Whenever M is executing code and encounters a global reference or a naked reference, a string is automatically created which contains that global reference without the last subscript used in the reference. This incomplete reference is called the naked indicator and is always determined by the most recently executed global reference. Each time a new global reference is made, M resets the naked indicator. M uses the naked indicator along with the naked reference to reconstruct the full global reference.

For example, if the global reference were ^VECS7G(RECNR,0)

Then the naked indicator is set to: ^VECS7G(RECNR,

VA M PROGRAMMING

Intermediate

The Naked Reference

Once the naked indicator has been set by a global reference, a naked reference can be made. The syntax for the naked reference is:

$\wedge(\text{subscript on last level referenced, optional additional subscript(s)})$

Notice that the name of the global is not included. When a naked reference is encountered, M takes the current naked indicator and adds the subscript(s) specified in the naked reference.

Using the same global from the preceding page, suppose the most recent global reference is $\wedge\text{VECS7G}(\text{RECNR}, 0)$,

then the naked indicator is $\wedge\text{VECS7G}(\text{RECNR}, .$

The following table lists examples of naked references and the corresponding expanded reference, the global reference M reconstructs from the naked indicator and the naked reference.

Naked reference

Expanded reference

$\wedge(0)$

$\wedge\text{VECS7G}(\text{RECNR}, 0)$

$\wedge(1)$

$\wedge\text{VECS7G}(\text{RECNR}, 1)$

$\wedge(4, 13)$

$\wedge\text{VECS7G}(\text{RECNR}, 4, 13)$

The last example uses more than one subscript in the naked reference. The expanded reference follows the same rules as with a single subscript, and M appends all the subscripts to the end of the naked indicator. Note that this last example resets the naked indicator.



VA Programming Standards and Conventions

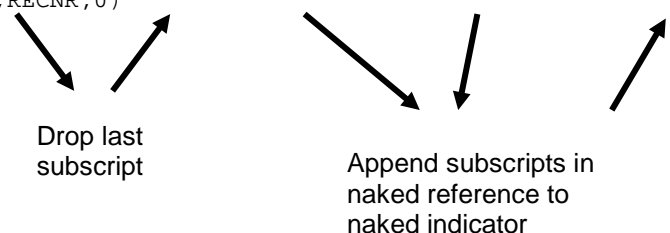
When a naked indicator is not defined on the same line as the corresponding naked reference, it must be documented. Occurrences of this within a routine must be documented in the routine; all other occurrences must be documented in the technical manual. All documentation must include the location that defines the naked indicator.

VA M PROGRAMMING

Intermediate

M Execution of Naked References:

Show below are additional examples of naked references using the global VECA. The arrows at the bottom of the table indicate how the naked indicator is formed and how M reconstructs the full global reference from the naked reference and naked indicator.

<u>Most Recent Reference</u>	<u>Naked Indicator</u>	<u>Naked Reference</u>	<u>Reconstructed Global Reference</u>
<code>^VECA(0)</code>	<code>^VECA(</code>	<code>^(1)</code>	<code>^VECA(1)</code>
<code>^VECA(1,0)</code>	<code>^VECA(1,</code>	<code>^(1)</code>	<code>^VECA(1,1)</code>
<code>^VECA("B",NAME,RECNR)</code> <code>^VECA("B",NAME,RECNR,0)</code>	<code>^VECA("B",NAME,</code>	<code>^(RECNR,0)</code>	
			

We need to be cautious when using naked references. As you see in the diagram in the above, there are particular steps M takes to deal with naked references. M always uses the last executed full global reference to set the naked indicator and then uses this with the naked reference to create the global reference that it executes. The example on the next page shows how this works with the SET statement and why caution is necessary.

VA M PROGRAMMING

Intermediate

Suppose we want to copy record number 1 to record number 2, which doesn't exist yet. We may try the following code:

What the user types is double-underlined.

M Code Sample

```
W $D(^VECS7G(1))           10
W $D(^VECS7G(2))           0
S ^ (2,0)=^(1,0)

D ^%G
Global ^VECS7G
^VECS7G
^VECS7G(0) = MAILING LIST^1^2^2
^VECS7G(1,0) = MOORE,LINDA....
^VECS7G(1,2,0) = MOORE,LINDA....same as ^VECS7G(1,0)
```

In this example \$DATA is used to check the existence of global nodes. Record 1 has descendents but does not exist and record 2 does not exist. The user's intention with the third line of code is to copy node ^ (1,0) into node ^ (2,0). But, this does not work as expected.

The user then lists the global and finds that there is no node VECS7G(2,0)!

When we look at the code above and trace the global references in order of execution, we must remember that M evaluates a SET command by evaluating what is to the right of the equals (=) sign first:

Naked Indicator

```
W $D(^VECS7G(1))           ^VECS7G(
```

The naked indicator is SET in the first WRITE statement.

```
W $D(^VECS7G(2))           ^VECS7G(
```

The second WRITE statement resets the naked indicator, but its value does not change.

VA M PROGRAMMING

Intermediate

S $^{(2,0)} = ^{(1,0)}$ Starting with what is to the right of the = sign, we take the most recent naked indicator ^VECS7G(and attach subscripts found in the naked reference. The full global reference is:
 ^VECS7G(1,0)

The SET statement evaluates the expression on the right side of the equal sign first using the current naked indicator. However, M forms a new global reference.

No problem so far but we must not forget that this global reference now resets the naked indicator to:
 ^VECS7G(1,

Looking at the left side of the = sign, we take the naked indicator and attach the subscripts in the naked syntax. The full global reference becomes:
 ^VECS7G(1,2,0)

Then, M uses this new naked indicator to construct the global reference for the naked reference on the left side of the equal sign. So, the data are copied to the (1,2,0) node instead of the intended, and wanted, (2,0) node.

Naked References in the Sample System

The set of code in the M code sample below is the original code for the ADD subroutine of VECS6CRI, the Create routine.

M Code Sample

```
ADD      ;Create a new record
L +^VECS6G(0):5 I '$T W !,"File is busy, try again later." S BUSY=1 Q
I '$D(^VECS6G(0)) S ^VECS6G(0)="MAILING LIST^1^0^0"
F RECNR=$P(^VECS6G(0),"^",3)+1:1 I '$D(^VECS6G(RECNR)) Q
S $P(^VECS6G(0),"^",3,4)=RECNR_"^"_"($P(^VECS6G(0),"^",4)+1)
S ^VECS6G(RECNR,0)=NAME,^VECS6G("B",NAME,RECNR)=" L -^VECS6G(0)
W !,"Record # ",RECNR,!
Q
```

VA M PROGRAMMING

Intermediate

The set of code in the M code sample below is the same code using naked references on lines ADD+2 through ADD+4.

M Code Sample

```
ADD ;Create a new record
L +^VECS7G(0):5 I '$T W !,"File is busy, try again later." S BUSY=1 Q
I '$D(^VECS7G(0)) S ^(0)="MAILING LIST^1^0^0"
F RECNR=$P(^VECS7G(0),"^",3)+1:1 I '$D(^(RECNR)) Q
S $P(^VECS7G(0),"^",3,4)=RECNR_"^"_$P(^(0),"^",4)+1)
S ^VECS7G(RECNR,0)=NAME,^VECS7G("B",NAME,RECNR)=" L -^VECS7G(0)
W !,"Record # ",RECNR,!
Q
```

This M code sample has been compared with the previous M code sample without naked references. The naked references are shown in bold type in this M code sample.

ANSI M Standard

M is an ANSI Standard Language. This means that on September 15, 1977, the American National Standards Institute recognized a set of M components as the standards of the language. The Standard was then updated in 1984, in 1990, and in 1995.

If a software vendor wants to develop a version of M and market it as Standard M, it must include all the features of the standard. As a computer user and programmer, if you choose to purchase a standard version of M, you can be assured that the code you write can be transported to another standard version of M with little or no modification.

In addition to the standards of the language, several opportunities are built into the standard for tailoring M to fit specific hardware needs. These commands or functions are called implementation specific. In order to use these components, it is necessary to check the reference manual for your implementation of M.

VA M PROGRAMMING

Intermediate

Components of the Language Standard

Shown below is a list of most the M commands that help define the standard. There are also some transaction commands that are part of the standard but will not be used during this course.

M Commands

BREAK	CLOSE	DO	ELSE	FOR	GOTO
HALT	HANG	IF	JOB	KILL	LOCK
MERGE	NEW	OPEN	QUIT	READ	SET
USE	WRITE	XECUTE			

VIEW (Implementation specific)

Z... (Commands that begin with Z are implementation specific)

Shown below is a list of the M standard intrinsic functions that help define the standard.

M Functions

\$ASCII()	\$CHAR()	\$DATA()	\$EXTRACT()
\$FIND()	\$FNUMBER()	\$GET()	\$JUSTIFY()
\$LENGTH()	\$NAME()	\$ORDER()	\$PIECE()
\$QUERY()	\$QLength()	\$QSUBSCRIPT()	\$RANDOM()
\$REVERSE	\$SELECT()	\$STACK()	\$TEXT()
\$TRANSLATE()			

\$VIEW() (Implementation specific)

\$Z...() (Functions that start with \$Z are implementation specific)

Shown below are the special variables that comprise the M standard.

M Special Variables

\$DEVICE	\$ECODE	\$ESTACK	\$ETRAP	\$HOROLOG	\$IO
\$JOB	\$KEY	\$PRINCIPAL	\$QUIT	\$STACK	\$STORAGE
\$SYSTEM	\$TLEVEL	\$TEST	\$TRESTART	\$X	\$Y

\$Z...(Special variables that start with Z are implementation specific)

VA M PROGRAMMING

Intermediate



VA Programming Standards and Conventions

It is a VA Programming Standard that the BREAK, JOB, VIEW, or Z commands, and \$V or \$Z functions will not be used, except by the Kernel. Vendor specific subroutines will not be used, except by the Kernel.



VA Programming Standards and Conventions

It is a VA Programming Convention that the 1995 ANSI M Standard will be adhered to unless modified by an update to the official document VA DHCP Programming Standards and Conventions.

VA Written Utility %INDEX

The routine %INDEX will give you valuable information about a routine or set of routines. In the example below the user would enter the information that is double underlined.

M Code Sample

>D ^%INDEX

C R O S S - R E F E R E N C E R

Routines: VECS7MNI 1 routine found

Routines: VECS7FLD 1 routine found

Routines: VECS7CRI 1 routine found

Routines: VECS7EDI 1 routine found

Routines: VECS7PRI 1 routine found

Routines:

Exclusions: 5 ROUTINES SELECTED.

Select BUILD NAME: <ENTER>

Select PACKAGE NAME: <ENTER>

Print more than compiled errors and warnings? YES//<ENTER>

Print summary only? NO//<ENTER>

Print routines? YES//<ENTER>

Print (R)egular,(S)tructured or (B)oth? R//S

VA M PROGRAMMING

Intermediate

If you choose Structured routine listings, %INDEX will take any line where you have multiple commands and print them on separate lines. %INDEX will also expand any command abbreviations you use.

M Code Sample

Print errors and warnings with each routine? YES//<ENTER>

Index all called routines? NO//YES

I will QUEUE the report if sent to any device other than HOME

DEVICE: PT0;P-DEC RIGHT MARGIN: 132// 80

Shown below is an annotated output generated by the cross referencer.

M Code Sample

```
V. A. C R O S S R E F E R E N C E R  6.0
UCI: MUM,TRN CPU: TRN      FEB 15, 1991@08:16:12
Indexed Routines: 5

VECS7CRI  VECS7FLD  VECS7EDI  VECS7MNI  VECS7PRI
Called Routines

--- CROSS REFERENCING ---
```

```
Compiled list of Errors and Warnings      FEB 15, 1991@08:16:12
```

Any violations of the VA Programming Standards or M syntax will be shown at this point.

M Code Sample

No errors or warnings to report

There are wrap-around lines in the following routines as you saw in the previous sections. However, since %INDEX does not note them in any special way, neither will we!

VA M PROGRAMMING

Intermediate

Shown below and on the next page is an example of a structured program listing using the create routine from our mailing list system.

M Code Sample

--- PRINT STRUCTURED ROUTINE LISTING ---

VECS7CRI 1022 printed FEB 15, 1991@11:29:37

The number to the right of the routine name on the line above is the size of the routine in bytes. If it is over 10,000 bytes, a warning message will be issued.

M Code Sample

VECS7CRI ;ATL/ACE PROGRAMMER--MAILING LIST SYSTEM ;12/1/90

```
+1      ;;1
+2      ;
+3      WRITE @IOF
+4      WRITE !!,?20,"Create Mailing List Entry"
ASK      ;
+1      READ !!,"Enter NAME: ",NAME:DTIME
        IF '$TEST!(NAME["^")!(NAME="")
        GOTO EXIT
+2      SET OK=1
        SET XDUP=0
        DO CHECK
        IF 'OK!(XDUP)
        GOTO ASK
+3      SET BUSY=0
        DO ADD
        IF BUSY
        GOTO ASK
+4      DO ^VECS7FLD
+5      GOTO ASK
EXIT     ;
+1      KILL OK,BUSY,XDUP,NAME,RECNR
+2      QUIT
ADD      ;create a new record
+1      LOCK +^VECS7G(0):5
        IF '$TEST
        WRITE !,"File is busy, try again later."
        SET BUSY=1
        QUIT
+2      IF '$DATA(^VECS7G(0))
```

VA M PROGRAMMING

Intermediate

```
      SET ^(^0)="MAILING LIST^1^0^0"
+3    FOR RECNR=$PIECE(^VECS7G(0),"^",3)+1:1
      IF '$DATA(^(^RCNR))
      QUIT
+4    SET $PIECE(^VECS7G(0),"^",3,4)=RECNR_"^"_$PIECE(^(^0),"^",4)+1)
+5    SET ^VECS7G(RECNR,0)=NAME
      SET ^VECS7G("B",NAME,RECNR)=" "
      LOCK =^VECS7G(0)
+6    WRITE !,"Record # ",RECNR,!
+7    QUIT
CHECK ;
+1    IF NAME[""]
      SET NAME=$PIECE(NAME,"",2)
      GOTO CHKEXIT
+2    IF $DATA(^VECS7G("B",NAME))
      WRITE !,"Record with this name "
+3    IF $TEST
      WRITE "already exists, if you wish to add, enter name "
+4    IF $TEST
      WRITE "enclosed",!," in quotes",!
      SET XDUP=1
      QUIT
CHKEXIT ;
+1    IF NAME'?1U.U1","1U.UP!($LENGTH(NAME)>30)!(NAME["?")
+2    IF $TEST
      WRITE !,"Enter the name of the desired entry.  "
+3    IF $TEST
      WRITE "Enter last name,first name.",!
      SET OK=0
+4    QUIT
```


VA M PROGRAMMING

Intermediate

We have not included the listings of the other routines in the mailing list system. They would be printed in the same format as shown by the examples using VECS7CRI. After all routines are listed, a cross reference for all routines will be printed. This is shown below.

Notice that there is a list of local variables and the routines that use each one. A similar list follows for the global variables. There is also a list of routines that invoke the function \$T. Also, there is a list of external calls for each routine. Finally, a list of routines invoked by other routine is printed.

--- CROSS-REFERENCING ALL ROUTINES ---

*** CROSS REFERENCE OF ALL ROUTINES printed MAY
22,1990@08:04:49 ***

Local Variables	Routines	(>> not killed explicitly)
		(* Changed ! Killed)

>> %ZIS	VECS7PRI*	
ADDR	VECS7FLD*!	
ANS	VECS7FLD*!,VECS7PRI*!	
BUSY	VECS7CRI*!	
CH	VECS7EDI*!	
CHOICE	VECS7EDI*!	
CITY	VECS7FLD*!	
COUNT	VECS7EDI*!	
DTIME	VECS7CRI,VECS7FLD,VECS7EDI,VECS7MNI*,VECS7PRI	
II	VECS7MNI*!	
IN	VECS7PRI!	
INDATA	VECS7FLD*,VECS7FLD*	
IO	VECS7MNI*,VECS7PRI	
>> IO(VECS7MNI*	
IOF	VECS7CRI,VECS7EDI,VECS7MNI*,VECS7PRI	
IOM	VECS7MNI*	
IOSL	VECS7MNI*,VECS7PRI	
IOST	VECS7MNI*,VECS7PRI	
ITEMS	VECS7MNI*!	
LCD	VECS7FLD*!,VECS7PRI*!	
NAME	VECS7CRI*!,VECS7FLD*!,VECS7EDI*!	
NAMES	VECS7EDI!	
>> NAMES(VECS7EDI*	
NODE	VECS7PRI*!	
OK	VECS7CRI*!,VECS7FLD*!	
OLDNAME	VECS7FLD*!	
OPT	VECS7MNI*!	
OUT	VECS7PRI*!	

VA M PROGRAMMING

Intermediate

```
PAGE      VECS7PRI*!
PHONE     VECS7FLD*!
POP       VECS7PRI!
RECNR     VECS7CRI*!,VECS7FLD!,VECS7EDI*!,VECS7PRI*!
RECORD    VECS7FLD*!
ROUTINE   VECS7MNI*!
STATE     VECS7FLD*!
XDATE     VECS7FLD*!,VECS7PRI*!
XDESCR    VECS7MNI*!
XDUP      VECS7CRI*!
ZIP       VECS7FLD*!
```

Global Variables

```
^%ZIS(      VECS7PRI
^VECS7G(     VECS7CRI*,VECS7FLD*!,VECS7EDI,VECS7PRI
^VECS7G("B" VECS7CRI*,VECS7FLD*!
```

\$T References

```
OPTIONS+II   VECS7MNI
OPTIONS+OPT   VECS7MNI
```

Routine Invokes:

```
VECS7CRI     VECS7FLD
VECS7EDI     VECS7FLD
VECS7PRI     %ZIS
```

Routine is Invoked by:

```
^%ZIS        VECS7PRI
^VECS7FLD    VECS7CRI,VECS7EDI
```

```
*****      END      *****
```

```
--- D O N E ---
```

```
>
```

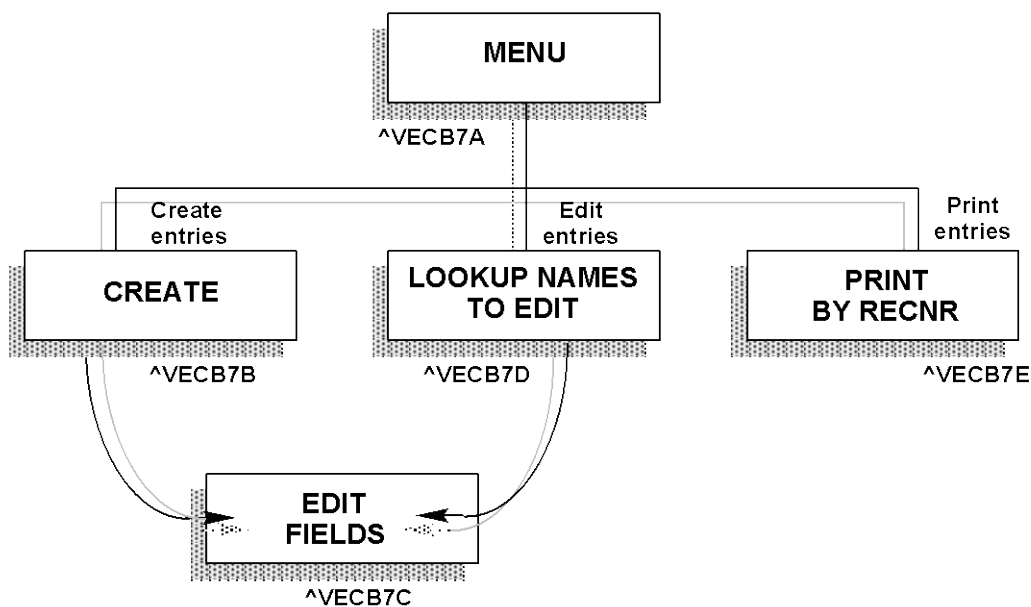

VA M PROGRAMMING
Intermediate

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) on a test system under the namespace assigned to you by your local IRM chief or IT manager. For this lesson, use the following suffix pattern:

VECB7A	for the menu routine
VECB7B	for the create routine
VECB7C	for the edit-fields routine
VECB7D	for the lookup names to edit routine
VECB7E	for the print by record number routine



Important

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

VECB7G

(G stands for Global)

VA M PROGRAMMING

Intermediate

Mandatory Specifications

You are to redesign and reimplement the personnel system you first developed in Lesson 5 and enhanced in Lesson 6. You should follow the Sample System in this lesson. You must add index files ("B" cross references).

The fields and their validation are the same as in the Lesson 6 assignment:

<i>Field</i>	<i>Format</i>
Name	Last name,First name\ (all CAPS; no space before or after comma)
Address	No more than 25 characters
City	No more than 20 characters
State	Two-letter code
Zip	Five numeric digits
Social security number	nnn-nn-nnnn (use hyphens only)
Department	a code: 01, 02, 03, or 04 are the only codes that are acceptable (make sure the user types in the leading zero)
Job Title	No more than 20 characters
Home phone number	nnn-nnn-nnnn (use area code and hyphens)
Employment date	Display in mm-dd-yyyy (<u>use hyphens</u> ; the sample shows /s but you are to use -'s) mm is month number dd is day number yyyy is year store as yyymmdd (no hyphens)

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

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

Allow the user to add duplicate names by enclosing them in quotes.

If they want to delete a record, they should enter a @ at the name prompt in the edit function. If they want to delete a field, they should enter a @ at the field prompt.

VA M PROGRAMMING

Intermediate

There should be one edit routine now that handles the validation for both the create and edit function of the menu. The edit function should list duplicate names (with SSN) in groups of 7 and give the user the option of typing in a selection number.

If you have Kernel in the account you use for the course, use %ZIS for OPENing and CLOSEing the device.