

**DIC("?N",file#)
=n**

(Optional) The number "n" should be an integer set to the number of entries to be displayed on the screen at one time when using "?" help in a lookup. Usually, file# will be the number of the file on which you're doing the lookup. However, if doing a lookup using an index on a pointer field, and if DIC(0) contains "L", then the user also is allowed to see a list of entries from the pointed-to file, so in that case file# could be the number of that pointed-to file. For example, when doing a lookup in test file 662001, if the developer wants only five entries at a time to be displayed in question-mark help, set DIC("?N",662001)=5

**DIC("?PARAM",
file#,"INDEX")=
Index name**

(Optional) Used to control entries displayed during online "?" help only. If provided, this index will be used to display the entries from the file specified by file#. Otherwise, VA FileMan uses the first lookup index specified for the ^DIC call. This value is used as the INDEX parameter to the Lister call to display the entries. See documentation for LIST^DIC for more information.

**DIC("?PARAM",
file#,"FROM",n)
=value**

(Optional) Used to control entries displayed during online "?" help only. This array can be set to define a starting value for an entry in the lookup index used to list entries from the file. Integer value "n" is associated with the "nth" data value subscript in the index (e.g., regular old-style indexes always have just one indexed data value so "n" would be 1). If a starting value is defined for subscript "n," then starting values must also be defined for all of the subscripts preceding "n."

This information is used to set the FROM parameter for a call to LIST^DIC in order to display the entries in the file specified by file#. Therefore, the entries must meet the same rules as the FROM parameter described in that call. See documentation for LIST^DIC for detailed information.

If DIC(0) contains an "L" and the first indexed field is a pointer, then after displaying the current entries on the file, VA FileMan allows the user to see entries on the pointed-to file. In that case, the developer may request starting values for any pointed-to file in the pointer chain. If the user enters "^value" when asked whether

they wish to see the entries in the file, the value entered by the user will override the starting list value passed by the developer in this array.

**DIC("?PARAM",
file#,"PART",n)
=value**

(Optional) Used to control entries displayed during online "?" help only. This array can be set to define partial match value(s) for each of the "n" subscripts on the lookup index used during online help. The information is used to set the PART parameter for a Lister call to display the entries. See documentation for LIST^DIC for more information. As with DIC("?PARAM",file#,"FROM",n), if DIC(0) contains "L", the developer can define partial match values for any pointed-to file in the pointer chain.

DLAYGO

(Optional) If this variable is set equal to the file number, then the users will be able to add a new entry to the file whether or not they have LAYGO access to the file. This variable, however, does not override any LAYGO node that may exist on the .01 field, i.e. ^DD(file#,.01,"LAYGO",#,0). M code in the LAYGO node will still be executed, and must set the truth value to TRUE for an entry to be added.

NOTE: In addition, DIC(0) must contain L to allow addition of entries to the file.

Output Variables

Y DIC always returns the variable Y. The variable Y is returned with one of these three formats:

Y=-1 The lookup was unsuccessful.

Y=N^S N is the internal number of the entry in the file and S is the value of the .01 field for that entry.

Y=N^S^1 N and S are defined as above and the 1 indicates that this entry has just been added to the file.

- Y(0)** This variable is only set if DIC(0) contains a Z. When the variable is set, it is equal to the entire zero node of the entry that was selected.
- Y(0,0)** This variable also is only set if DIC(0) contains a Z. When the variable is set, it is equal to the external form of the .01 field of the entry.

The following are examples of returned Y variables based on a call to the EMPLOYEE file stored in ^EMP(:

```
S DIC="^EMP( ",DIC(0)="QEZ",X="SMITH"
D ^DIC
```

Returned are:

```
Y      = "7^SMITH,SAM"
Y(0)   = "SMITH,SAM^M^2231109^2"
Y(0,0) = "SMITH,SAM"
```

If the lookup had been done on a file whose .01 field points to the EMPLOYEE file, the returned variables might look like this:

```
Y      = "32^7" [ Entry #32 in this file and #7
                in EMPLOYEE file.]
Y(0)   = "7^RX 2354^ON HOLD"
Y(0,0) = "SMITH,SAM" [.01 field of entry 7 in
                    EMPLOYEE file]
```

- X** Contains the value of the field where the match occurred.

If the lookup index is compound (i.e., has more than one data subscript), and if DIC(0) contains "A" so that the user is prompted for lookup values, then X will be output as an array X(n) where "n" represents the position in the subscript and will contain the values from the index on which the entry was found. Thus, X(2) would contain the value of the second subscript in the index. If possible, the entries will be output in their external format (i.e., if the subscript is not computed and doesn't have a transform). If the entry is not found on an index (example, when lookup is done with X=" " (the space-bar return feature)), then X and X(1) will contain the user input, but the rest of the X array will

be undefined.

DTOUT This is only defined if DIC has timed-out waiting for input from the user.

DUOUT This is only defined if the user entered an up-arrow.

DIC(0) Input Variables in Detail

The effects of the various characters which can be contained in DIC(0) are described below:

- A** DIC asks for input from the terminal and asks again if the input is erroneous. A response of null or a string containing ^ is accepted. Input is returned in X when DIC quits. If DIC(0) does not contain the character A, the input to DIC is assumed to be in the local variable X.
- B** Without the B flag, if there are cross-referenced pointer or variable pointer fields in the list of indexes to use for lookup and if DIC(0) contains "M" and there is no screening logic on the pointer that controls the lookup on the pointed-to file, then:
1. For each cross-referenced pointer field, FileMan checks ALL lookup indexes in each pointed-to file for a match to X (time-consuming);
 2. If X matches any value in any lookup index (not just the "B" index) on the pointed-to file and the IEN of the matched entry is in the home file's pointer field cross-reference, FileMan considers this a match. This may perhaps not be the lookup behavior you wanted, see Examples section.

The B flag prevents this behavior by looking for a match to X only in the B index (.01 field) of files pointed to by cross-referenced pointer or variable pointer fields. This makes lookups quicker and avoids the risk of FileMan matching an entry in the pointed-to file based on some unexpected indexed field in that file.

- C** Normally, when DIC does a lookup and finds an entry that matches the input, that entry is presented to the user only once even if the entry appears in more than one cross-reference. This is called cross-reference suppression and can be

overridden by including a C in DIC(0). If, for example, a person with the name ZACHARY,DAVID is an entry in a file, then his name will appear in the B cross-reference of the file. If he has a nickname of ZACH, which is in the C cross-reference of the file, then when a user enters ZACH as a lookup value, the name, ZACHARY,DAVID, will appear only once in the choices. But if there is a C in DIC(0), then ZACHARY,DAVID will appear twice in the choices; once as a hit in the B cross-reference and again as a hit in the C cross-reference.

F Prevents saving the entry number of the matched entry in the ^DISV global. Ordinarily, the entry number is saved at ^DISV(DUZ,DIC). This allows the user to do a subsequent lookup of the same entry simply by pressing the space bar and Enter/Return key. To avoid the time cost of setting this global, include an F in DIC(0).

I If DIC(0) contains I, any special user-written lookup program for a file will be ignored and DIC will proceed with its normal lookup process.

You can write a special lookup program to be used to find entries in a particular file. This special program can be defined by using the Edit File option of the Utility Functions submenu (see the Special Lookup Programs section in the Advanced File Definition chapter.) When a lookup program is defined, VA FileMan will bypass the normal lookup process of DIC and branch to the user written program. This user written lookup program must respond to the variables documented in this section and provide the functionality of DIC as they pertain to the file.

K This flag causes ^DIC to use the Uniqueness index for the Primary Key as the starting index for the lookup, rather than starting with the B index. (If developers want to specify some other index as the starting index, then they can specify the index by using the "D" input variable, and either the IX^DIC or the MIX^DIC1 call instead of ^DIC.)

L If DIC(0) contains L and the user's input is in valid format for the file's .01 field, then DIC will allow the user to add a new entry to the file at this point (Learn-As-You-GO), as long as at least one of these four security-check conditions is true:

The local variable DUZ(0) is equal to the @-sign.

If Kernel's File Access Security System (formerly known as Kernel Part 3) is being used for security, the file is listed in the user's record of accessible files with LAYGO access allowed.

If file access management is not being used, a character in DUZ(0) matches a character in the file's LAYGO access code or the file has no LAYGO access code.

The variable DLAYGO is defined equal to the file number.

NOTE: Even if DIC(0) contains L and one of these security checks is passed, LAYGO will not be allowed if a test in the data dictionary's LAYGO node fails.

M If DIC(0) contains M, DIC will do a multiple lookup on all of the file's cross-references from B on to the end of the alphabet. For example, if a given file is cross-referenced both by Name and by Social Security Number, and the user inputs 123-45-6789, DIC, failing to find this input as a Name, will automatically go on to look it up as a Social Security Number.

NOTE: For finer control in specifying the indexes used for lookup, see the alternate lookup entry points IX^DIC and MIX^DIC1.

N If DIC(0) contains N, the input is allowed to be checked as an internal entry number even if the file in question is not normally referenced by number. However, input is only checked as an IEN if no other matches are found during regular lookup.

If DIC(0) does not contain an N, the user is still allowed to select by entry number by preceding the number with the accent grave character (`). When a ` is used, the lookup is limited to internal entry numbers only.

Placing N in DIC(0) does not force IEN interpretation; it only permits it. In order to force IEN interpretation, you must use the accent grave (`) character.

NOTE: With this flag, when DIC(0) contains an L, users may be allowed to force the internal entry number when adding new entries to the file. If the user enters a number N that is not found on any of the cross-references, and if the .01 field is not numeric and the file is not DINUMed, and if FileMan can talk to the users (DIC(0) ["E"]), then the user will be asked

whether they want to add the new entry, and will be prompted for the value of the .01 field. The entry will be added at the record number N that was originally entered by the user. Note that if there is a .001 field on the file, the number N must also pass the INPUT transform for the .001 field.

- O** If DIC(0) contains the letter O, then for each index searched, FileMan looks first for exact matches to the lookup value before looking for partial matches. If an exact match is found, then FileMan returns only that match and none of the partial matches on the index. Thus if an index contained the entries 'SMITH,SAM' and 'SMITH,SAMUEL' and if the user typed a lookup value of 'SMITH,SAM', then only the 'SMITH,SAM' entry would be selected, and the user would never see the entry 'SMITH,SAMUEL'. Note that if partial matches but no exact matches are found in the first index(es) searched, but if exact matches are found in an index searched later, then the partial matches from the first index(es) are returned along with the exact match from the later index(es).
- Q** If DIC(0) contains Q and erroneous input is entered, two question marks (??) will be displayed and a "beep" will sound.
- S** If DIC(0) does not contain S, the value of the .01 field and Primary Key fields (if the file has a Primary Key) will be displayed for all matches found in any cross-reference. If DIC(0) does contain S, the .01 field and Primary Key fields will not be displayed unless they are one of the indexed fields on which the match was made.
- T** "T flag in DIC(0). Present every match to the lookup value, quitting only when user either selects one of the presented entries, enters ^^ to quit, or there are no more matching entries found.

Currently, if one or more matches are found in the first pass through the indexes, then FileMan quits the search, whether or not one of the entries is selected. Only if no matches are found in the first pass does FileMan continue on to try transforms to the lookup value. This includes transforms to find internal values of pointers, variable pointers, dates or sets.

Another feature of the "T" flag is that indexes are truly searched in the order requested. If, for example, an index on a pointer field comes before an index on a free-text field, matches

from the pointer field will be presented to the user before matches to the free-text field. When used in combination with the "O" flag, all indexes will be searched for an exact match. Then, only if no matches are found, will FileMan make a second pass through the indexes looking for partial matches.

- U** Normally the lookup value is expected to be in external format (for dates, pointers and such). FileMan first searches the requested index for a match to the user input as it was typed in. Then, if no match is found, FileMan automatically tries certain transforms on the lookup value.

For instance, if one of the lookup indexes is on a date field, FileMan tries to transform the lookup value to an internal date, then checks the index again. The U flag causes FileMan to look for an exact match on the index and to skip any transforms. Thus the lookup value must be in internal format. This is especially useful for lookups on indexed pointer fields, where the internal entry number (i.e., internal pointer value) from the pointed-to file is already known.

Ordinarily this flag would not be used along with the "A", "B", "M", "N" or "T" flags. In many cases it makes sense to combine this with the "X" flag.

- V** If DIC(0) contains V and only one match is made to the user's lookup value, then they will be asked "OK?" and they will have to verify that the looked-up entry is the one they wanted. This is an on the fly way of getting behavior similar to the permanent flag that can be set on a file by answering "YES" to the question "ASK 'OK' WHEN LOOKING UP AN ENTRY?" (See the EDIT FILE option within the FileMan UTILITY option, described in the Advanced User Manual).

- X** If DIC(0) contains X, for an exact match, the input value must be found exactly as it was entered. Otherwise, the routine will look for any entries that begin with the input X. Unless 'X-act match' is specified, lowercase input that fails in the lookup will automatically be converted to uppercase, for a second lookup attempt. The difference between X and O (described above) is that X requires an exact match. If there is not one, either DIC exits or tries to add a new entry. With O, if there is not an exact match, DIC looks for a partial match beginning with the input.

Z If DIC(0) contains Z and if the lookup is successful, then the variable Y(0) will also be returned. It will be set equal to the entire zero node of the entry that has been found. Another array element, Y(0,0), is also returned and will be set equal to the printable expression of the .01 field of the entry selected. This has no use for Free Text and Numeric data types unless there is an OUTPUT transform. However, for Date/Time, Set of Codes and Pointer data types, Y(0,0) will contain the external format.

Adding New Subentries to a Multiple

You can use ^DIC or FILE^DICN to add new subentries to a multiple. In order to add a subentry, the following variables need to be defined:

- DIC** Set to the full global root of the subentry. For example, if the multiple is one level below the top file level:
file's_root,entry#,multiple_field's_node,
- DIC(0)** Must contain "L" to allow LAYGO.
- DIC("P")** Set to the 2nd piece of 0-node of the multiple field's DD entry. **NOTE:** As of Version 22 of FileMan, the developer is no longer required to set DIC("P"). The only exception to this is for a few files that are not structured like a normal FileMan file, where the first subscript of the data is variable in order to allow several different 'globals' to use the same DD. An example of this is the FileMan Audit files where the first subscript is the file number of the file being audited.
- DA(1)...** Set up this array such that DA(1) is the IEN at the next higher file level above the multiple that the lookup is being performed in, DA(2) is the IEN at the next higher file level (if any), ... DA(n) is the IEN at the file's top level.
- DA(n)**
- NOTE:** The value of the unsubscripted DA node should not be defined when doing lookups in a subfile—that's the value you're trying to obtain!

A.) Below is an example of code that:

1. Uses ^DIC to interactively select a top-level record.
2. Uses ^DIC to select or create a **subentry** in a multiple in that record.
3. Uses ^DIE to edit fields in the selected or created subentry.

The file's root in this example is '^DIZ(16150,', the multiple's field number is 9, and the multiple is found on node 4. The code for this example follows:

```
; a call is made to DIC so the user can select an entry in the file
;
S DIC="^DIZ(16150,",DIC(0)="QEAL" D ^DIC
I Y=-1 K DIC Q ;quit if look-up fails
;
; a second DIC call is set up to select the subentry
;
S DA(1)=+Y ;+Y contains the internal entry number of entry chosen
S DIC=DIC_DA(1)_"",4," ;the root of the subfile for that entry
S DIC(0)="QEAL" ;LAYGO to the subfile is allowed
S DIC("P")=$P(^DD(16150,9,0),"^",2) ;returns the subfile# and specifiers
D ^DIC I Y=-1 K DIC,DA Q ;user selects or adds subentry
;
; a DIE call is made to edit fields in subfile
;
S DIE=DIC K DIC ;DIE now holds the subfile's root
S DA=+Y ;+Y contains the internal entry number of subentry chosen
S DR="1;2" D ^DIE ;edit fields number 1 and 2
K DIE,DR,DA,Y Q
```

B) File #662002 has a .01 field that points to the NEW PERSON file (#200). In this example, we'll use input arrays in DIC("?PARAM",662002,"FROM",1) to start the list of entries in the "B" index of File #662002 with the letter "M". Since DIC(0) contains "L" (user can add entries to the pointed-to File #200), VA FileMan will also display entries from File #200, so we use DIC("?PARAM",200,"PART",1) to display only entries that start with the letter "S".

```
>S DIC=^DIZ(662002,DIC(0)="AEQZL"
>S DIC("?PARAM",200,"PART",1)="S"
>S DIC("?PARAM",662002,"FROM",1)="M"
```

```
>D ^DIC
```

```
Select ZZTAMI POINT TO NEW PERSON PERSON NAME: ??
```

```
Choose from:
MANNERS,JULIA      NOV 11, 1961      ANOTHER GREAT PROGRAMMER      JM
PROGRAMMER
MARSHALL,DELBERT  MAY 05, 1965      WIZARD      TOAD      PROGRAMMER
OGDEN,MARSHALL  JUL 07, 1977      GREAT PROGRAMMER      MO PROGRAMMER
RETROMAN,USER K M JR      JAN 01, 1969      COOLDUDE      UR
TIMOTHY,S J      APR 03, 1948      KOOL KAT      SJT PROGRAMMER
```

```

WERLY,BIRD      JUN 12, 1955      GROOVY GUY      BW PROGRAMMER
WINNER,BIG      AUG 28, 1949      COMPUTER SPECIALIST  BW
PROGRAMMER
WINNER,SMALL    AUG 28, 1948      SECOND PLACE    SW PROGRAMMER
ZERO,BOB        MAR 02, 1948      VERY GOOD PROGRAMMER  BZ
IRMFO          PROGRAMMER
    
```

You may enter a new ZYTAMI POINT TO NEW PERSON, if you wish

```

Choose from:
SHARED,MAIL
SMITH,JOHN HOWARD STEVEN II
STRALL,SEG      SAS
SUPERMAN,JOE X Y JR
    
```

C.) In this example we are using the same files as in example "B", we will display entries from the pointing File #662002, using the "AC" index, which sorts the entries by TITLE, then by NAME. In this case, we will limit the number of entries displayed at one time from both File #662002 and File #200 to 5.

```

>S DIC="^DIZ(662002," ,DIC(0)="AEQZL"
>S DIC("?PARAM",662002,"INDEX")="AC"
>S DIC("?N",662002)=5
>S DIC("?N",200)=5
    
```

```
>D ^DIC
```

Select ZYTAMI POINT TO NEW PERSON PERSON NAME: ??

```

Choose from:
A STATE CALIFORNIA,MR      MAR 01, 1875      A STATE      MC      ABCD
ANOTHER GREAT PROGRAMMER  MANNING,DARYL    NOV 11, 1961    ANOTHER
GREAT
PROGRAMMER      DM      PROGRAMMER
BROKER WHIZ     CROSS,BOB      FEB 05, 1950      BROKER WHIZ     BC
COMPUTER SPECIALIST  WILLY,TED      AUG 28, 1949      COMPUTER SPECIALIST TW
COOLDUDE        RETROMAN,USER K M JR      JAN 01, 1969      COOLDUDE        UR
    
```

^

You may enter a new ZYTAMI POINT TO NEW PERSON, if you wish

Answer with NEW PERSON NAME

Do you want the entire NEW PERSON List? Y <RET> (Yes)

Choose from:

```

ATESTMAN,BOB K III      BKA
CALIFORNIA,MR      MC      ABCD
CLARK,KENT      KC
CROSS,BOB      BC      PROGRAMMER
DELANCY,NAN      ND      PROGRAMMER
    
```

IX^DIC: Lookup/Add

This entry point is similar to ^DIC and MIX^DIC1, except for the way it uses cross-references to perform lookup. The three entry points perform lookups as follows:

- ^DIC** Starts with the B cross-reference, or uses only the B cross-reference [unless K is passed in DIC(0)].
- IX^DIC** Starts with the cross-reference you specify or uses only the cross-reference you specify.
- MIX^DIC1** Uses the set of cross-references you specify.

Input Variables (Required)

NOTE: All of the input variables described in ^DIC can be used in the IX^DIC call. The following variables are required.

- DIC** The global root of the file, e.g., ^DIZ(16000.1,.
- DIC(0)** The lookup parameters as previously described for ^DIC.
- D** The cross-reference in which to start looking. If DIC(0) contains M, then DIC will continue the search on all other lookup cross-references, in alphabetical order. If it does not, then the lookup is only on the single cross-reference. This variable is killed by VA FileMan; it is undefined when the IX^DIC call is complete.

If DIC(0) contains "L", (i.e., user will be allowed to add a new entry to the file), then either a) D should be set to "B" or b) D should be set to an index that alphabetically comes before "B" and DIC(0) should contain "M" or c) D should contain the name of a compound index.
- X** If DIC(0) does not contain an A, then the variable X must be defined equal to the value you want to look up.

If the lookup index is compound (i.e., has more than one data subscript), then X can be an array X(n) where "n" represents the position in the subscript. For example, if X(2) is passed in, it will be used as the lookup value to match to the entries in the

second subscript of the index. If only the lookup value X is passed, it will be assumed to be the lookup value for the first subscript in the index, X(1).

Input Variables (Optional)

All of the ^DIC input variables can be used in the IX^DIC call. These variables below are optional.

DIC("A"),
DIC("B"),
DIC("DR"),
DIC("P"),
DIC("PTRIX",f,p,t)=d
DIC("S"),
DIC("V"),
DIC("W")
DIC("?N",file#)=n

This set of input variables affects the behavior of lookup as described for ^DIC.

Output Variables

- Y** DIC always returns the variable Y. The variable Y is returned in one of these three formats:
- Y=-1** The lookup was unsuccessful.
 - Y=N^S** N is the Internal Entry Number of the entry in the file and S is the value of the .01 field for that entry.
 - Y=N^S^1** N and S are defined as above and the 1 indicates that this entry has just been added to the file.
- Y(0)** This variable is only set if DIC(0) contains a Z. When the variable is set, it is equal to the entire zero node of the entry that was selected.
- Y(0,0)** This variable also is only set if DIC(0) contains a Z. When the variable is set, it is equal to the external form of the .01 field of the entry.

The following are examples of returned Y variables based on a call to the EMPLOYEE file stored in ^EMP(:

```
S DIC="^EMP( ",DIC(0)="QEZ",X="SMITH"
D ^DIC
```

Returned are:

```
Y      = "7^SMITH,SAM"
Y(0)   = "SMITH,SAM^M^2231109^2"
Y(0,0) = "SMITH,SAM"
```

If the lookup had been done on a file whose .01 field points to the EMPLOYEE file, the returned variables might look like this:

```
Y      = "32^7" [ Entry #32 in this file and #7 in
                EMPLOYEE file.]
Y(0)   = "7^RX 2354^ON HOLD"
Y(0,0) = "SMITH,SAM" [.01 field of entry 7 in
                EMPLOYEE file]
```

X Contains the value of the field where the match occurred.

If the lookup index is compound (i.e., has more than one data subscript), and if DIC(0) contains an A so that the user is prompted for lookup values, then X will be output as an array X(n) where "n" represents the position in the subscript and will contain the values from the index on which the entry was found. Thus, X(2) would contain the value of the second subscript in the index. If possible, the entries will be output in their external format (i.e., if the subscript is not computed and doesn't have a transform). If the entry is not found on an index (for example, when lookup is done with X=" " [the space-bar return feature]), then X and X(1) will contain the user input, but the rest of the X array will be undefined.

DTOUT This is only defined if DIC has timed-out waiting for input from the user.

DUOUT This is only defined if the user entered an up-arrow.

DO^DIC1: File Info Setup

This entry point retrieves a file's file header node, code to execute its identifiers and its screen (if any), and puts them into local variables for use during lookup into a file.

If \$D(DO) is greater than zero, DO^DIC1 will QUIT immediately. If DIC("W") is defined before calling DO^DIC1, it will not be changed.

Input Variables

- DIC** The global root of the file, e.g., ^DIZ(16000.1,.
- DIC(0)** The lookup parameters as previously described for ^DIC.

Output Variables

- DO** File name^file number and specifiers. This is the file header node.
- NOTE:** Use the letter O, not the number zero, in this variable name.
- DO(2)** File number and specifiers. This is the second ^piece of DO. +DO(2) will always equal the file number.
- DIC("W")** This is an executable variable which contains the write logic for identifiers. When an entry is displayed, the execution of this variable shows other information to help identify the entry. This variable is created by \$ORDERing through the data dictionary ID level, for example:
- ```
^DD(+DO(2),0,"ID",value)
```
- NOTE:** The specifier, I, must be in DO(2) for VA FileMan to even look at the ID-nodes.
- DO("SCR")**    An executable variable which contains a file's screen (if any). The screen is an IF-statement that can screen out certain entries in the file. This differs from DIC("S") in that it is used on every lookup regardless of input or output; that is, the screen is applied to inquiries and printouts as well as to lookups. The value for this variable comes from ^DD(+DO(2),0,"SCR") and the specifier "s" must be in DO(2).

## MIX^DIC1: Lookup/Add

This entry point is similar to ^DIC and IX^DIC, except for the way it uses cross-references to do lookup. The three entry points perform lookups as follows:

- ^DIC** Starts with the B cross-reference or uses only the B cross-reference (unless K is passed in DIC(0)).
- IX^DIC** Starts with the cross-reference you specify or uses only the cross-reference you specify.
- MIX^DIC1** Uses the set of cross-references you specify.

### Input Variables (Required)

**NOTE:** All of the input variables described in ^DIC can be used in the MIX^DIC1 call. The following variables are required.

- DIC** The global root of the file, e.g., ^DIZ(16000.1,.
- DIC(0)** The lookup parameters as previously described for ^DIC.
- D** The list of cross-references, separated by up-arrows, to be searched, e.g., D="SSN^WARD^B". This variable is killed by VA FileMan; it is undefined when the MIX^DIC1 call is complete. If DIC(0) contains "L", meaning that the user can add a new entry to the file, then either a) the "B" index should be included in the list contained in D, or b) D should be set to the name of a compound index.  
  
Make sure DIC(0) contains M; otherwise, only the first cross-reference in D will be used for the lookup.
- X** If DIC(0) does not contain an A, then the variable X must be defined equal to the value you want to look up.  
  
If the lookup index is compound (i.e., has more than one data subscript), then X can be an array X(n) where "n" represents the position in the subscript. For example, if X(2) is passed in, it will be used as the lookup value to match to the entries in the second subscript of the index. If only the lookup value X is

passed, it will be assumed to be the lookup value for the first subscript in the index, X(1).

### Input Variables (Optional)

All of the ^DIC input variables can be used in the MIX^DIC1 call. The variables below are optional.

**DIC("A"),  
DIC("B"),  
DIC("DR"),  
DIC("P"),  
DIC("PTRIX",f,p,t)=d  
DIC("S"),  
DIC("V"),  
DIC("W")  
DIC("?N",file#)=n**

This set of input variables affects the behavior of lookup as described for ^DIC.

### Output Variables

- Y** DIC always returns the variable Y. The variable Y is returned in one of the three following formats:
- Y=-1** The lookup was unsuccessful.
- Y=N^S** N is the Internal Entry Number of the entry in the file and S is the value of the .01 field for that entry.
- Y=N^S^1** N and S are defined as above and the 1 indicates that this entry has just been added to the file.
- Y(0)** This variable is only set if DIC(0) contains a Z. When the variable is set, it is equal to the entire zero node of the entry that was selected.
- Y(0,0)** This variable also is only set if DIC(0) contains a Z. When the variable is set, it is equal to the external form of the .01 field of the entry.

The following are examples of returned Y variables based on a call to the EMPLOYEE file stored in ^EMP(:

```
S DIC=" ^EMP(" ,DIC(0)="QEZ" ,X="SMITH"
D ^DIC
```

**Returned are:**

```
Y = "7^SMITH,SAM"
Y(0) = "SMITH,SAM^M^2231109^2"
Y(0,0) = "SMITH,SAM"
```

If the lookup had been done on a file whose .01 field points to the EMPLOYEE file, the returned variables might look like this:

```
Y = "32^7" [Entry #32 in this file and #7 in
 EMPLOYEE file.]
Y(0) = "7^RX 2354^ON HOLD"
Y(0,0) = "SMITH,SAM" [.01 field of entry 7 in
 EMPLOYEE file]
```

**X** Contains the value of the field where the match occurred.

If the lookup index is compound (i.e., has more than one data subscript), and if DIC(0) contains an A so that the user is prompted for lookup values, then X will be output as an array X(n) where "n" represents the position in the subscript and will contain the values from the index on which the entry was found. Thus, X(2) would contain the value of the second subscript in the index. If possible, the entries will be output in their external format (i.e., if the subscript is not computed and doesn't have a transform). If the entry is not found on an index (for example, when lookup is done with X=" " [the space-bar return feature]), then X and X(1) will contain the user input, but the rest of the X array will be undefined.

**DTOUT** This is only defined if DIC has timed-out waiting for input from the user.

**DUOUT** This is only defined if the user entered an up-arrow.

## **WAIT^DICD: Wait Messages**

Use this entry point to display VA FileMan's informational messages telling users that the program is working and they must wait a while. The selection of the phrase is random. There are no input or output variables.

Some sample messages are:

```
...EXCUSE ME, I'M WORKING AS FAST AS I CAN...
```

```
...SORRY, LET ME THINK ABOUT THAT A MOMENT...
```

## **FILE^DICN: Add**

This entry point adds a new entry to a file. The INPUT transform is **not** used to validate the value being added as the .01 field of the new entry. This call does not override the checks in the LAYGO nodes of the data dictionary; they must still prove true for an entry to be added.

FILE^DICN can also be used to add subentries in multiples. See the Adding New Subentries to a Multiple discussion in the description of ^DIC.

### **Variables to Kill**

**DO** If DO is set, then FileMan assumes that all of the variables described as output in the call to DO^DIC1 have been set as well and that they describe the file to which you wish to add a new record. If you're not sure, then DO should be killed and the call will set it up for you based on the global root in DIC.

**NOTE:** This variable is D with the letter O, not zero.

### **Input Variables**

**DIC** The global root of the file.

**DIC(0)** (Required) A string of alphabetic characters which alter how DIC responds. At a minimum this string must be set to null. The characters you can include are:

**E** **Echo back information.** This tells DIC that you are in an interactive mode and are expecting to be able to receive input from the user. If there are identifiers when adding a new entry, for example, the user can edit them as the entry is added if the E flag is used.

**F** **Prevents saving the entry number of the matched entry in the ^DISV global.** Ordinarily, the entry number is saved at ^DISV(DUZ,DIC). This allows the user to do a subsequent lookup of the same entry simply by pressing the space bar and the Enter/Return key. To avoid the time cost of setting this global, include an F in DIC(0).

**Z** Zero node returned in Y(0) and external form in Y(0,0).

**DIC("P")**

**NOTE:** Beginning with Version 22.0 of VA FileMan, the developer is no longer required to set DIC("P").

The only exception to this is for a few files that are not structured like a normal VA FileMan file, where the first subscript of the data is variable in order to allow several different "globals" to use the same DD. An example of this is the VA FileMan Audit files where the first subscript is the file number of the file being audited.

Used when adding subentries in multiples. See description in ^DIC section.

**DA**

Array of entry numbers. See the Adding New Subentries to a Multiple discussion in the description of ^DIC.

**X**

The internal value of the .01 field, as it is to be added to the file. The programmer is responsible for ensuring that all criteria described in the INPUT transform have been met. That means that the value X must be in internal format as it would be after executing the input transform. For example, a date must be in FileMan internal format '2690302', not 'March 02, 1969'. Also local variables set by the input transform code must be set. For example, if the input transform sets DINUM, then DINUM must be set to the record number at which the entry must be added.

**DINUM**

(Optional) Identifies the subscript at which the data is to be stored, that is, the internal entry number of the new record, shown as follows. (This means that DINUM must be a canonic number and that no data exists in the global at that subscript location.)

```
$D(@ (DIC_DINUM_ ")) = 0
```

If a record already exists at the DINUM internal entry number, no new entry is made. The variable Y is returned equal -1.

**DIC("DR")**

(Optional) Used to input other data elements at the time of adding the entry. If the user does not enter these elements, the entry will not be added. The format of DIC("DR") is the same as the variable DR described under the discussion of ^DIE.

If there are any required Identifiers for the file or if there are keys defined for the file (in the KEY file), and if DIC(0) does not

contain an E, then the identifier and key fields **MUST** be present in DIC("DR") in order for the record to be added. If DIC(0) contains E, the user will be prompted to enter the identifier and key fields whether or not they are in DIC("DR").

## Output Variables

- Y** DIC always returns the variable Y, which can be in one of the two following values:
- Y=-1** Indicates the lookup was unsuccessful; no new entry was added.
  - Y=N^S^1** N is the internal number of the entry in the file, S is the value of the .01 field for that entry, and the 1 indicates that this entry has just been added to the file.
- Y(0)** This variable is only set if DIC(0) contains a Z. When it is set, it is equal to the entire zero node of the entry that was selected.
- Y(0,0)** This variable is also only set if DIC(0) contains a Z. When it is set, it is equal to the external form of the .01 field of the entry.
- DTOUT** This is only defined if DIC has timed-out waiting for input from the user.
- DUOUT** This is only defined if the user entered an up-arrow.
- X** The variable X will be returned unchanged from the input value.

**YN^DICN: Yes/No**

This entry point is a reader for a YES/NO response. You must display the prompt yourself before calling YN^DICN. YN^DICN displays the question mark and the default response, reads and processes the response, and returns %.

Recommendation: Instead of using this entry point, it is suggested that you use the generalized reader ^DIR. ^DIR gives you greater flexibility in displaying prompts and help messages and also presents more information about the user's response.

**Input Variables**

|                     |                                             |
|---------------------|---------------------------------------------|
| %                   | Determines the default response as follows: |
| <b>% = 0 (zero)</b> | No default                                  |
| <b>% = 1</b>        | YES                                         |
| <b>% = 2</b>        | NO                                          |

**Output Variables**

|                     |                                                                                                              |
|---------------------|--------------------------------------------------------------------------------------------------------------|
| %                   | The processed user's response. It can be one of the following:                                               |
| <b>% = -1</b>       | The user entered an ^ (up-arrow).                                                                            |
| <b>% = 0 (zero)</b> | The user pressed the Enter/Return key when no default was presented OR the user entered a ? (question mark). |
| <b>% = 1</b>        | The user entered a YES response.                                                                             |
| <b>% = 2</b>        | The user entered a NO response.                                                                              |
| <b>%Y</b>           | The actual text that the user entered.                                                                       |

## DQ^DICQ: Entry Display for Lookups

This entry point displays the list of entries in a file a user can see. It can be used to process question mark responses directly. If DO is not defined, the first thing that DQ^DICQ does is call DO^DIC1 to get the characteristics of the selected file.

### Input Variables

|                                                         |                                                                                                                                  |
|---------------------------------------------------------|----------------------------------------------------------------------------------------------------------------------------------|
| <b>DIC</b>                                              | (Required) The global root of the file.                                                                                          |
| <b>DIC(0)</b>                                           | (Required) The lookup input parameter string as described for ^DIC.                                                              |
| <b>DIC("S")</b>                                         | (Optional) Use this variable in the same way as it is described as an input variable for ^DIC.                                   |
| <b>DIC("?N",file#)=<br/>n</b>                           | (Optional) Use this variable in the same way it is described as input to ^DIC.                                                   |
| <b>DIC("?PARAM",<br/>file#,"INDEX")=<br/>index name</b> | (Optional) Use this input array in the same way it is described as input to ^DIC.                                                |
| <b>DIC("?PARAM",<br/>file#,"FROM",n)<br/>=value</b>     | (Optional) Use this input array in the same way it is described as input to ^DIC.                                                |
| <b>DIC("?PARAM",<br/>file#,"PART",n)=<br/>value</b>     | (Optional) Use this input array in the same way it is described as input to ^DIC.                                                |
| <b>D</b>                                                | (Required) Set to "B".                                                                                                           |
| <b>DZ</b>                                               | (Required) Set to "??". This is set in order to prevent VA FileMan from issuing the "DO YOU WANT TO SEE ALL nn ENTRIES?" prompt. |

## **DT^DICRW: FM Variable Setup**

Sets up the required variables of VA FileMan. There are no input variables; simply call the routine at this entry point.

**NOTE:** This entry point kills the variables DIC and DIK.

### **Output Variables**

- |               |                                                                                                                                                                      |
|---------------|----------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| <b>DUZ</b>    | Set to zero if it is not already defined.                                                                                                                            |
| <b>DUZ(0)</b> | Set to null if not already defined. If DUZ(0)="@", this subroutine will enable terminal break if the operating system supports such functionality.                   |
| <b>IO(0)</b>  | Set to \$I if IO(0) is not defined. Therefore, this program should not be called if the user is on a device different from the home terminal and IO(0) is undefined. |
| <b>DT</b>     | Set to the current date, in VA FileMan format.                                                                                                                       |
| <b>U</b>      | Set to the up-arrow (^).                                                                                                                                             |

## **EN^DID: Data Dictionary Listing**

This entry point prints and/or displays a file's data dictionary listing by setting the input variables (the same as the output from the List File Attributes option described in the VA FileMan Advanced User Manual).

### **Input Variables**

- DIC** Set to the data dictionary number of the file to list.
- DIFORMAT** Set to the desired data dictionary listing format. Must be one of the following strings:
- STANDARD
  - BRIEF
  - MODIFIED STANDARD
  - TEMPLATES ONLY
  - GLOBAL MAP
  - CONDENSED
  - INDEXES AND CROSS-REFERENCES ONLY
  - KEYS ONLY

## **^DIE: Edit Data**

This routine handles input of selected data elements for a given file entry. You should use ^DIE only to edit *existing* records.

**NOTE:** When you call the DIE routine, it does not lock the record; you must do that yourself. See the discussion of locking below.

To allow the user to interactively choose the fields to edit, use the EN^DIB entry point instead.

### **Input Variables**

**DIE** (Required) The global root of the file in the form ^GLOBAL( or ^GLOBAL(#, or the number of the file.

If you are editing a subfile, set DIE to the full global root leading to the subfile entry, including all intervening subscripts and the terminating comma, up to but not including the IEN of the subfile entry to edit.

**DA** (Required) If you are editing an entry at the top level of a file, set DA to the internal entry number of the file entry to be edited.

If you are editing an entry in a subfile, set up DA as an array, where DA=entry number in the subfile to edit, DA(1) is the entry number at the next higher file level,...DA(n) is the entry number at the file's top level. See the section below on Editing a Subfile Directly for more information.

**NOTE:** The variable DA is killed if an entry is deleted within DIE. This can happen if the user answers with the @-sign when editing the entry's .01 field.

**DR** (Required) A string specifying which data fields are asked for the given entry. The fields specified by DR are asked *whether or not VA FileMan Write access security protection has been assigned* to the fields.

You can include the following in the DR-string:

**Field number:** The internal number of a field in a file.

**Field with Default Value:** A field number followed by // (two slashes), followed by a *default value*. You can make a field with no current data value default to a particular data value you specify. For example, if there is a file entry stored descendent from ^FILE(777), and field #27 for this file is DATE OF ADMISSION, and you want the user to see:

```
DATE OF ADMISSION: TODAY//
```

then the calling program should be:

```
S DR="27//TODAY",DIE="^FILE(",DA=777
D ^DIE
```

If the user just presses the Enter/Return key when seeing the prompt, DIE acts as though the user typed in the word TODAY.

**Stuff a Field Value (Validated):** A field number followed by /// (three slashes), followed by a value. The value should be the *external* form of the field's value, that is, the format that would be acceptable as a user's response. The value is *automatically inserted* into the database after passing through the INPUT transform. For example:

```
S DR="27///TODAY",DIE="^FILE(",DA=777
D ^DIE
```

The user sees no prompts, and the current date is automatically stuffed into field #27 of entry #777, *even if other data previously existed there*.

In the course of writing a routine, you may want to pass the value contained in a variable to DIE and automatically insert the value into a field. In that case, you would write:

```
S DR="27///^S X=VAR"
```

You can also use the three-slash stuff to automatically add or select an entry in a multiple. For example, if field #60 is a multiple field, and you write:

```
S DR="60///TODAY"
```

the entry in the subfile corresponding to TODAY would be selected, or added if it didn't already exist. Note, however, that if TODAY didn't already exist in the file, but couldn't be added (because LAYGO wasn't allowed, for example), or if more than one TODAY entry already existed in the file (that is, the lookup value was ambiguous), ^DIE will prompt the user to select an entry in the subfile. If you wish to add entries or edit existing entries non-interactively, consider using UPDATE^DIE and FILE^DIE instead.

**Stuff a Field Value (Unvalidated):** A field number followed by *////* (four slashes), followed by a value. The value is *automatically inserted without validation* into the database. For example:

```
S DR="27////2570120",DIE="^FILE(",DA=777
D ^DIE
```

The user sees no prompts, and the value 2570120 is put into field 27 without going through the INPUT transform. When using this form, the data after the four slashes must already be in its internally stored form. *This cannot be used for .01 fields due to the differences between DIE and DIC.*

**NOTE:** Key uniqueness is not enforced when a 4-slash stuff is used.

**Field Value Deletion:** A field number followed by three or four slashes (*///* or *////*) and an @-sign. This *automatically deletes the field value*. For example:

```
S DR="27///@"
```

The user does not see any prompts, and the value for field #27 is deleted.

**NOTE:** You cannot use this method to delete the value of a required field, an uneditable field, a key field, or a field the user does not have Delete access to.

**Field Number Range:** A range of field numbers, in the form M:N, where M is the first and N the last number of the *inclusive range*. All fields whose numbers lie within this range are asked.

**Placeholder for Branching:** A placeholder like @1. See the discussion of branching below.

**M Code:** A line of M code.

**Combination:** A sequence of any of the above types, separated by semicolons. If field numbers .01, 1, 2, 4, 10, 11, 12, 13, 14, 15, and 101 exist for the file stored in ^FILE, and you want to have fields 4, .01, 10 through 15, and 101 asked in that order for entry number 777, you simply write:

```
S DIE="^FILE(" ,DA=777,DR="4;.01;10;15;101"
D ^DIE
```

**NOTE:** The DR-string contains the semicolon delimiter to specify field numbers and the colon to specify a range of fields. This prevents these two characters from being used as defaults. They can, however, be placed in a variable which is then used as the default instead of a literal, for example:

```
S DR="27///^S X=VAR"
```

**INPUT template:** An INPUT template name, preceded by an open bracket ([]) and followed by a closed bracket (]). All the fields in that template are asked.

**DIE("NO^")**

(Optional) Controls the use of the ^ in an edit session. If this variable does not exist, unrestricted use of the ^ for jumping and exiting is allowed. The variable may be set to one of the following:

- "OUTOK"**                      Allows exiting and prevents all jumping.
- "BACK"**                        Allows jumping back to a previously edited field and does not allow exiting.
- "BACKOUTOK"**                Allows jumping back to a previously edited field and allows exiting.
- "Other value"**                Prevents all jumping and does not allow exiting.

**DIE("PTRIX",f  
,p,t)=d**

**DIE("PTRIX",f,p,t)=d** where,

**f** = the from (pointing) file number

**p** = the pointer field number

**t** = the pointed-to file number

**d** = an up-arrow (^) delimited list of index names

This optional input array allows you to control how lookups are done on both multiple and non-multiple pointer and variable pointer fields. Each node in this array is set to a list of index names, separated by up-arrows (^). When the user edits a pointer or variable pointer field, only those indexes in the list are used when searching the pointed-to file for matches to the lookup value.

For example, if your input template contains a field #5 on file #16100 that is a pointer to the NEW PERSON file (#200), and you want the lookup on the NEW PERSON file to be by name ("B" index), or by the first letter of the last name concatenated with the last 4 digits of the social security number ("BS5" index), you would set the following node before the ^DIE call:

```
DIE("PTRIX",16100,5,200)="B^BS"
```

Note that if you allow records to be added to the pointed-to file, you should include a "B" in the list of indexes, since when ^DIE adds an entry, it assumes the .01 field for the new entry is the lookup value. However, the "B" index would not need to be included if the first index in the "PTRIX" node is a compound index whose first subscript is the .01 field.

**DIDEL**

(Optional) Overrides the Delete access on a file or subfile. Set DIDEL equal to the number of the file before calling DIE to allow the user to delete an entire entry from that file, even if the user does not normally have the ability to delete. This variable does not override the "DEL"-nodes described in the Other Field Definition Nodes of the Global File Structure section.

## Output Variables

**DTOUT** Is set when a time-out has occurred.

**NOTE:** DA, DIE, DR, DIE("NO^"), and DIDEL are not killed by DIE; however, the variable DA is killed if the entry is deleted within DIE. This can happen if the user answers with an @-sign when editing the entry's .01 field.

## Details and Features of Data Editing

1. Locking
2. Edit Qualifiers
3. Branching
4. Specific Fields in Multiples
5. Continuation DR-Strings
6. Detecting Up-Arrow Exits
7. Editing a Subfile Directly
8. Screening Variable Pointers
9. Filing
10. New Style Compound Indexes and Keys

### 1. Locking

If you want to ensure that two users cannot edit an entry at the same time, lock the entry. It is recommended that you use incremental locks.

Here is a simple example of using incremental locks to lock an entry before editing and to remove the lock after:

```
S DIE="^FILE(",DA=777,DR="[EDIT]"
L +^FILE(777):0 I $T D ^DIE L -^FILE(777) Q
W !?5,"Another user is editing this entry." Q
```

**NOTE:** The DIE call itself does *NO* locking.

## 2. Edit Qualifiers

In the DR string, you can use edit qualifiers (described in the *VA FileMan Advanced User Manual*) in conjunction with the fields you specify. The possible qualifiers are T, DUP, REQ, and text literal strings in quotes.

In interactive mode, users can combine qualifiers with fields by using semicolon separators. But, in DR-strings, semicolons are already used to delimit individual fields, so you must use a different syntax for DR. Basically, leave out the semicolon and the unnecessary characters. Using field #3 as an example, the syntax for edit qualifiers in DR-strings is:

| <b>Interactive Syntax</b> | <b>Syntax for DR-string</b> | <b>Explanation</b>                                                                   |
|---------------------------|-----------------------------|--------------------------------------------------------------------------------------|
| <b>3;T</b>                | 3T                          | The T follows the field number immediately.                                          |
| <b>3;"xxx"</b>            | 3xxx                        | The quotes are removed from the literal and it follows the field number immediately. |
| <b>3;DUP</b>              | 3d                          | The D becomes lowercase and the UP is dropped.                                       |
| <b>3;REQ</b>              | 3R                          | The EQ is dropped and the uppercase R follows immediately.                           |

You can combine specifiers as long as you separate them with tildes (~). For example, if you want to require a response to field #3, and issue the title rather than the prompt, put 3R~T in the DR-string.

## 3. Branching

You can include **branching logic** within DR. To do this, insert an executable M statement in one of the semicolon-pieces of DR. The M code is executed when this piece of DR is encountered by the DIE routine.

If the M code sets the variable Y, DIE jumps to the field whose number (or label) matches Y. (The field must be specified elsewhere within the DR variable.) Y may look like a placeholder, e.g., @1. If Y is set to zero or the null string, DIE exits. If Y is killed, or never set, no branching occurs.

The M code can calculate Y based on X, which equals the internal value of the field previously asked for (as specified by the previous semicolon-piece of DR). Take the

example below and suppose that you do not want the user to be asked for field .01 if the answer to field 4 was YES, you would write the following:

```
S DIE="^FILE(",DA=777
S DR="4;I X="YES" S Y=10;.01;10;15;101"
D ^DIE
```

**NOTE:** The ability to up-arrow jump to specific fields does not take into account previous branching logic. You must ensure that such movements are safe.

#### 4. Specific Fields in Multiples

When you include the field number of a multiple in a DR-string, all the subfields of the multiple are asked. However, suppose you want to edit only selected subfields in the multiple. To do this, set DR in the usual manner and in addition set a subscripted value of DR equal to the subfields to edit. Subscript the additional DR node by file level and then by the multiple's subfile number.

For example, if field #15 is a multiple and the subfile number for the multiple is 16001.02 and you want the user to be prompted only for subfields .01 and 7, do the following:

```
S DR=".01;15;6;8"
S DR(2,16001.02)=".01;7"
```

where the first subscript, 2, means the second level of the file and the second subscript is the subfile number of the multiple field (#15).

#### 5. Continuation DR-Strings

If there are more than 245 characters in a DR-string, you can set continuation strings by defining the DR-array at the third subscript level. These subscripts should be sequential integers starting at 1. For example, the first continuation node of DR(2,16001.02) would be DR(2,16000.02,1); the second would be DR(2,16001.02,2), and so on.

#### 6. Detecting Up-Arrow Exits

You can determine, upon return from DIE, whether the user exited the routine by typing an up-arrow. If the user did so, the subscripted variable Y is defined; if all questions were asked and answered in normal sequence, \$D(Y) is zero.

#### 7. Editing a Subfile Directly

You can call ^DIE to directly edit an entry in a subfile; you can descend into as many subfiles as you need to. Set the DIE input variable to the full global root

leading to the subfile entry, including all intervening subscripts and the terminating comma, up to—but not including—the IEN of the subfile entry to edit. Then set an array element for each file and subfile level in the DA input variable, where DA=entry number in the subfile to edit, DA(1) is the entry number at the next higher file level,...DA(n) is the entry number at the file's top level.

For example, suppose that the data in subfile 16000.02 is stored descendent from subscript 20 and you are going to edit entry number 777, subentry number 1; you would write the following:

```
S DIE="^FILE(777,20," ; global root of subfile
 S DA(1)=777 ; entry number in file
 S DA=1 ; entry number in subfile
 S DR="3;7" ; fields in subfile to edit
D ^DIE
```

**NOTE:** The internal number of the entry into the file appears in the variable DIE and appears as the value of DA(1). When doing this, it is necessary that the subfile descriptor node be defined. In this example, it would be:

```
^FILE(777,20,0)="^16000.02^last number entered^number of entries"
```

## 8. Screening Variable Pointers

A variable pointer field can point to entries in more than one file. You can restrict the user's ability to input entries to certain files by setting the DIC("V") variable in a DR-string or in an INPUT template. It screens files from the user. Set DIC("V") equal to a line of M code that returns a truth value when executed. The code is executed after someone enters data into a variable pointer field. If the code tests false, the user's input is rejected; FileMan responds with ?? and a "beep."

The code setting the DIC("V") variable can be put into a DR-string or into an INPUT template. It is **not** a separate input variable for ^DIE or ^DIC. It should be set immediately before the variable pointer field is edited and it should be killed immediately after the field is edited.

When the user enters a value at a variable pointer field's prompt, FileMan determines in which file that entry is found. The variable Y(0) is set equal to information for that file from the data dictionary definition of the variable pointer field. You can use Y(0) in the code set into the DIC("V") variable. Y(0) contains the following:

| <b>^-Piece</b> | <b>Contents</b>                     |
|----------------|-------------------------------------|
| <b>Piece 1</b> | File number of the pointed-to file. |

- Piece 2** Message defined for the pointed-to file.
- Piece 3** Order defined for the pointed-to file.
- Piece 4** Prefix defined for the pointed-to file.
- Piece 5** y/n indicating if a screen is set up for the pointed-to file.
- Piece 6** y/n indicating if the user can add new entries to the pointed to file.

All of this information was defined when that file was entered as one of the possibilities for the variable pointer field.

For example, suppose field #5 is a variable pointer pointing to files 1000, 2000, and 3000. If you only want the user to be able to enter values from files 1000 or 3000, you could set up your INPUT template like this:

```
THEN EDIT FIELD: ^S DIC("V")="I +Y(0)=1000!(+Y(0)=3000)"
THEN EDIT FIELD: 5
THEN EDIT FIELD: ^K DIC("V")
```

## 9. Filing

DIE files data when any one of the following conditions is encountered:

- The field entered or edited is cross-referenced
- A change of level occurs, i.e., either DIE must descend into a multiple or ascend to the level above
- Navigation to another file occurs
- M code is encountered in one of the semicolon-pieces of the DR-string or in a template
- \$\$ becomes less than 2000
- The user up-arrows to a field
- The end of the DR-string or INPUT template is reached
- Templates are compiled and the execution is transferred from one routine to the next

## 10. New Style Compound Indexes and Keys

^DIE traditionally fires cross-references when the field on which the cross-reference is defined is edited. New-style cross-references that have an execution of "RECORD" (hereafter referred to as record-level indexes) are fired once at the end of the ^DIE call, after all the semicolon pieces of the DR string have been processed.

When record-level uniqueness indexes are fired, the corresponding keys (hereafter called record-level keys) are checked to ensure that they are unique. If edits to a field in a key result in a duplicate key, then changes to that field are backed out and an error message is presented to the user.

You can set the variable DIEFIRE in any of the semicolon-pieces of DR to instruct FileMan to fire the record-level indexes at that point and validate the corresponding record-level keys. You can also control what FileMan does if any of the record-level keys is invalid.

| <b>DIEFIRE contains:</b> | <b>Action:</b>                                           |
|--------------------------|----------------------------------------------------------|
| <b>M</b>                 | Print error message to user                              |
| <b>L</b>                 | Return the DIEBADK array (see example immediately below) |
| <b>R</b>                 | Restore invalid key fields to their pre-edited values    |

If DIEFIRE contains an L and a key is invalid, the DIEBADK array is set as follows:

```
DIEBADK(rFile#,key#,file#,IENS,field#,"O") = the original value of the field
DIEBADK(rFile#,key#,file#,IENS,field#,"N") = the new (invalid) value of the field
```

where,

|               |   |                                                                                                                                                                                                                               |
|---------------|---|-------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| <b>rFile#</b> | = | the <b>root file</b> of the <b>uniqueness index</b> of the key. This is the file or subfile number of the fields that make up the key.                                                                                        |
| <b>key#</b>   | = | the internal entry number of the key in the KEY file.                                                                                                                                                                         |
| <b>file#</b>  | = | the <b>file</b> of the <b>uniqueness index</b> of the key. This is the file or subfile where the <b>uniqueness index</b> resides. For whole file indexes, this is a file or subfile at a higher level than <b>root file</b> . |
| <b>IENS</b>   | = | the IENS of the record that—with the edits—would have a non-unique key.                                                                                                                                                       |
| <b>field#</b> | = | the field number of the field being edited.                                                                                                                                                                                   |

If any of the Keys is invalid, FileMan sets the variable X to the string "BADKEY", which can be checked by M code in the subsequent semicolon-piece of the DR string.

The variable X and the local array DIEBADK are available for use only in the semicolon piece immediately following the piece where the DIEFIRE was set.

For example:

```
S DIE="^FILE(",DA=777
S DR="@1;.01;.02;S DIEFIRE="R";I X="BADKEY"
 S Y="@1";1;2"
D ^DIE
```

Here, the .01 and .02 field makes up a key to the file. After prompting the user for the value of the .02, DIEFIRE is set to force VA FileMan to fire the record-level indexes and validate the key. If the key turns out to be invalid, FileMan sets X equal to "BADKEY" and, since DIEFIRE equals R, restores the fields to their pre-edited values. In the next semicolon-piece, we check if X equals "BADKEY" and, if so, branch the user back to the placeholder @1.

## **^DIEZ: Input/Compile**

Interactively compiles or recompiles an INPUT template.

Compiling an INPUT template means telling VA FileMan to write a hard-coded M routine that will do just what a particular INPUT template tells the Enter or Edit File Entries option to do. This can enhance system performance by reducing the amount of data dictionary lookup that accompanies VA FileMan input. The routines created by DIEZ should run from 20% to 80% more efficiently than DIE does for the same input.

Call ^DIEZ and specify the maximum number of characters you want in your routines, the name of the INPUT template you are using, and the name of the M routine you want to create. If more code is compiled than will fit into a single routine, overflow code will be incorporated in routines with the same name, followed by 1, 2, etc. For example, routine DGT may call DGT1, DGT2, etc.

Once DIEZ has created a hard-coded routine for a particular INPUT template, VA FileMan **automatically** uses that routine in the Enter or Edit File Entries option, whenever that template is specified for input. When definitions of fields used in the EDIT template are altered by the Modify File Attributes or Utility Functions option, the hard-code routine(s) is (are) recompiled immediately.

## **EN^DIEZ: Input/Compile**

This entry point **compiles** or **recompiles** an INPUT template, without user intervention. For more information about compiled INPUT templates, see ^DIEZ.

### **Input Variables**

- X**            The name of the routine for the compiled INPUT template.
- Y**            The internal entry number of the INPUT template to be compiled.
- DMAX**        The maximum size the compiled routines should reach. Consider using the \$\$ROUSIZE^DILF function to set this variable.

## **^DIK: Delete Entries**

Call DIK at ^DIK to delete an entry from a file.

**WARNING:** Use DIK to delete entries with extreme caution. It does not check Delete access for the file or any defined "DEL" nodes. Also, it does not update any pointers to the deleted entries. However, it does execute all cross-references and triggers.

### **Input Variables**

**DIK**            The global root of the file from which you want to delete an entry.

If you are deleting a subentry, set DIK to the full global root leading to the subentry, including all intervening subscripts and the terminating comma, up to—but not including—the IEN of the subfile entry to delete.

**DA**            If you are deleting an entry at the top level of a file, set DA to the internal entry number of the file entry to delete. For example, to delete SAM SMITH, who is entry number 7, from the EMPLOYEE file, stored in the global ^EMP, write the following:

```
S DIK=" ^EMP(" ,DA=7
D ^DIK
```

If you are deleting an entry in a subfile, set up DA as an array, where DA=entry number in the subfile to delete, DA(1) is the entry number at the next higher file level,...DA(n) is the entry number at the file's top level. For example, suppose employee JOHN JONES (record #1) has two skill entries (subrecords #1 and #2) in a SKILL multiple. To delete the SKILL multiple's subrecord #2 you would write:

```
S DA(1)=1,DA=2,DIK=" ^EMP("_DA(1)_" ,""SX"" , "
D ^DIK
```

where DA is the skill entry number in the subfile and DA(1) is the employee's internal entry number in the EMPLOYEE file.

## Looping to Delete Several Entries

^DIK leaves the DA-array and DIK defined. So you can loop through a file to delete several entries:

```
S DIK="^EMP(" F DA=2,9,11 D ^DIK
```

This deletes entries 2, 9 and 11 from the EMPLOYEE file.

## Deleting Fields from a File

As discussed in the How to Read an Attribute Dictionary section of the Global File Structure chapter, each attribute dictionary is also in the form of a file. You can therefore use the routine DIK to delete a *single-valued field* (i.e., not a multiple) from a file. To do this, the variable DIK is set to the file's data dictionary global node; DA is set to the number of the field to be deleted; and DA(1) is set to the file number. To delete the field SEX from our EMPLOYEE file example, simply write:

```
S DIK="^DD(3," ,DA=1,DA(1)=3
D ^DIK
```

When you use ^DIK to delete fields from a file, the data is not deleted.

**EN^DIK: Reindex****Reindexing Quick Reference**

| <b>Entry Point</b> | <b>Reindexes Entries</b> | <b>Reindexes Xrefs</b>  | <b>Executes Logic</b> |
|--------------------|--------------------------|-------------------------|-----------------------|
| <b>EN^DIK</b>      | 1                        | Some or all for 1 field | KILL then SET         |
| <b>EN1^DIK</b>     | 1                        | Some or all for 1 field | SET                   |
| <b>ENALL^DIK</b>   | All                      | Some or all for 1 field | SET                   |
| <b>IX^DIK</b>      | 1                        | All                     | KILL then SET         |
| <b>IX1^DIK</b>     | 1                        | All                     | SET                   |
| <b>IXALL^DIK</b>   | All                      | All                     | SET                   |

EN^DIK reindexes one or more cross-references of a field for one entry in a file. It executes the KILL logic first and then executes the SET logic of the cross-reference. Before reindexing, you should be familiar with the effects of all relevant cross-references that could be fired (including bulletins, triggers, and MUMPS-type).

**Input Variables**

- DIK** If you are reindexing an entry at the top level of a file, set DIK to the global root of the file.
- If you are reindexing a subentry, set DIK to the full global root leading to the subentry, including all intervening subscripts and the terminating comma, up to—but not including—the IEN of the subfile entry to reindex.
- DA** If you are reindexing an entry at the top level of a file, set DA to the internal entry number of the file entry to reindex.
- If you are reindexing an entry in a subfile, set up DA as an array, where DA=entry number in the subfile to reindex, DA(1) is the entry number at the next higher file level,...DA(n) is the entry number at the file's top level.

**DIK(1)** Use the field number (to get all indexes) or the field number *and* specific indexes of the cross-reference. See the ENALL^DIK entry point description for examples.

**EN1^DIK: Reindex****Reindexing Quick Reference**

| <b>Entry Point</b> | <b>Reindexes Entries</b> | <b>Reindexes Xrefs</b>  | <b>Executes Logic</b> |
|--------------------|--------------------------|-------------------------|-----------------------|
| <b>EN^DIK</b>      | 1                        | Some or all for 1 field | KILL then SET         |
| <b>EN1^DIK</b>     | 1                        | Some or all for 1 field | SET                   |
| <b>ENALL^DIK</b>   | All                      | Some or all for 1 field | SET                   |
| <b>IX^DIK</b>      | 1                        | All                     | KILL then SET         |
| <b>IX1^DIK</b>     | 1                        | All                     | SET                   |
| <b>IXALL^DIK</b>   | All                      | All                     | SET                   |

EN1^DIK reindexes one or more cross-references of a field for one entry in a file. It **only** executes the SET logic of the cross-reference.

Before reindexing, you should be familiar with the effects of all relevant cross-references that could be fired (including bulletins, triggers, and MUMPS-type).

**Input Variables**

- DIK** If you are reindexing an entry at the top level of a file, set DIK to the global root of the file.
- If you are reindexing a subentry, set DIK to the full global root leading to the subentry, including all intervening subscripts and the terminating comma, up to—but not including—the IEN of the subfile entry to reindex.
- DA** If you are reindexing an entry at the top level of a file, set DA to the internal entry number of the file entry to reindex.

If you are reindexing an entry in a subfile, set up DA as an array, where DA=entry number in the subfile to reindex, DA(1) is the entry number at the next higher file level,...DA(n) is the entry number at the file's top level.

**DIK(1)** Use the field number (to get all cross-references) or the field number *and* specific indexes of the cross-references you want. See the ENALL^DIK entry point description for examples.

**ENALL^DIK: Reindex****Reindexing Quick Reference**

| <b>Entry Point</b> | <b>Reindexes Entries</b> | <b>Reindexes Xrefs</b>  | <b>Executes Logic</b> |
|--------------------|--------------------------|-------------------------|-----------------------|
| <b>EN^DIK</b>      | 1                        | Some or all for 1 field | KILL then SET         |
| <b>EN1^DIK</b>     | 1                        | Some or all for 1 field | SET                   |
| <b>ENALL^DIK</b>   | All                      | Some or all for 1 field | SET                   |
| <b>IX^DIK</b>      | 1                        | All                     | KILL then SET         |
| <b>IX1^DIK</b>     | 1                        | All                     | SET                   |
| <b>IXALL^DIK</b>   | All                      | All                     | SET                   |

ENALL^DIK reindexes all entries in a file for the cross-references on a specific field. It may also be used to reindex all entries within a single subfile, that is a subfile corresponding to only one of the file's entries. ENALL^DIK only executes the SET logic.

Before reindexing, you should be familiar with the effects of all relevant cross-references that could be fired (including bulletins, triggers, and MUMPS-type).

**NOTE:** IXALL^DIK, ENALL^DIK, and the Re-Index File option on the Utility Functions menu set the 3rd piece of the 0 node of the file's global root (the file header) to the last internal entry number used in the file. They set the 4th piece to the total number of entries in the file.

**Input Variables**

**DIK** If you are reindexing an entry at the top level of a file, set DIK to the global root of the file.

If you are reindexing subentries, set DIK to the full global root leading to the subentry, including all intervening subscripts and the terminating comma, up to—but not including—the iens of the subfile entries to reindex.

**DA(1..n)** If you are reindexing entries in a subfile, set up DA as an array, where DA(1) is the entry number at the next higher file level,...DA(n) is the entry number at the file's top level. Since ENALL^DIK reindexes all entries at a given file level, don't set the unsubscripted DA node.

**DIK(1)** Use the field number (to get all indexes) or the field number *and* specific cross-references separated by up-arrows as shown below:

```
S DIK(1)="FLD#" ;Just the field number to get all indexes.
```

OR:

```
;Field number followed by x-ref name or number.
```

```
S DIK(1)="FLD#^INDEX"
```

```
;See the examples below:
```

```
S DIK(1)=".01^B"
```

```
S DIK(1)=".01^B^C"
```

```
S DIK(1)=".01^1^2"
```

```
D ENALL^DIK
```

**IX^DIK: Reindex****Reindexing Quick Reference**

| <b>Entry Point</b> | <b>Reindexes Entries</b> | <b>Reindexes Xrefs</b>  | <b>Executes Logic</b> |
|--------------------|--------------------------|-------------------------|-----------------------|
| <b>EN^DIK</b>      | 1                        | Some or all for 1 field | KILL then SET         |
| <b>EN1^DIK</b>     | 1                        | Some or all for 1 field | SET                   |
| <b>ENALL^DIK</b>   | All                      | Some or all for 1 field | SET                   |
| <b>IX^DIK</b>      | 1                        | All                     | KILL then SET         |
| <b>IX1^DIK</b>     | 1                        | All                     | SET                   |
| <b>IXALL^DIK</b>   | All                      | All                     | SET                   |

IX^DIK reindexes all cross-references of the file for only one entry in the file. It executes first the KILL logic and then the SET logic. Reindexing occurs at all file levels at or below the one specified in DIK and DA.

Before reindexing, you should be familiar with the effects of all relevant cross-references that could be fired (including bulletins, triggers, and MUMPS-type).

**Input Variables**

- DIK** If you are reindexing an entry at the top level of a file, set DIK to the global root of the file.
- If you are reindexing only a subentry, set DIK to the full global root leading to the subentry, including all intervening subscripts and the terminating comma, up to—but not including—the IEN of the subfile entry to reindex.
- DA** If you are reindexing an entry at the top level of a file, set DA to the internal entry number of the file entry to reindex.
- If you are reindexing an entry in a subfile, set up DA as an array, where DA=entry number in the subfile to reindex, DA(1) is the entry

number at the next higher file level,...DA(n) is the entry number at the file's top level.

**IX1^DIK: Reindex****Reindexing Quick Reference**

| <b>Entry Point</b> | <b>Reindexes Entries</b> | <b>Reindexes Xrefs</b>  | <b>Executes Logic</b> |
|--------------------|--------------------------|-------------------------|-----------------------|
| <b>EN^DIK</b>      | 1                        | Some or all for 1 field | KILL then SET         |
| <b>EN1^DIK</b>     | 1                        | Some or all for 1 field | SET                   |
| <b>ENALL^DIK</b>   | All                      | Some or all for 1 field | SET                   |
| <b>IX^DIK</b>      | 1                        | All                     | KILL then SET         |
| <b>IX1^DIK</b>     | 1                        | All                     | SET                   |
| <b>IXALL^DIK</b>   | All                      | All                     | SET                   |

IX1^DIK reindexes all cross-references of the file for only one entry in the file. It **only executes the SET logic** of the cross-reference. Reindexing occurs at all file levels at or below the one specified in DIK and DA.

Before reindexing, you should be familiar with the effects of all relevant cross-references that could be fired (including bulletins, triggers, and MUMPS-type).

**Input Variables**

- DIK** If you are reindexing an entry at the top level of a file, set DIK to the global root of the file.
- If you are reindexing a subentry, set DIK to the full global root leading to the subentry, including all intervening subscripts and the terminating comma, up to but not including the IEN of the subfile entry to reindex.
- DA** If you are reindexing an entry at the top level of a file, set DA to the internal entry number of the file entry to reindex.

If you are reindexing an entry in a subfile, set up DA as an array, where DA=entry number in the subfile to reindex, DA(1) is the entry number at the next higher file level,...DA(n) is the entry number at the file's top level.

**IXALL^DIK: Reindex****Reindexing Quick Reference**

| <b>Entry Point</b> | <b>Reindexes Entries</b> | <b>Reindexes Xrefs</b>  | <b>Executes Logic</b> |
|--------------------|--------------------------|-------------------------|-----------------------|
| <b>EN^DIK</b>      | 1                        | Some or all for 1 field | KILL then SET         |
| <b>EN1^DIK</b>     | 1                        | Some or all for 1 field | SET                   |
| <b>ENALL^DIK</b>   | All                      | Some or all for 1 field | SET                   |
| <b>IX^DIK</b>      | 1                        | All                     | KILL then SET         |
| <b>IX1^DIK</b>     | 1                        | All                     | SET                   |
| <b>IXALL^DIK</b>   | All                      | All                     | SET                   |

IXALL^DIK reindexes **all** cross-references for **all** entries in a file. It only executes the SET logic.

Before reindexing, you should be familiar with the effects of all relevant cross-references (including bulletins, triggers, and MUMPS-type) that could be fired.

**NOTE:** IXALL^DIK, ENALL^DIK, and the Re-Index File option on the Utility Functions menu set the 3rd piece of the 0 node of the file's global root (the file header) to the last internal entry number used in the file. They set the 4th piece to the total number of entries in the file.

**Input Variable**

**DIK**        The global root of the file to be indexed.

**Examples****Example 1**

A simple call to reindex the EMPLOYEE file would be:

```
>S DIK="^EMP(" D IXALL^DIK
```

**Example 2**

The reindexing of data dictionary #3 would be:

```
>S DA(1)=3,DIK="^DD(3," D IXALL^DIK
```

## **^DIKZ: Cross-reference Compilation**

Cross-references can be compiled into M routines by calling ^DIKZ. You will be prompted to specify the maximum routine size and the name or number of the file. If you specify the routine name XXX and more code is generated than can fit into that one routine, overflow routines (XXX1, XXX2, etc.) will be created. Routine XXX may call XXX1, XXX2, etc.

Once DIKZ has been used to create hard-coded cross-reference routines, those routines are used when calls to any entry point in DIK are made. However, if you restrict the cross-references to be reindexed by using the DIK(1) variable, the compiled routines are not used. As soon as data dictionary cross-references are added or deleted, the routines are recompiled. The purpose of this DIKZ code generation is simply to improve overall system throughput.

See the Edit File section of the *VA FileMan Advanced User Manual* for instructions on permanently stopping the use of compiled cross-references, uncompiling cross-references.

## **EN^DIKZ: Compile**

EN^DIKZ recompiles a file's cross-references by setting the input variables without user intervention.

### **Input Variables**

- |             |                                                                                                                         |
|-------------|-------------------------------------------------------------------------------------------------------------------------|
| <b>X</b>    | The routine name.                                                                                                       |
| <b>Y</b>    | The file number of the file for which you want the cross-references recompiled.                                         |
| <b>DMAX</b> | The maximum size the compiled routines should reach. Consider using the \$\$ROUSIZE^DILF function to set this variable. |

## **\$\$ROUSIZE^DILF: Routine Size**

This argumentless function returns the maximum routine size that should be used when compiling cross-references, print templates, or input templates.

### **Format**

```
$$ROUSIZE^DILF
```

### **Input Parameters**

None

### **Output**

This function returns the maximum routine size defined in the MUMPS OPERATING SYSTEM file (#.7).

### **Example**

```
>W $$ROUSIZE^DILF
4000
```

## **^DIM: M Code Validation**

Call ^DIM to validate any line of M code. ^DIM checks that code conforms to the 1995 ANSI Standard. Code is also checked against aspects of VHA's Programming Standards and Conventions (SAC).

**NOTE:** ^DIM does not allow killing an unsubscripted global.

### **Input Variable**

**X**            Invoke ^DIM with the line to be validated in the local variable X.

### **Output Variable**

**X**            ^DIM either kills X or leaves it unchanged. If \$D(X) is zero on return from ^DIM, the line of code is invalid. However, the converse is not always true; in other words, ^DIM is not as smart as a real M interpreter and sometimes validates strings when it should not.

## DT^DIO2: Date/Time Utility

This entry point takes an internal date in the variable Y and *writes out* its external form.

### Example

```
>S Y=2690720.163 D DT^DIO2
JUL 20,1969 1630
```

This results in Y being equal to JUL 20,1969 16:30. (No space before the 4-digit year; 2 spaces before the hours [1630].)

### Input Variable

**Y** (Required) This contains the internal date to be converted. Y is required and it is not changed.

In addition, see X ^DD("DD") and DD^%DT, which also convert a date from internal YYYYMMDD format to external format.

## **^DIOZ: Sort/Compile**

This entry point marks a SORT template compiled or uncompiled. The ^DIOZ entry point asks for the name of the SORT template to be used and whether the user wishes (1) to mark it compiled or (2) to uncompile it if it is already marked compiled. Actual compilation occurs at the time the template is used in the sort/print. There are no input or output variables.

SORT templates can be compiled into M routines to increase efficiency of the sort and improve system performance. Good candidates for compilation are sorts with many sort fields or those that sort on fields reached with relational syntax. The process of sort compilation is different from other FileMan compiling activities. SORT templates can be "marked" for compilation, then each time the SORT template is used in a FileMan sort/print, a new compiled routine is created. When the print job finishes, the routine is deleted. The routine is named DISZnnnn where "nnnn" is a four-digit number. The routine names are reused. Routine numbers are taken from the Compiled Routine file (described in the section on the ENRLS^DIOZ utility in the *VA FileMan Advanced User Manual*). Thus, a routine name is not tied to a particular SORT template.

## EN1^DIP: Print Data

Use EN1^DIP to print a range of entries, in columnar format.

### Input Variables

#### Required

**L** (Required) A required variable which should be set to zero or some string whose numeric evaluation is zero, e.g., "LIST DRUGS". If set to a text string, the string is used to replace the word "SORT" in the "SORT BY:" prompts, when FileMan asks the user for sort values:

```
LIST DRUGS BY: NAME//
```

**DIC** (Required) The open global root of the file in the usual format, e.g., "^DIZ(16540," or the file number.

#### Optional: Sorting and Print Fields

**FLDS** (Optional) The various fields to be printed. If this parameter is not sent, the user will be prompted for fields to print. FLDS can contain the following:

- The numbers or names of the fields to be printed, separated by commas. These fields are printed in the order that they are listed. Print qualifiers which determine column width, caption contents, and many other features of the output may be included exactly as they are when answering the "PRINT FIELD:" prompt. (See the Print chapter in the *VA FileMan Getting Started Manual* for details on print qualifiers.) For example:

```
FLDS=" .01, .03,1;C20"
```

If there are more fields than can fit on one string, FLDS can be subscripted (FLDS(1), FLDS(2), and so forth), but FLDS as a single-valued variable must exist.

- The name of a PRINT template preceded by an open bracket ([]) and followed by a close bracket (]). For example:

```
FLDS=" [DEMO] "
```

## **BY**

(Optional) The fields by which the data is to be sorted. If BY is undefined, the user is prompted for the sort conditions. You can sort by up to 7 fields; that is, you can have up to a 7-level sort.

You can set BY to:

- The numbers or names of the fields separated by commas. Sort qualifiers which determine aspects of the sort and of the printout may be included exactly as they are when answering the "SORT FIELD:" prompt. For example:

```
BY=" .01 ; C1 , 1 "
```

If one of the comma pieces of the BY variable is the @-sign character, the user will be asked for that SORT BY response. So if you want to sort by DIAGNOSIS but allow the user to order the sort within DIAGNOSIS, set BY="DIAGNOSIS,@".

- The name of a SORT template preceded by an open bracket ([]) and followed by a close bracket (]). For example:

```
BY=" [DEMOSORT] "
```

**NOTE:** You cannot use the name of a SORT template in the BY variable if the BY(0) input variable has been set. If you want to create such complex sorts, you can include the BY(0) information within the SORT template. See the section Storing BY(0) Specifications in SORT Templates, within the Details and Features section of Controlling Sorts with BY(0) at the end of this call.

The name of a SEARCH template, preceded by an open bracket ([]) and followed by a close bracket (]). The SEARCH template must have results stored in it. Only those records in the SEARCH template will print, and they will print in IEN order. For example:

```
BY=" [DEMOSEARCH] "
```

**NOTE:** If more than one field is included in the BY variable, separate the fields with commas. The same comma-pieces will identify the field in the FR and TO variables. If, for example, you wanted a sorted report of entries with DOBs in 1960 and with ZIP CODEs in the 90000s, you could define the variables by writing:

```
BY="DOB,ZIP CODE"
FR="01/01/60,90000"
TO="12/31/60,99999"
```

Since the delimiter of BY is a comma, the value placed in the variable should not contain a comma. Therefore, if your field name contains a comma, use the field number in the BY variable instead of its name. For the same reason, if sort from or to values contain commas, the alternate FR(n) and TO(n) input arrays described below should be used instead of the FR and TO input variables.

## FR

(Optional) The START WITH: values of the SORT BY fields. If FR is undefined, the user will be asked the START WITH: question for each SORT BY field. If FR is defined, it consists of one or more comma pieces, where the piece position corresponds to the order of the sort field in the BY variable. Each comma piece can be:

- The value from which the selection of entries will begin.
- Null. If a comma piece of FR is null, then the sort will start from the very beginning of the file for that field.
- ?. The question mark as one of the comma pieces causes the "START WITH:" prompt to be presented to the user for the corresponding SORT BY field.
- @. The at-sign indicates that the sort should begin with null values, that is, with entries that have no data on file. If the corresponding piece of the TO variable or array also is set to @, then only entries with null values for this sort field will be selected during the sort. If TO does not contain @, then after the null values, the sort will start at the first non-null value and will go to the value indicated by TO.

**NOTE:** If BY contains the name of a SORT template and if the developer answered NO to the question SHOULD TEMPLATE

USER BE ASKED 'FROM'-'TO' RANGE... for a field at the time the template was defined, then the information in the FR and TO variables is ignored for that field. Instead, the from/to ranges stored in the sort template are used.

If you customize sorts using BY(0), see special note on FR in that section at the end of this call.

**FR(n)**

(Optional) An alternate way to provide the START WITH: values of the SORT BY fields. If FR is defined, it will override this array. The subscript n corresponds to the comma piece in the BY variable (i.e., the sort by field number). This alternate way of inputting the from and to values allows the use of values containing commas, such as PATIENT NAMEs. Each nth entry in the array corresponds to, and can have the same value as, the nth comma piece in the FR variable. The only difference is that any nth entry, FR(n), can be undefined, causing the START WITH: question to be asked for the nth SORT FIELD.

For example, if you were using the unsubscripted TO and FR variables to do a sort on two fields, you might do as follows:

```
S FR="A,01/01/95",TO="Zz,01/31/95"
```

To set up the same sort using the subscripted forms of TO and FR, you would set them up as follows:

```
S FR(1)="A",FR(2)="01/01/95"
S TO(1)="Zz",TO(2)="01/31/95"
```

**NOTE:** If you customize sorts using BY(0), see special note on FR in that section at the end of this call.

**TO**

(Optional) The GO TO: values of the SORT BY fields. Its characteristics correspond to the FR variable. If undefined, the user will be asked the GO TO: questions for each SORT BY field. If TO is defined, it consists of one or more comma pieces. Each comma piece can be:

- The value at which the selection of entries will end.
- Null. If TO is null, then the sort will go from FR to the end of the file.

- ?. The question mark as one of the comma pieces causes the "GO TO:" prompt to be presented to the user for the corresponding SORT BY field.
- @. The at-sign indicates that the sort should include null values, that is, entries that have no data on file. If the corresponding piece of the FR variable or array also is set to @, then only entries with null values for this sort field will be selected during the sort. If FR does not contain @, then after the null values, the sort will start at the FR value and include all other non-null values to the end of the file.

**NOTE:** If BY contains the name of a SORT template and if the developer answered NO to the question SHOULD TEMPLATE USER BE ASKED 'FROM'-'TO' RANGE... for a field at the time the template was defined, then the information in the FR and TO variables is ignored for that field. Instead, the from/to ranges stored in the SORT template are used.

### **TO(n)**

(Optional) An alternate way to provide the GO TO: values of the SORT BY fields. If TO is defined, it will override this array. The subscript "n" corresponds to the comma piece in the BY variable. This alternate way of inputting the from and to values allows the use of values containing commas, such as PATIENT NAMES. Each nth entry in the array corresponds to, and can have the same value as, the nth comma piece in the TO variable. The only difference is that any nth entry, TO(n), can be undefined, causing the GO TO: question to be asked for the nth SORT BY field.

If you customize sorts using BY(0), see special note on TO(n) in that section at the end of this call.

### **Optional: Miscellaneous Features**

### **DHD**

(Optional) The header desired for the output. DHD can be one of the following:

- @ if header is not desired.
- @@ if header **and** formfeed are not desired.

- A literal which will be printed, as is, in the upper left hand corner of the printout. The date, page and field headings will be in their normal places.
- A line of M code which must begin with a write statement, e.g., `DHD="W ?0 D ^ZZHDR"`.
- A PRINT template name preceded by an open bracket ([]) and followed by a close bracket (]). In this case, the template replaces all parts of the header that VA FileMan normally generates.
- Two PRINT templates separated by a minus sign. The first will be used as the header and the second will be used as the trailer. For example:

```
DHD=" [HEADER] - [TRAILER] "
```

|                |                                                                                                                                                                                                                                                                 |
|----------------|-----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| <b>DIASKHD</b> | (Optional) If this variable is defined, the user will be prompted to enter a header. Set it equal to null (""). If this variable is undefined, the user will not have the opportunity to change the header on the print.                                        |
| <b>DIPCRIT</b> | (Optional) If this variable is set to 1, the SORT criteria will print in the header of the first page of the report.                                                                                                                                            |
| <b>PG</b>      | (Optional) Starting page number. If variable is undefined, page 1 will be assumed.                                                                                                                                                                              |
| <b>DHIT</b>    | (Optional) A string of M code which will be executed for every entry after all the fields specified in FLDS have been printed.                                                                                                                                  |
| <b>DIOEND</b>  | (Optional) A string of M code which is executed after the printout has finished but before returning to the calling program.                                                                                                                                    |
| <b>DIOBEG</b>  | (Optional) A string of M code which is executed before the printout starts.                                                                                                                                                                                     |
| <b>DCOPIES</b> | (Optional) If %ZIS chooses an SDP device, and if multiple copies are desired, you can call for them by setting DCOPIES equal to the number (greater than one) of copies desired. For more information about SDP devices, see the <i>Kernel Systems Manual</i> . |

**IOP**

(Optional) EN1^DIP calls the ^%ZIS entry point to determine which device output should go to. This requires user interaction unless you preanswer the DEVICE prompt. You can do this by setting IOP equal to the name of the device (as it is stored in the DEVICE file) to which the output should be directed. You can also set IOP in any of the additional formats recognized by ^%ZIS to specify the output device (see the *Kernel Systems Manual* for more information on ^%ZIS and IOP).

If you need to call ^%ZIS beforehand to obtain the name of the device in question from the user, call it with the %ZIS N flag set so that ^%ZIS doesn't actually open the device. The name of the device is then returned in the ION output variable. EN1^DIP will open and close the device you specify in IOP on its own; don't open it yourself beforehand.

In addition to setting IOP equal to a device for printing, you can use this variable (in conjunction with the DQTIME variable described immediately below) to queue the printing of a report. This functionality is only available if Kernel is present. Also, you must set up all of the input variables for EN1^DIP so that the user is not asked any questions. For example, the BY, FR, and TO variables must be defined. To establish queuing, IOP should equal Q;output device. For example:

```
S IOP="Q;MY PRINTER - NLQ".
```

**DQTIME**

(Optional) If output is queued, this variable contains the time for printing. You can set it equal to any value that %DT recognizes. For example:

```
S DQTIME="NOW"
```

OR:

```
S DQTIME="T@11PM"
```

**DIS(0)**

(Optional) You can screen out certain entries so that they do not appear on the output by setting the optional array DIS. The first subscript in this array can be 0 (zero). This variable

(as well as all the others) contains an executable line of M code which includes an IF-statement. If the execution of the IF sets

\$T to 1, then the entry will print. The internal number of the entry being processed is in D0.

**DIS(n)** (Optional) You can set other elements in the DIS array: DIS(1), DIS(2), DIS(3), etc. The subscripts must be consecutive integers starting at 1. Again, they must contain M code that sets \$T. If many elements are defined, then DIS(0) (if it exists) must be true and any one of the other elements in the array must be true for the entry to print.

**DISUPNO** (Optional) If this variable is set to 1 and if no records are found within the sort ranges specified for the print, the report header and the "No Records to Print" message is not printed.

**DISTOP** (Optional) If Kernel is present, by default, prints queued through the EN1^DIP call can be stopped by the user with a TaskMan option. However, if this variable is set to 0, users will not be able to stop their queued prints.

DISTOP can also be set equal to M code that will be executed once near the start of a queued print. If the code sets \$T to true, the user will be able to stop the job; if \$T is false, the user will not be able to. For example:

```
S DISTOP="I DUZ(0)="@"
```

would mean that only those with programmer access could stop the print.

**DISTOP("C")** (Optional) If the user stops a queued print job by using TaskMan's option, code in this optional variable will be executed before the output device is closed. It might, for example, do clean up necessary because the job did not run to completion.

### Optional: Controlling Sorts with BY(0)

**BY(0)** See the section called CONTROLLING SORTS WITH BY(0) (In Detail) at the end of this call for more information.

**L(0)**

**FR(0,n)**

**TO(0,n)**

**DISPAR(0,n)**

**DISPAR(0,n,"  
OUT")**

## Output Variables

None

**NOTE:** Unlike most calls, EN1^DIP kills all the input variables before it quits. You do not have to kill them.

## Details and Features

### Input Variables to Control Sorts

You can use a special set of input variables to:

- ◆ Preselect a set of records for printing.
- ◆ Preselect the order that these records should be printed in.

The set of variables for controlling sorts is:

BY(0), L(0), FR(0,n), TO(0,n), DISPAR(0,n), and DISPAR(0,n,"OUT")

Please see the Controlling Sorts with BY(0) section at the end of this call for more information.

### Setting up BY, FR, and TO Variables to Sort within a Multiple

If you have a file like:

```
.01 PARENT NAME
1 SPOUSE (mult.)
 .01 SPOUSE NAME
 1 SPOUSE DOB
 2 CHILDREN (mult.)
 .01 CHILDS NAME
 1 CHILDS DOB
 2 CHILDS SEX
 3 CHILDS NICKNAME
 2 PARENT NICKNAME
```

And you wish to sort on the NICKNAME field for CHILDREN, from "A" to "Z", then by the PARENT

NICKNAME field from "B" to "E". You set:

```
BY = "1,2,3,2"
FR = "A,B"
TO = "Z,E"
```

You must put in all field numbers to get down to the multiple in the BY (1,2,3), but then it pops you out of the multiple so that the following number '2' in the BY gets you field 2 at the top level (PARENT NICKNAME), rather than field 2 within the lowest multiple (SEX).

But note the FR and TO: here you just put the starting and ending values for the two fields on which you wish to sort.

**NOTE:** This same logic does not work on the FLDS multiple. It is suggested that in order to print fields within a multiple, the print logic should be set up in a PRINT template.

### **Using EN1^DIP to Print Audit Trails**

The audit files are structured differently than other FileMan files. To print audit trails for a file's data or Data Dictionary, the DIC variable must contain the global location of the requested audit file and the file number of the file that was audited as the open root.

To print a data audit trail for File #662001, set DIC="^DIA(662001, ". To print the DD audit trail, set DIC="^DDA(662001, ". The other input variables are set as for a normal print. Remember that the fields being printed and sorted come from the audit files, not from the file for which the audit trail was recorded.

### **EN1^DIP: CONTROLLING SORTS WITH BY(0) (In Detail)**

Ordinarily, you control the way EN1^DIP sorts output using the BY, FR, and TO input variables. This lets you sort based on field values, a previous sort stored in a SORT template, or on the records stored in a SEARCH template.

The BY(0) feature allows you to control the sort. With BY(0), you can force VA FileMan to sort using an existing compound index (i.e., one that indexes more than a single data field) for efficiency. Or, use of BY(0) allows you to pre-sort a list of record numbers in a global and pass this pre-sorted list to EN1^DIP. This lets you pre-sort reports in any way that you can use subscripts to sort a global. The only

limitation is that the total number of subscripts in the global that you sort by must be seven or less.

The two main ways in which the BY(0) feature should be used are as follows:

- Set BY(0) to the global location of an **existing FileMan index**. In particular, this lets you sort based on a MUMPS cross-reference or a compound cross-reference defined on the INDEX file (not possible otherwise). Since the sorting is already done in advance, any such prints are very fast.
- Set BY(0) to the global location of a list of records you create "**on the fly**." This lets you sort the records in any order you want, and also lets you easily limit the number of records by pre-selecting them.

### Input Variables for Sorting with BY(0)

**BY(0)** (Optional; Required for BY(0) sorts) Set this variable to an open global root. The open global root should be the static part of a global; a list of record numbers must be stored at a descendent subscript level.

```

^DIZ(662001,"E","ALBERT",1009)
^DIZ(662001,"E","ANDREA",339)
^DIZ(662001,"E","ANDREW",552)

<-static part-> <-dynamic part->

```

In the example just above, you would set BY(0) to '^DIZ(662001,"E",.'

There can be intervening subscript levels between the static, fixed global root and the subscript level where the list of records numbers is stored. Any intervening subscript levels define a sort order. Use the L(0) input variable to tell FileMan the number of dynamic subscript levels it needs to sort through (see L(0) description below).

Alternatively, you can set BY(0) to the name of a SEARCH template, in [brackets]. This tells VA FileMan to sort on the list of record numbers contained in the corresponding SEARCH template entry in the ^DIBT global.

BY(0) affects your sorts as follows:

It restricts the possible records for printing to those in the specified list.

When you set BY(0) to a static global reference, each intervening subscript level (between the static part of the global reference and the subscript level containing record numbers) defines a sort level, starting from the highest intervening subscript level.

### **BY(0) for a VA FileMan Index**

If you set BY(0) to sort based on an existing FileMan-maintained cross-reference, make sure the subscript you set L(0) to point to is in fact the location where FileMan stores its list of records (when sorting on a regular single-field index, L(0) should be 2).

### **BY(0) for a List of Records "On the Fly"**

If you build your own list of sorted records on the fly in a temporary global (as opposed to setting BY(0) to a VA FileMan-maintained cross-reference) it's best not to let the final subscript of your static global reference be "B". For more information, see the discussion in the Details and Features section below.

NOTE: If you are using both the BY and BY(0) input variables, don't set BY to the name of a template; an error message will print or hard errors could result.

## **L(0)**

(Optional; Required if BY(0) is set to an open global root.)

Use L(0) to specify the number of dynamic subscript levels that exist beyond the static global root, including the subscript level containing the list of record numbers. The minimum value of L(0) is 1.

EN1^DIP lets you sort by up to 7 subscripts; therefore the maximum value of L(0) is 8.

For example, if BY(0) refers to a regular "E" index on a file -- '^DIZ(662001,"E",' -- you should set L(0)=2 -- that is, one for the subscript containing the (dynamic) value

of the field being cross-referenced, plus one for the record number.

**FR(0,n)**

(Optional) To select only a subset of records at a given subscript level "n", you can use FR(0,n) and/or TO(0,n). For "n" equal to any of the "n" dynamic sorting subscript levels in the global specified by BY(0), you can set FR(0,n) to the sort-from value for that subscript level.

This restricts the printed records to those whose subscript values at subscript level n sort the same or greater than the value you set into FR(0,n). If FR(0,n) is undefined for any subscript n, the sort on that subscript level begins with the first value for that subscript.

**NOTE:** These values must be in internal format, as they are stored in the subscript of the index or global defined by BY(0).

**TO(0,n)**

(Optional) This variable contains the ending value (the sort-to value) for any of the "n" dynamic sorting subscripts in the global specified by BY(0). If TO(0,n) is undefined for any subscript "n", the sort on that subscript level ends with the last value for that subscript.

**NOTE:** These values must be in internal format, as they are stored in the subscript of the index or global defined by BY(0).

**DISPAR(0,n)**

(Optional) Like the FR(0,n) and TO(0,n) variables, this variable array can be set for any of the "n" dynamic sorting subscripts in the global specified by BY(0). This array allows you to create subheaders for the sorting subscripts in the global. In order to create a sub-header, you must define a title for the subscript, as VA FileMan has no knowledge of the subscripts. Each entry in the array can have information in two ^-pieces.

The first piece contains the sort qualifiers that are normally entered interactively before a sort field (see the User Manual for more information.) Two of the sort

qualifiers can be used here: "!" to number the entries by sort value and "#" to page break when the sort values changes.

The second piece contains the sort qualifiers that are normally entered interactively after the sort field. In order to print a subheader, you must enter literal subheader "caption" (e.g., ;"Station/PO Number: "). To have no subheader text other than the subheader value, use a null caption (e.g., ;"). You can also use the sort qualifiers ;Cn ;Ln or ;Sn, (see the User *Getting Started Manual* for more information.)

The subheaders defined in DISPAR(0,n) cannot be suppressed.

**DISPAR(0,n,"OUT")** (Optional) If a literal title is input to DISPAR(0,n) above, then you can also enter M code to transform the value of the subscript from the global before it is printed as a subheader. It acts like an OUTPUT transform. At the time of execution, the untransformed value will be in Y. The code should put the transformed value back into Y. Any other variables used in the code should be NEWed.

## Examples

### Example 1

Suppose you have a simple MUMPS cross-reference that inverts dates so that the values in the cross-reference are 99999999-date. The cross-reference might look something like:

```
^DIZ(662001,"AC",97069889,2)=" "
^DIZ(662001,"AC",97969898,3)=" "
^DIZ(662001,"AC",97969798,1)=" "
...etc.
```

If you wanted to sort all entries by this inverse date and to convert the date values into a readable format for the subheader, you would set up the variables for the EN^DIP call like this:

```
>S DIC="^DIZ(662001,",L=0,FLDS="your field list"
>S BY(0)="^DIZ(662001,""AC"","
```

```
>S L(0)=2
>S DISPAR(0,1)="^;" "DATE""
>S DISPAR(0,1,"OUT")="S:Y Y=99999999-Y S Y=$$FMTE^XLFD(Y)"
```

## Example 2

Suppose you have a list of record numbers in a global that looked like this:

```
^TMP($J,1)=" "
^TMP($J,3)=" "
^TMP($J,35)=" "
^TMP($J,39)=" "
...etc.
```

If you wanted to print those records sorted by the .01 field of the file, you would:

```
>S DIC="^DIZ(662001," ,L=0,BY=.01,(FR,TO)="",FLDS="your
 field list"
>S BY(0)="^TMP($J,"
>S L(0)=1
```

## Example 3

Suppose you have a MUMPS multifield-style cross-reference, with subscripts based on the values of two fields. The first field in the subscript is free-text, and the second is a number. The cross-reference might look like:

```
^DIZ(662001,"AD","ANY",4.99,5)=" "
^DIZ(662001,"AD","ANYTHING",1.3,2)=" "
^DIZ(662001,"AD","ANYTHING",1.45,1)=" "
^DIZ(662001,"AD","SOMETHING",.4,10)=" "
...etc.
```

You want to sort from value "A" to "AZ" on the free-text field and from 1 to 2 on the numeric field. Also, you want to print a subheader for the numeric field. You could set your variables like this:

```
>S DIC="^DIZ(662001," ,L=0,FLDS="your field list"
>S BY(0)="^DIZ(662001," "AD" ,"
>S L(0)=3
>S FR(0,1)="A",TO(0,1)="AZ"
>S FR(0,2)=1,TO(0,2)=2
>S DISPAR(0,2)="^;" "NUMBER""
>S DISPAR(0,2,"OUT")="S Y=$J(Y,2)"
```

## Details and Features

### **Sorting on MUMPS Cross-references**

The BY(0) feature is designed to let you pre-sort your FileMan reports using MUMPS cross-references. As long as the MUMPS cross-reference has 0 to 7 dynamic (sorting) subscripts, followed by the record numbers stored in a final subscript level, you can order your reports based on that cross-reference using BY(0).

While you may have used MUMPS cross-references in the past only for sorting hard-coded reports, you may want to consider using them with FileMan-based reports as well.

### **Sorting a Compound Cross-reference Defined in the INDEX file**

The BY(0) feature will allow you to sort using a compound cross-reference on the new INDEX file (a compound cross-reference is one that indexes more than one data field). This feature will let you use any index that has no more than 7 data valued subscripts.

### **Sorting Using One or More Subscript Levels**

Each intervening subscript level between the static part of the open global root in BY(0) and the record number subscript level serves as one sort level, starting with the highest subscript level.

In example 3 above, the records would sort by the value of the free-text field stored in the first dynamic subscript, and within that by the value of the numeric field stored in the second dynamic subscript.

### **Additional Sorting with BY, FR, and TO**

When using BY(0), you can still sort in the usual way (setting BY, FR, and TO) to further sort and limit the range within the list provided by BY(0). Note that if you set BY(0), BY cannot contain the name of a SORT template. If your sort is complicated, see the documentation below on "Storing BY(0) specifications in SORT Templates."

VA FileMan selects only the list of records specified by BY(0) and its associated variables. FileMan accepts as-is the sort sequence created by any dynamic subscripts in the global specified in BY(0). Then within that sort sequence, it further sorts the records by the information provided in the BY, FR, and TO variables.

You can only sort by up to 7 sort levels in EN1^DIP, so the number of subscripts you sort by using BY(0) combined with the number of fields you sort by using BY must not total more than 7.

If BY(0) has been defined without BY, FR, and TO, the user will *not* be prompted for the SORT BY or FROM/TO ranges.

### Storing BY(0) Specifications in SORT Templates

You can store the BY(0) information in a SORT template, in order to design more complicated sorts. This allows you to sort using the global described in the BY(0) variable, and within those subscripts, to sort by additional fields and to save the entire sort description into a template. You need programmer access to do this.

In FileMan's sort dialog (with programmer access), at the SORT BY: prompt, you can enter the characters BY(0) as shown in the example immediately below. When you enter BY(0), you are then prompted for the BY(0), L(0) and all related values, exactly the same as if they were entered as input variables to the EN1^DIP call.

```
Select OPTION: 2 PRINT FILE ENTRIES

OUTPUT FROM WHAT FILE: ZZTAMI TEST//
SORT BY: NAME// BY(0)

BY(0): // ^DIZ(662001,"H",
L(0): //2

Edit ranges or subheaders? NO// YES

SUBSCRIPT LEVEL: 1// 1
FR(0,n): // 2690101
TO(0,n): // 2701231
DISPAR(0,n) PIECE ONE: //
DISPAR(0,n) PIECE TWO: // ;"Date of Birth: "
DISPAR(0,n,OUT): // S Y=$$FMTE^XLFD(Y,1)

Edit ranges or subheaders? NO//

BY(0)=^DIZ(662001,"H", L(0)=2

SUB: 1 FR(0,1): 2690101
 TO(0,1): 2701231
 DISPAR(0,1) PIECE ONE:
 DISPAR(0,1) PIECE TWO: ;"Date of Birth: "
 DISPAR(0,1,OUT): S Y=$$FMTE^XLFD(Y,1)

OK? YES//
Enter additional sort fields? NO// YES
```

```

WITHIN BY(0), SORT BY: NAME
START WITH NAME: FIRST//
 WITHIN NAME, SORT BY:

```

```

STORE IN 'SORT' TEMPLATE: ZZTAMIBY0

```

When you enter BY(0), you are prompted for BY(0) and L(0). In addition, you're asked if you want to edit ranges or subheaders. This lets you enter the FR(0,n), TO(0,n), DISPAR(0,n) and DISPAR(0,n,"OUT") values for various subscript levels. This lets you specify all the aspects of sorting using BY(0). You can store this criteria in a SORT template. If you answer YES to "Enter additional sort fields?", you will be allowed to enter additional sort fields, exactly the same as you would when creating a SORT template without the BY(0) features.

The functionality of BY(0) interactively or in a SORT template is identical to its functionality in the EN1^DIP programmer call.

An error results if, in a call to EN1^DIP, you sort by a SORT template that contains BY(0) sort criteria, and also use BY(0) as an input variable.

NOTE: The sort ranges associated with subscripts in the BY(0) global or index can be set dynamically by setting the FR(0,n) and TO(0,n) input variables. These input variables will override any sort ranges set in the template.

The "SUBSCRIPT LEVEL" prompt refers to the position of the data value in the global or index. Thus, entering a value for FR(0,n) when the SUBSCRIPT LEVEL is 1, sets the "from" value for the first data valued subscript.

Use the documentation for the BY(0) and related input variables for additional help. Also be sure to use online ? and ?? help.

The following is an example of how to call EN1^DIP when the BY(0) information is contained in a template:

```

S DIC="^DIZ(16600," ,L=0,BY="[ZZTEST]" ,FR(0,1)=
 70001,FLDS=.01
D EN1^DIP

```

**BY(0) "Don'ts"**

You should not use BY(0) if you are merely setting it to the global location of an existing regular cross-reference. You will not gain any speed, because FileMan's built-in sort optimizer already knows to sort on regular cross-references.

Also, don't specify a field's regular cross-reference as the global reference in BY(0) to sort on, and then sort on the same field using BY, FR, and TO. This actually increases the amount of work FileMan needs to do!

**"On the Fly"  
Globals  
Whose Static  
Global  
Reference  
Ends with "B"**

If you build your own list of sorted records on the fly in a temporary global (as opposed to setting BY(0) to a VA FileMan-maintained cross-reference) it's best not to let the final subscript of your static global reference be "B".

This will avoid problems that might be caused by VA FileMan's special handling of the "B" index for mnemonic cross-references.

## **^DIPT: Print/Display**

The PRINT template file contains a computed field labeled PRINT FIELDS which displays a PRINT template exactly as it was entered. Use this entry point to make this display immediately available to a user.

### **Input Variable**

**D0** (Required) Set D0 equal to the internal number of the template in the PRINT template file. For example, to display the PRINT template whose record number is 70:

```
S D0=70 D ^DIPT
```

**NOTE:** Use the number 0 (zero) not the letter O in this variable name.

## **DIBT^DIPT: Sort/Display**

The SORT template file contains a computed field labeled SORT FIELDS which displays a SORT template exactly as it was entered. Use this entry point to make this display immediately available to a user.

### **Input Variable**

**D0** (Required) Set D0 equal to the internal number of the template in the SORT template file. For example, to display the SORT template whose record number is 70:

```
S D0=70 D DIBT^DIPT
```

**NOTE:** Use the number 0 (zero) not the letter O in this variable.

## **^DIPZ: Compile**

PRINT templates can be compiled into M routines just as INPUT templates can be. The purpose of this DIPZ code generation is simply to improve overall system throughput.

Only regular PRINT templates can be compiled. You cannot compile FILEGRAM, EXTRACT, Selected Fields for Export, or EXPORT templates that are also stored in the PRINT template file.

Call the ^DIPZ routine and specify the maximum routine size, the name of the PRINT template to be used, the name of the M routine to be created, and the margin width to be used for output (typically 80 or 132). If you specify the routine name XXX and if more code is generated than can fit into that one routine, overflow routines (XXX1, XXX2, etc.) will be created. Routine XXX may call XXX1, XXX2, etc.

Once DIPZ has been used to create a hard-coded output routine, that routine is usually invoked automatically by VA FileMan within the Print File Entries and Search File Entries options and when called at EN1^DIP whenever the corresponding PRINT template is used. The compiled routines are not used if a user-specified output margin width is less than the compiled margin. Also, if the template is used with ranked sorting (i.e., the ! sort qualifier is used), the compiled version is not used.

As with compiled INPUT templates, as soon as data dictionary definitions of fields used in the PRINT template are changed, the hard-core routine(s) is/(are) compiled immediately.

### **Invoking Compiled PRINT Templates**

A DIPZ-compiled M routine may be called by any program that passes to it the variables DT, DUZ, IOSL (screen length), U (^), and D0 (the entry number to be displayed). Additionally, the variable DXS must be killed before calling the routine and after returning from it. The compiled routine writes out its report for that single entry. However, routines compiled from templates that include statistical totals cannot be called in this way.

## **EN^DIPZ: Print/Compile**

PRINT templates can be compiled into M routines just as INPUT templates can be. The purpose of this DIPZ code generation is simply to improve overall system throughput.

Only regular PRINT templates can be compiled. You cannot compile Filegram, Extract, Selected Fields for Export, or EXPORT templates that are also stored in the PRINT template file.

This entry point recompiles a PRINT template without user intervention by setting the input variables:

### **Input Variables**

- X**            The routine name.
- Y**            The internal number of the template to be compiled.
- DMAX**        The maximum size the compiled routines should reach. Consider using the \$\$ROUSIZE^DILF function to set this variable.

## **D^DIQ: Display**

This entry point takes an internal date in the variable Y and converts it to its external form. This call is very similar to DD^%DT.

### **Input Variable**

**Y** (Required) Contains the internal date to be converted. If this has five or six decimal places, seconds are automatically returned.

### **Output Variable**

**Y** External form of the date or date/time value, e.g., JAN 01, 1998.

## **DT^DIQ: Display**

This call converts the date in Y exactly like D^DIQ. Unlike D^DIQ, however, it also writes the date after it has been converted.

### **Input Variable**

Y            (Required) Contains the internal date to be converted. If this has five or six decimal places, seconds are automatically returned.

### **Output Variable**

Y            External form of the date or date/time value, e.g., JAN 01, 1998.

## EN^DIQ: Display

This entry point displays a range of data elements in captioned format, to the current device. The output from this call is very similar to that of the Inquiry to File Entries option (described in the Inquire Option section of the *VA FileMan Getting Started Manual*).

### Input Variables

**DIC** (Required) The global root of the file in the form ^GLOBAL( or ^GLOBAL(#,

If you are displaying an entry in a subfile, set DIC to the full global root leading to the subfile entry, including all intervening subscripts and the terminating comma, up to but not including the ien of the subfile entry to display.

**DA** (Required) If you are displaying an entry at the **top level** of a file, set DA to the internal entry number of the file entry to display.

If you are editing an entry in a **subfile**, set up DA as an array, where DA=entry number in the subfile to display, DA(1) is the entry number at the next higher file level,...DA(n) is the entry number at the file's top level.

**DR** (Optional) Names the global subscript or subscripts which are to be displayed by DIQ. If DR contains a colon (:), the range of subscripts is understood to be specified by what precedes and follows the colon. Otherwise, DR is understood to be the literal name of the subscript. All data fields stored within, and descendent from, the subscript(s) will be displayed, even those which normally have Read access security protection.

If DR is not defined, all fields are displayed.

**DIQ(0)** (Optional) You can include the following flags in this variable to change the display of the entry:

**A** To display **A**udit records for the entry.

**C** To display **C**omputed fields.

**R** To display the entry's **R**ecord number (IEN).

## Y^DIQ: Display

This entry point converts the internal form of any data element to its external form. It works for all FileMan data types, uses output transforms, and follows pointer trails to their final resolution. The equivalent Database Server call is \$SEXTERNAL^DILFD.

### Input Variables

#### **Naked Global Reference**

The naked global reference must be at the zero node of the data dictionary definition which describes the data [i.e., it must be at ^DD(File#,Field#,0)].

See the description of input variable C below for an example of setting the naked reference.

#### **C**

Set C to the second piece of the zero node of the data dictionary which defines that element. Typically, the programmer would:

```
S C=$P(^DD(file#,field#,0),U,2)
```

and then:

```
D Y^DIQ
```

This set will correctly set the naked global reference as described above.

#### **Y**

Set Y to the internal form of the value being converted. This is the data that you want to convert to external form.

### Output Variable

#### **Y**

The external form of the value. Basically, Y is changed from internal to external.

## EN^DIQ1: Data Retrieval

This entry point retrieves data from a file for a particular entry.

**NOTE:** The equivalent Database Server calls are GETS^DIQ and \$\$GET1^DIQ.

It is your responsibility to kill the output array, ^UTILITY("DIQ1",\$J), before and after using this call.

### Input Variables

**DIC**            The file number or global root.

**DR**             A string specifying the data fields to retrieve for the given entry.  
The DR-string may contain:

A single number corresponding to the internal number of a field in the file.

A range of field numbers, in the form M:N, where M is the first and N the last number of the inclusive range. All fields whose numbers lie within this range will be retrieved.

A combination of the above, separated by semicolons. If field numbers .01, 1, 2, 4, 10, 11, 12, 13, 14, 15, and 101 exist for a file, and you want to retrieve the data in these fields, simply write:

```
S DR=".01;1;4;10;11;12;13;14;15;101"
```

**DR(subfile  
\_number)**      If you want to retrieve values from fields from a subentry in a multiple field, include the top-level field number for the multiple in DR. Then, include the multiple's subfield numbers whose values you want to retrieve in a node in DR, subscripted by the subfile number.

See also DA(subfile\_number) below for how to specify which subfile entry to retrieve.

For example, if you want to retrieve data from subfields .01 and 7 for subentry 1 from field 4 which defines the multiple 16000.02, then you write:

```
S DIC=16000,DR="4",DA=777
S DR(16000.02)=".01:7",DA(16000.02)=1
D EN^DIQ1
```

- DA** The internal number of the entry from which data is to be extracted.
- DA(subfile\_number)** If you want to retrieve values from fields from a subentry in a multiple, set DA to the top-level entry number. Then, include the subentry number in a node in DA, subscripted by the subfile number. See DR(subfile\_number) above for how to specify which fields in the subfile entry to retrieve.

You can descend one or more subfile levels; however, you can only retrieve values for one subentry at any given subfile level. The full path from the top level of the file to the lowest-level subfile entry must be fully specified in nodes in DR and DA.

- DIQ** (Optional) The local array name into which the field values will be placed. ^UTILITY("DIQ1", \$J, will be used if DIQ is not present. This array name should not begin with DI.

- DIQ(0)** (Optional) This variable is used to control which is returned: internal values, external values, or both. DIQ(0) also indicates when null values are not returned. The DIQ(0) string can contain the values that follow:

- I** return **I**nternal values
- E** return **E**xternal values
- N** do not return **N**ull values

## Output

The format and location of the output from EN^DIQ1 depends on the status of input variables DIQ and DIQ(0) and on whether or not a word processing field is involved.

### DIQ and DIQ(0) undefined

Output into:

```
^UTILITY("DIQ1", $J, file#, DA, field#)=external value
```

This is for backward compatibility. Each field requested will be defined in the utility global but the value may be null. The only exception to this would be when DA held the number of an entry which does not exist. In that case, nothing is returned. The

values returned are the external values. Printable values—pointers, sets of codes, etc.—are resolved; dates are in external format.

### **DIQ(0) defined, DIQ undefined**

Output into:

```
^UTILITY("DIQ1", $J, file#, DA, field#, "E")=external value
^UTILITY("DIQ1", $J, file#, DA, field#, "I")=internal value
```

If DIQ(0) contains "E", the external value is returned with a final global subscript of "E".

If DIQ(0) contains "I", the internally stored value is returned with a final global subscript of "I". The internal value is the value stored in the file, for example, the record number of the entry in the pointed-to file, not the resolved value of the pointer. Since computed fields store no data, no nodes are returned for computed fields.

If DIQ(0) contains "N", no nodes are set for either internal or external values if the field is null.

If DIQ(0) contains both "I" and "E", generally two nodes are returned for each field: one with the internal value, one with the external value. However, no nodes are produced for the internal value if the field is computed and no nodes are produced at all for null-valued fields if DIC(0) contains "N". Nodes are subscripted as described above.

### **DIQ defined**

The output is similar except that the data is stored in the specified local array. So if DIQ(0) is not defined, then the output is:

```
@(DIQ(file#, DA, field#))=external value
```

If DIQ(0) is defined, then the output is:

```
@DIQ(file#, DA, field#, "E")=external value
@DIQ(file#, DA, field#, "I")=internal value
```

### **Word Processing Field**

Output from a word processing field will only be an external value. The status of DIQ(0) has no effect. If DIQ is not defined, it goes into the global nodes that follow:

```
^UTILITY("DIQ1", $J, file#, DA, field#, 1)
^UTILITY("DIQ1", $J, file#, DA, field#, 2)
.
.
.
```

If DIQ is defined, it goes into:

```
@DIQ(file#,DA,field#,1)=External Value 1
@DIQ(file#,DA,field#,2)=External Value 2
@DIQ(file#,DA,field#,3)=External Value 3
@DIQ(file#,DA,field#,4)=External Value 4
.
.
.
```

**^DIR: Reader**

DIR is a general purpose response reader that can be used to issue a prompt, read input interactively, perform syntax checking on the input, issue error messages or help text, and return input in a processed form. Its use is recommended to standardize user dialog and to eliminate repetitive coding.

DIR is reentrant: A DIR call may be made from within a DIR call. To reenter DIR, use the NEW command to save the DIR array (NEW DIR) before setting input variables and making the second call.

- A. Input and Output Variables (Summary)**
- B. Required Input Variables (Full Listing)**
- C. Optional Input Variables (Full Listing)**
- D. Output Variables (Full Listing)**
- E. Examples**

**A. Input and Output Variables (Summary)****Input Variables-Required**

|               |                                                                       |                  |
|---------------|-----------------------------------------------------------------------|------------------|
| <b>DIR(0)</b> | Required: First character of Piece-1 (first 3 characters for DD-type) | Read type        |
|               | Optional: Subsequent characters of Piece-1                            | Input modifiers  |
|               | Optional: Piece-2                                                     | Input parameters |
|               | Optional: Piece-3                                                     | INPUT transform  |

**Input Variables-Optional**

|                   |                                                                           |
|-------------------|---------------------------------------------------------------------------|
| <b>DA</b>         | For DD-type reads, can specify entry from which to retrieve default value |
| <b>DIR("A")</b>   | Programmer-supplied prompt to override default                            |
| <b>DIR("A",#)</b> | Array for information to be displayed before the prompt                   |
| <b>DIR("B")</b>   | Default response                                                          |
| <b>DIR("L")</b>   | For set-of-code fields: programmer-specified format to display            |

|                   |                                                            |
|-------------------|------------------------------------------------------------|
| <b>DIR("L",#)</b> | codes.                                                     |
| <b>DIR("S")</b>   | Screen for pointer, set-of-code, and list/range reads      |
| <b>DIR("T")</b>   | Time specification to be used instead of DTIME             |
| <b>DIR("?")</b>   | Help displayed when the user enters a single question mark |
| <b>DIR("?",#)</b> |                                                            |
| <b>DIR("??")</b>  | Help displayed when the user enters a double question mark |

### **Output Variables-Always Returned**

|          |                           |
|----------|---------------------------|
| <b>X</b> | Unprocessed user response |
| <b>Y</b> | Processed user response   |

### **Output Variables-Conditionally Returned**

|               |                                                                                     |
|---------------|-------------------------------------------------------------------------------------|
| <b>Y(0)</b>   | External form of response for set, pointer, list, and date                          |
| <b>DTOUT</b>  | Defined if the user times out                                                       |
| <b>DUOUT</b>  | Defined if the user entered an up-arrow                                             |
| <b>DIRUT</b>  | Defined if the user entered an up-arrow, pressed the Enter/Return key, or timed out |
| <b>DIROUT</b> | Defined if the user enters two up-arrows                                            |

## **B. Required Input Variables (Full Listing)**

**DIR(0)** DIR(0) is the only required input variable. It is a three piece variable. The first character of the first piece must be defined (or first 3 characters for DD-type). Additional characters of the first piece and the second two pieces are all optional.

The first character of the first up-arrow piece indicates the type of the input to be read. The second piece describes parameters, delimited by colons, to be applied to the input. Examples are maximum length for free text data or decimal digits for numeric data. The third piece is executable M code that acts on the input in the same manner as an INPUT transform. The acceptable types are shown below:

**DIR(0) (Summary)**

| <b>DIR(0)<br/>Read<br/>Type</b> | <b>Piece-1<br/><br/>First<br/>Charac-<br/>ter (re-<br/>quired)</b> | <b>Subsequent<br/>Characters<br/>(optional)</b> | <b>Piece-2<br/><br/>Format</b>              | <b>Piece-3<br/><br/>Executable<br/>M code<br/>(optional)</b> |
|---------------------------------|--------------------------------------------------------------------|-------------------------------------------------|---------------------------------------------|--------------------------------------------------------------|
| <b>Date</b>                     | D                                                                  | A,O                                             | Minimum<br>date:-<br>Maximum<br>date:%DT    | code                                                         |
| <b>End-of-<br/>Page</b>         | E                                                                  | A                                               | --                                          | --                                                           |
| <b>Free-<br/>text</b>           | F                                                                  | A,O,U,r                                         | Minimum<br>length:<br>Maximum<br>length     | code                                                         |
| <b>List or<br/>range</b>        | L                                                                  | A,O,C                                           | Minimum:<br>Maximum:<br>Maximum<br>decimals | code                                                         |
| <b>Numeric</b>                  | N                                                                  | A,O                                             | Minimum:<br>Maximum:<br>Maximum<br>decimals | code                                                         |
| <b>Pointer</b>                  | P                                                                  | A,O,r                                           | Global<br>Root or<br>#:DIC(0)               | code                                                         |
| <b>Set of<br/>Codes</b>         | S                                                                  | A,O,X,B                                         | Code:<br>Stands<br>for;Code:<br>stands for; | code                                                         |
| <b>Yes/No</b>                   | Y                                                                  | A,O                                             | --                                          | code                                                         |
| <b>DD</b>                       | #, #                                                               | A,O,r                                           | --                                          | code                                                         |

## **DIR(0) (Detailed Explanation)**

### **Piece-1 of DIR(0) (Subsequent Characters are Optional):**

The first up-arrow piece of DIR(0) can contain other parameters that help to specify the nature of the input or modify the behavior of the reader. These characters must appear after the character indicating type (or after the field number if it is a DD type). They are described below and examples are provided later in this section):

- A**        Indicates that nothing should be **A**ppended to the programmer-supplied prompt DIR("A"), which is described below. If there is no DIR("A"), then no prompt is issued.
- B**        Only applies to a set of codes; indicates that the possible choices are to be listed horizontally after the prompt.
- C**        Only applies to list reads. The values returned in Y and the Y() array are **C**ompressed. They are not expanded to include each individual number, rather, ranges of values are returned using the hyphen syntax. This is similar to the format in which the user can enter a range of numbers.

This flag is particularly useful when a user may select many numbers, e.g., when decimals are involved. The call is much faster and the possibility of the local symbol table filling up with nodes in the Y() array is eliminated.

- O**        Indicates that a response is **O**ptional. If this is not included, then a null response is not allowed. For DD type reads, the O is automatically included if the field in question is not a required field.
- r**        If user does not choose to accept the default, they must type in their entire response. They will not get the "Replace-With" prompt, no matter how long the default response is.
- U**        Only applies to free text reads. It allows the user response to contain ^ (**U**p-arrow). A leading up-arrow aborts the read and sets DUOUT and DIRUT whether or not U is in DIR(0). However, U allows ^s to be embedded in the user response.
- X**        Only applies to set of codes. Indicates a request for an e**X**act match. No lower- to uppercase conversion is to be done.

**Piece-2 of DIR(0) (Optional)**

Qualifying limits on user response are as described in summary table above.

**Piece-3 of DIR(0) (Optional)**

The third piece of DIR(0) is executable M code that acts like the INPUT transform of a field in a data dictionary. The value that was entered by the user is contained in the variable X. The code can examine X and, if it is not appropriate, should KILL X. If X is undefined after the execution of the third piece of DIR(0), the reader knows that the input was unacceptable, issues a help message, and re-asks for input. It is unnecessary to put checks for minimum and maximum or length in the third piece. These should be specified in the second piece of DIR(0). An example of DIR(0) with all three pieces is:

```
S DIR(0)="F^3:30^K:X'? .U X"
```

which says that if the input is not all uppercase, then the data is unacceptable. The check for a length from 3 to 30 characters takes place automatically because of the second piece. The third piece is not executed if the specifications in the second piece are not met. If the user combines the DD data type with a third piece in DIR(0), for example:

```
S DIR(0)="19,.01^^K:X'?1"DI" X"
```

then the third piece of DIR(0) is not executed until after the INPUT transform has been executed and X was not Killed by the transform.

**C. Optional Input Variables (Full Listing)**

**DA** (Optional) For DD-type reads only, if DIR("B") is not set, you may retrieve a value from the database to display as a default. Identify the entry from which the value should come by setting the DA variable to its record number. If a subfile is involved, set up a DA() array where DA equals the record number for the lowest level subfile, DA(1) for the next higher, and so on.

**NOTE:** Although you can retrieve defaults from the database by using DA, the values in the database are not changed by ^DIR calls.

**DIR("A")** (Optional) The reader provides a generic default prompt for each type, e.g., enter a number or enter response. To issue a

more meaningful prompt, DIR("A") can be set to a character string that more clearly indicates the nature of the data being requested. For example, setting the following:

```
S DIR("A")="PRICE PER DISPENSE UNIT: "
S DIR(0)="NA^0:5:2"
```

causes the prompt to appear as:

```
PRICE PER DISPENSE UNIT:
```

**DIR("A",#)**

(Optional) If you want to issue a longer message before actually reading the input, you can set the DIR("A",#) array in addition to DIR("A"). The #'s must be numeric. After the array has been displayed, DIR("A") is issued as the prompt for the read. It is necessary for DIR("A") to be set if the programmer is to use this array. For example, setting the following:

```
S DIR("A")="PRICE PER DISPENSE UNIT:"
S DIR("A",1)="Enter price data with two decimal points."
S DIR("A",2)="Cost calculations require this precision."
```

causes the following dialog to appear to the user:

```
Enter price data with two decimal points.
Cost calculations require this precision.
PRICE PER DISPENSE UNIT:
```

**DIR("B")**

(Optional) Set this variable to the default response for the prompt issued. It appears after the prompt and before the // (double slashes). If the user simply presses the Enter/Return key, the default response is accepted by the reader.

**DIR("L")  
DIR("L",#)**

(Optional) Only applies to set-of-codes fields. Lets you replace the standard vertical listing of codes that the Reader displays with your own listing. It is up to you to ensure that the contents of the DIR("L") array match the codes in the second ^-piece of DIR(0).

The format of the DIR("L") array is similar to DIR("A") and DIR("?"). The #'s must be numeric starting from 1. The numeric subscripted array nodes are written first and the DIR("L") node is written last. For example, if you code:

```
S DIR(0)="SO^1:ONE;2:TWO;3:THREE;4:FOUR;5:FIVE"
S DIR("L",1)="Select one of the following:"
S DIR("L",2)=""
S DIR("L",3)=" 1 ONE 4 FOUR"
S DIR("L",4)=" 2 TWO 5 FIVE"
S DIR("L")=" 3 THREE"
D ^DIR
```

the user sees the following:

```
Select one of the following:
```

```
1 ONE 4 FOUR
2 TWO 5 FIVE
3 THREE
```

```
Enter response:
```

## **DIR("S")**

(Optional) Use the DIR("S") variable to screen the allowable responses for pointer, set of codes, and list/range reads. This variable works as the DIC("S") variable does for ^DIC calls. Set DIR("S") equal to M code containing an IF statement. After execution, if \$T is set to 1, the user response is accepted; if set to 0, it is not.

For pointer reads, when DIR("S") is executed, the M naked indicator is equal to the 0 node of the entry being screened. The variable Y equals its record number.

For set of codes reads, when the DIR("S") is executed, Y equals the internal code.

For list/range reads, if you also use the C flag in piece 1 of DIR(0), your output is still compressed. Internally during the call, however, the range must be uncompressed so that each number in the range can be screened. So using DIR("S") with the C flag during list/range reads loses the C flag's advantages in speed (but the C flag's advantage in avoiding storage overflows remains).

## **DIR("T")**

(Optional) Time-out value to be used in place of DTIME. Value is represented in seconds.

## **DIR("?")**

(Optional) This variable contains a simple help prompt, which is displayed to the user when one question mark is entered. It usually takes the place of the reader's default prompt. For example, if you code:

```
S DIR(0)="F^3:10"
S DIR("?")="Enter from three to ten characters"
S DIR("A")="NICKNAME"
D ^DIR
```

the user sees the following:

```
NICKNAME: ?
```

```
Enter from three to ten characters.
```

**NOTE:** When displayed, a period (.) is added to the DIR("?") string. Periods are not appended when displaying the DIR("?",#) array, however.

When one question mark is entered in DD reads, the data dictionary's help prompt is shown before DIR("?"). For pointer reads, a list of choices from the pointed-to file is shown in addition to DIR("?").

As an alternative, you can set DIR("?") to an up-arrow followed by M code, which is executed when the user enters one question mark. An example might be:

```
S DIR(" ? ") = "^D HELP^%DTC"
```

Execution of this M code overrides the reader's default prompt. If DIR("?") is defined in this way (a non-null second piece), the DIR("?",#) array is not displayed.

**DIR("?",#)** (Optional) This array allows the user to display more than one line of help when the user types a single question mark. The first up-arrow piece of DIR("?") must be set for the array to be used. The second up-arrow piece of DIR("?") must be null, otherwise the DIR("?",#) array is ignored. The #'s must be numeric starting from 1. The numbered lines are written first, that is, first DIR("?",1), then DIR("?",2), etc. The last help line written is DIR("?",#). These lines are the only ones written, which means that the reader's default prompt is not issued.

**DIR("??")** (Optional) This variable, if defined, is a two-part variable. The first up-arrow piece may contain the name of a help frame. The help processor displays this help frame if the user enters two question marks.

The second part of this variable (after the first up-arrow piece) may contain M code that is executed after the help frame is displayed.

For example:

```
S DIR("??") = "DIHELPXX^D EN^XXX"
```

**NOTE:** In order to use this variable, you must have Kernel's help processor on your system.

**D. Output Variables (Full Listing)**

**X** This is the unprocessed response entered by the user. It is always returned. If the user accepts the default in DIR("B"), it is the default. If the user up-arrows out or just presses the Enter/Return key on an optional input, X is the up-arrow or null.

**Y** Y is always defined as the processed output. The values returned are:

| <b>Type</b>          | <b>Y Returned as</b>                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                     |
|----------------------|------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| <b>Date</b>          | The date/time in VA FileMan format.                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                      |
| <b>End-of-page</b>   | Y=1 for continue (user pressed the Enter/Return key).<br>Y=0 for exit (the user pressed up-arrow).<br>Y="" for time out (the user timed out).                                                                                                                                                                                                                                                                                                                                                                                            |
| <b>Free-text</b>     | The data typed in by the user. In this case, it is the same as X.                                                                                                                                                                                                                                                                                                                                                                                                                                                                        |
| <b>List or range</b> | The list of numeric values, delimited by commas and ending with a comma.<br><br>If the C flag was not included in the first piece of DIR(0), an expanded list of numbers, including each individual number in a range, is returned. If the C flag was included, a compressed list that uses the hyphen syntax to indicate a range of numbers is returned.<br><br>Any leading zeros or trailing zeros following the decimal point are removed; i.e., only canonic numbers are returned. If the list of returned numbers has more than 245 |

characters, integer-subscripted elements of Y [Y(1), Y(2), etc.] contain the additional numbers. Y(0) is always returned equal to Y.

|                     |                                                                                                                                                                                                                                        |
|---------------------|----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| <b>Numeric</b>      | The canonic value of the number entered by the user; i.e., leading zeros are deleted and trailing zeros after the decimal are deleted.                                                                                                 |
| <b>Pointer</b>      | The normal value of Y from a DIC lookup, that is, Internal Entry Number^Entry Name. If the lookup was unsuccessful, Y=-1.                                                                                                              |
| <b>Set of Codes</b> | The internal value of the response.                                                                                                                                                                                                    |
| <b>Yes/No</b>       | Y=1 for yes.<br>Y=0 for no                                                                                                                                                                                                             |
| <b>DD (#,#)</b>     | The first ^-piece of Y contains the result of the variable X after it has been passed through the INPUT transform of the field specified. Depending on the data type involved, subsequent ^-pieces may contain additional information. |

The following list summarizes the values of Y upon timeout, up-arrows, or pressing the Enter/Return keys for all reads. Exceptions are noted.

| <b>Condition</b>             | <b>Value of Y</b>       | <b>Comments</b>                        |
|------------------------------|-------------------------|----------------------------------------|
| <b>Timeout</b>               | Y=""                    | --                                     |
| <b>Up-arrow (^)</b>          | Y=^                     | in all cases except end-of-page reads. |
| <b>Y=0</b>                   | upon end-of-page reads. | --                                     |
| <b>Double Up-arrow (^ ^)</b> | Y=^^                    | --                                     |

|               |           |                                                                  |
|---------------|-----------|------------------------------------------------------------------|
| <b>Return</b> | Y=""      | for optional reads (reads allowing a null response).             |
|               | Y=-1      | for pointer reads.                                               |
|               | Y=0       | for YES/NO type when NO is default.                              |
|               | Y=1       | for YES/NO type when YES is default.                             |
|               | Y=1       | for end-of-page reads.                                           |
|               | Y=default | when a default is provided other than for YES/NO type questions. |

**Y(0)** This is defined for the set of codes, list, pointer, date, and Yes/No reads. It is also returned for DD reads when the field has a set of codes, pointer, variable pointer, or date data type. It holds the external value of the response for set of codes or Yes/No, the zero node of the entry selected for a pointer, and the external date for a date and variable pointer. To have Y(0) returned for pointer-types, the DIC(0) string in the second piece of DIR(0) must contain a Z, for example:

```
DIR(0)="P^19:EMZ"
```

For list reads, it contains the same values as the Y variable. There may be additional nodes in the Y() array depending on the size of the list selected by the user.

**DTOUT** If the read has timed-out, then DTOUT is defined.

**DUOUT** If the user entered a leading up-arrow, DUOUT is defined.

**DIRUT** If the user enters a leading up-arrow, times out, or enters a null response, DIRUT is defined. A null response results from pressing the Enter/Return key at a prompt with no default or

entering the at-sign (@), signifying deletion. If, however, the user presses the Enter/Return key in response to an end of page read, DIRUT is not defined. If DIRUT is defined, the user can enter the following common check to quit after a reader call:

```
Q:$D(DIRUT)
```

**DIROUT** If the user entered two up-arrows, DIROUT is defined.

## E. Examples

1. Date
2. End-of-Page
3. Free Text
4. List or Range
5. Numeric
6. Pointer
7. Set
8. Yes/No
9. DD

### 1. Date Example

```
S DIR(0)="D^2880101:2880331:EX"
```

This tells the reader that the input must be an acceptable date. To determine that, ^%DT is invoked with the %DT variable equal to EX. If the date is a legitimate date, then it is checked to see if the date falls between January 1, 1988 and March 31, 1988. In general, both minimum and maximum are optional. If they are there, they must be in VA FileMan format. The only exceptions are that NOW and DT may be used to reference the current date/time. Remember that NOW contains a time stamp. If it is used as a minimum or maximum value, an R or T should be put into the %DT variable. If DIR(0) is set up to expect a time in the response, you can help the user by including that requirement in the prompt. Otherwise, a response without a time stamp (such as TODAY) might unexpectedly fail.

### 2. End-of-Page Example

```
S DIR(0)="E"
```

There are no parameters. Enter/Return and up-arrow are the only acceptable responses. This DIR(0) setting causes the following prompt to be issued:

```
Press the return key to continue or '^' to exit:
```

### 3. Free-Text Example

```
S DIR(0)="F^3:30"
```

This tells the reader that the input must be alphanumeric or punctuation, (control characters are not allowed) and that the length of input must be no fewer than 3 and no more than 30 characters. The maximum acceptable length for a free-text field is 245 characters.

**NOTE:** A leading up-arrow always aborts the read and sets DIRUT or DUOUT.

#### With DIR(0) containing U

```
S DIR(0)="FU^3:30"
```

The user can enter any response that is from 3 to 30 characters long. The response can contain embedded up-arrows. Without U, an embedded up-arrow causes the user to receive an error message.

#### With DIR(0) containing A

```
S DIR(0)="FA^2:5",DIR("A")="INITIAL"
```

The prompt is set only to the word INITIAL. If the A were not included, a colon and space would be appended to the prompt and it would look like this:

```
INITIAL:
```

### 4. List or Range Example

```
S DIR(0)="L^1:25"
```

This tells the reader that the input may be any set of numbers between 1 and 25. The numbers may be separated by commas, dashes, or a combination of both. Two acceptable responses to the example above are:

```
1,2,20
4-8,16,22-25
```

Remember that this is a numeric range or list. It can only contain positive integers and zero (no negative numbers).

#### With DIR(0) containing C

```
>S DIR(0)="LC^1:100:2" D ^DIR
```

```
Enter a list or range of numbers (1-100): 5,8.01,9-40,
7.03,45.9,80-100
```

```
>ZW Y
Y=5,7.03,8.01,9-40,45.9,80-100,
Y(0)=5,7.03,8.01,9-40,45.9,80-100,
```

Here the user can enter numbers from 1 to 100 with up to two decimal places. The C flag tells the reader not to return each individual number in Y. Instead, inclusive ranges of numbers are returned. In this case, without the C flag, 137 subscripted nodes of the Y() array would be returned; the call would be very slow and might cause an error if the size of the Y() array exceeded local storage.

## 5. Numeric Example

```
S DIR(0)="N^20:30:3"
```

This tells the reader that the input must be a number between 20 and 30 with no more than three decimal digits.

**NOTE:** If no maximum is specified in the second ^-piece, the default maximum is 999999999999.

### With DIR(0) containing O

```
S DIR(0)="NO^0:120",DIR("A")="AGE"
```

This allows the user to press the Enter/Return key without entering any response and leave the reader. Without the O, the following messages appear:

```
This is a required response. Enter '^' to exit.
```

## 6. Pointer Example

```
S DIR(0)="P^19:EMZ"
```

This tells the reader to do a lookup on File 19, setting DIC(0)="EMZ" before making the call.

If the user enters a response that causes the lookup to fail, the user is prompted again for a lookup value.

A pointer read can be used to look up in a subfile. In that case, the global root must be used in place of the file number. For example, to look up in the menu subfile (stored descendent from subscript 10) for entry #2 in File 19:

```
S DIR(0)="P^DIC(19,2,10,:QEM"
```

Remember to set any necessary variables, e.g., DA(1).

## 7. Set Example

```
S DIR(0)="S^1:MARRIED;2:SINGLE"
```

This tells the reader to only accept one of the two members of the set. The response may be 1, 2, MARRIED, or SINGLE. When DIR("A") is included without the A modifier on the first piece, the prompting is done as follows:

```
S DIR(0)="S^M:MALE;F:FEMALE"
S DIR("A")="SEX" D ^DIR
```

Select one of the following:

```
 M MALE
 F FEMALE
```

SEX:

### With DIR(0) containing A

```
S DIR(0)="SA^M:MALE;F:FEMALE"
S DIR("A")="SEX: " D ^DIR
```

Whereas, with the A, it would appear as follows:

SEX:

### With DIR(0) containing B

```
S DIR(0)="SB^M:MALE;F:FEMALE"
S DIR("A")="SEX" D ^DIR
```

When this is executed, instead of getting the vertical listing as shown above, the prompt would appear as:

SEX: (M/F):

### With DIR(0) containing X

```
S DIR(0)="SX^M:MALE;F:FEMALE"
S DIR("A")="SEX"
```

This would cause a lowercase M or F to be rejected. The prompting is done as follows:

Select one of the following:

```
 M Male
 F Female
```

SEX: **f** (user's response)

Enter a code from the list.

## 8. Yes/No Example

```
S DIR(0)="Y",DIR("B")="YES"
```

This tells the reader that the response can only be Yes or No. When using DIR("B") to provide a default response, spell out the entire word so that when the user presses the Enter/Return key to accept the default, echoing functions properly.

## 9. DD Example

```
S DIR(0)="19,1"
```

This format is different from the others in that the first number is a file number and the second is a field number in that file. The reader uses the data dictionary for

field 1 in file 19 and issues the label of that field as the prompt. The input is passed through the INPUT transform in the dictionary. Help messages are also the ones contained in the dictionary for this field.

Normally, DD reads based on a free text field do not allow embedded up-arrows. However, if the field specified is positioned on the data node using the Em,n format (instead of the ^-piece format), up-arrows embedded in the user's response are accepted. (See the Field Global Storage section of the Advanced File Definition chapter for an explanation of locating fields on the data node.) Initial up-arrows abort the read and set DIRUT and DUOUT.

It is not possible to use this format if the field defines a subfile, i.e., the second piece of the zero node of the field definition contains a subfile number. To use the reader for a field in a subfile, do the following:

```
S DIR(0)="Subfile#,field#"
```

It is the programmer's responsibility to set any variables necessary for the INPUT transform to execute correctly.

Always NEW or KILL DA before doing a DD-type DIR call, unless you wish to use the default feature. The default feature allows you to retrieve default values from the database for DD reads by setting DA (or the DA array for subfiles) equal to the record number containing the desired default value.

## EN^DIS: Search File Entries

You can call the Search File Entries option of VA FileMan for a given file when you want the user to be able to specify the search criteria. This is done by invoking EN^DIS. In addition to DT and DUZ, the program needs the DIC input variable.

### Input Variable

**DIC** (Required) The global root of the file in the form ^GLOBAL( or ^GLOBAL(#, or the number of the file.

If the search is allowed to run to completion, and if the search criteria have been stored in a template, then a list of the record numbers that meet the search criteria is stored in that same template.

**NOTE:** The same global array is used to store a list of record numbers saved in FileMan Inquire mode.

```
^DIBT(SORT_TEMPLATE#, 1, IEN) = " "
```

The 1 node indicates that the IEN list was created one of two ways:

1. The user was in FileMan INQUIRE mode, selected a number of records, and saved the list in a template.
2. The user ran the FileMan SEARCH, either through the interactive FileMan menu or through the programmer entry point EN^DIS. In this case, the IEN list is the group of record numbers that met the search criteria.

IEN is the internal entry number of a record in the file indicated by the fourth piece of the zero node of the template, ^DIBT(SORT\_TEMPLATE#,0).

The list of record numbers stored in the template can be used as input to the print routine, EN1^DIP, to create further reports.

## EN^DIU2: Data Dictionary Deletion

Occasionally you may need to delete a file's data dictionary and its entry in ^DIC in order to properly update a running system. Use this entry point to do it.

You usually have the option of deleting the data when you delete the data dictionary. (See the DIU(0) variable below.) However, data will always be deleted if your file is in ^DIC(File#,. Be careful using this utility when your data is in the ^DIC global.

In all cases, both DIU and DIU(0) are returned from the call. You will find that DIU is returned as the global root regardless of whether it was defined as the file number or as the global root when making the call.

**NOTE:** If the root of a file's data is an unsubscripted global [e.g., DIU="^MYDATA("], you must make sure that the systems on which you want to perform the deletion do not restrict the killing of the affected unsubscripted globals.

**REMINDER:** It is your responsibility to clean up (kill) DIU, the input variable, after any call to this routine!

### Input Variables

**DIU** (Required) The file number or global root, e.g., ^DIZ(16000.1,. This must be a subfile number when deleting a subfile's data dictionary.

**DIU(0)** Input parameter string that may contain the following:

**D** Delete the data as well as the data dictionary.

**E** Echo back information during deletion.

**S** Subfile data dictionary is to be deleted.

**T** Templates are to be deleted.

### Example

```
>S DIU="^DIZ(16000.1, ",DIU(0)="" D EN^DIU2
```

Only the data dictionary will be deleted. The data and templates remain. By including either the D or T, you can also delete the data or the templates. If the E is included, then the user will be asked whether or not the global should be deleted.

### **Subfile Deletion**

If you want to delete the dictionary for a subfile, you must include the S in DIU(0). The variable, DIU, in this case must be a subfile data dictionary number. It cannot be a global root. When deleting a subfile's dictionary, all dictionaries subordinate to that dictionary are also deleted. Data can also be deleted when deleting a subfile; this process could take some time depending on the number of entries in the whole file.

### **Example**

```
>S DIU=16000.01,DIU(0)="S" D EN^DIU2
```

## EN^DIWE: Text Editing

This routine is used to edit word processing text using VA FileMan's editors. If the user has established a Preferred Editor through Kernel, that editor is presented for use. FileMan's editors expect the text to contain only printable ASCII characters.

### Input Variables

- DIC**                    The global root of where the text is located.
- NOTE:** VA FileMan uses ^UTILITY(\$J,"W") when EN^DIWE is called. Thus, DIC should not be set equal to that global location.
- DWLW**                    (Optional) This variable indicates the maximum number of characters that will be stored on a word-processing global node. When the user enters text, the input line will not be broken to DWLW-characters until after the Enter/Return key is pressed. Thus, if DWLW=40 and the user types 90 characters before pressing the Enter/Return key, the text would be stored in three lines in the global. If this variable is not set, the default value is 245. This variable is always killed by FileMan.
- DWPK**                    (Optional) This variable determines how lines that are shorter than the maximum line length (set by DWLW) are treated by FileMan. It can be set to 1 or 2. This variable is always killed by FileMan.
- DWPK=1**                    If the user enters lines shorter than the maximum line length in variable DWLW, the lines will be stored as is; they will not be joined. If lines longer than DWLW are entered, the lines will be broken at word boundaries.
- DWPK=2**                    If the user types lines shorter than the maximum line length in variable DWLW, the lines will be joined until they get to the maximum length; the lines are "filled" to DWLW in length. If the lines are longer than DWLW, they will be broken at word boundaries. This is the default used if DWPK is not set prior to the EN^DIWE call.

- DWDISABL** This variable can be used to disable specific Line Editor commands. For example, if DWDISABL contains "P", then the Print command in the Line Editor is disabled. This variable is killed by FileMan. (Optional)
- DIWEPSE** (Optional) If this variable is defined before entering the Preferred Editor (if the Preferred Editor is not the Line Editor), the user receives the following prompt:
- ```
Press RETURN to continue or '^' to exit:
```
- Set this variable if you want to allow the user to read information on the screen before the display is cleared by a screen-oriented editor. This variable is always killed by FileMan.
- DIWESUB** (Optional) The first 30 characters of this variable are displayed within angle brackets (< and >) on the top border of the Screen Editor screen. This variable is killed by FileMan.
- DIWETXT** (Optional) The first IOM characters of this variable are displayed in high intensity on the first line of the Screen Editor screen. This variable is killed by FileMan.
- DDWLMAR** (Optional) This variable indicates the initial column position of the left margin when the Screen Editor is invoked. The user can subsequently change the location of the left margin. This variable is killed by FileMan.
- DDWRMAR** (Optional) This variable indicates the initial column position of the right margin when the Screen Editor is invoked. The user can subsequently change the location of the right margin. This variable is killed by FileMan.
- DDWRW** (Optional) This variable indicates to the Screen Editor the line in the document on which the cursor should initially rest. This variable has effect only if the user's preferred editor is the Screen Editor and applies only when the Screen Editor is first invoked. If the user switches from the Screen Editor to another editor and then back to the Screen Editor, the cursor always rests initially on line 1.
- If this variable is set to "B", the cursor will initially rest at the bottom of the document and the value of DDWC described

immediately below is ignored. The default value of DDWRW is 1. This variable is killed by FileMan.

DDWC

(Optional) This variable indicates to the Screen Editor the initial column position of the cursor. The same restrictions described above for DDWRW apply to DDWC.

If this variable is set to "E", the cursor will initially rest at the end of the line defined by DDWRW. The default value of DDWC is 1. This variable is killed by FileMan.

DDWFLAGS

Flags to control the behavior of the Screen Editor. The possible values are:

M Indicates that the Screen Editor should initially be in NO WRAP Mode when invoked.

Q Indicates that if the user attempts to Quit the editor with <PF1>Q, the confirmation message "Do you want to save changes?" is NOT asked.

R Indicates that the Screen Editor should initially be in REPLACE mode when invoked.

This variable is killed by FileMan. (Optional)

^DIWF: Form Document Print

Form Document Print Introduction (^DIWF)

The entry points in ^DIWF are designed to use the contents of a word processing field as a target document into which data can be inserted at print time. The data may come from another VA FileMan file or be provided by the user interactively at the time the document is printed. A file containing a word processing type field is first selected and then an entry from that file. The word processing text in that entry is then used as a form with which to print output from any other file.

The word processing text used will typically include windows into which data from the target file automatically gets inserted by DIWF. The window delimiter is the vertical bar (|). Thus, if a word processing document contains |NAME| somewhere within it, DIWF will try to pick the NAME field (if there is one) out of the file being printed. Any non-multiple field label or computed expression can be used within a |-window, if:

1. an expression within the |-window cannot be evaluated, and
2. the output of DIWF is being sent to a different terminal than the one used to call up the output,

then the user will be asked to type in a value for the window, for each data entry printed. Thus, the word processing text used as a target document might include the window |SALUTATION|, where SALUTATION is not a valid field name in the source file. When DIWF encounters this window, and failing to find a SALUTATION field in the source file, it will ask the user to enter SALUTATION text which then immediately gets incorporated into the output in place of that window. Notice that we are referring to two files-the document file which contains the word processing text and the print from file which DIWF will use to try to fill-in data for the windows.

Invoking DIWF at the top (i.e., D ^DIWF) results in an interactive dialog with the user.

Example

Suppose you had a file called FORM LETTER (File #16001) and data is stored in ^DIZ(16001,. This file has a word processing type field where the text of a form letter is stored. In this file, as shown below, there are several form letter entries one of which is APPOINTMENT REMINDER:

Classic VA FileMan API

```
Select Document File: FORM LETTER  
Select DOCUMENT: APPOINTMENT REMINDER  
Print from what FILE: EMPLOYEE  
WANT EACH ENTRY ON A SEPARATE PAGE? YES// <RET>  
SORT BY: NAME// FOLLOWUP DATE=MAY 1, 1999  
DEVICE:
```

In this example, the word processing text found in the APPOINTMENT REMINDER entry of the FORM LETTER file is used to print a sheet of output for each EMPLOYEE file entry whose FOLLOWUP DATE equals May 1,1999.

If the document file contains a pointer field pointing to File #1, and if the document entry selected has a value for that pointer, then the file pointed to will be automatically used to print from and the user will not be asked "Print from what FILE:".

NOTE: The Read access is checked by DIWF for both files selected.

EN1^DIWF: Form Document Print

Form Document Print Introduction (^DIWF)

The entry points in ^DIWF are designed to use the contents of a word processing field as a target document into which data can be inserted at print time. The data may come from another VA FileMan file or be provided by the user interactively at the time the document is printed. A file containing a word processing type field is first selected and then an entry from that file. The word processing text in that entry is then used as a form with which to print output from any other file.

The word processing text used will typically include windows into which data from the target file automatically gets inserted by DIWF. The window delimiter is the vertical bar (|). Thus, if a word processing document contains |NAME| somewhere within it, DIWF will try to pick the NAME field (if there is one) out of the file being printed. Any non-multiple field label or computed expression can be used within a |-window, if:

1. an expression within the |-window cannot be evaluated, and
2. the output of DIWF is being sent to a different terminal than the one used to call up the output,

then the user will be asked to type in a value for the window, for each data entry printed. Thus, the word processing text used as a target document might include the window |SALUTATION|, where SALUTATION is not a valid field name in the source file. When DIWF encounters this window, and failing to find a SALUTATION field in the source file, it will ask the user to enter SALUTATION text which then immediately gets incorporated into the output in place of that window. Notice that we are referring to two files-the document file which contains the word processing text and the print from file which DIWF will use to try to fill-in data for the windows.

This entry point is used when the calling program knows which file (document file) contains the desired word processing text to be used as a target document.

Input Variable

DIC A file number or a global root. The file identified must contain a word processing field.

Output Variable

- Y** This will be -1 only if the file sent to DIWF in the variable DIC does not contain a word processing field.

Example

```
>S DIC=16001 D EN1^DIWF
```

The user will then be branched to the "Select DOCUMENT:" prompt in the dialog described above to select a particular entry in the Form Letter file.

EN2^DIWF: Form Document Print

Form Document Print Introduction (^DIWF)

The entry points in ^DIWF are designed to use the contents of a word processing field as a target document into which data can be inserted at print time. The data may come from another VA FileMan file or be provided by the user interactively at the time the document is printed. A file containing a word processing type field is first selected and then an entry from that file. The word processing text in that entry is then used as a form with which to print output from any other file.

The word processing text used will typically include windows into which data from the target file automatically gets inserted by DIWF. The window delimiter is the vertical bar (|). Thus, if a word processing document contains |NAME| somewhere within it, DIWF will try to pick the NAME field (if there is one) out of the file being printed. Any non-multiple field label or computed expression can be used within a |-window, if:

1. an expression within the |-window cannot be evaluated, and
2. the output of DIWF is being sent to a different terminal than the one used to call up the output,

then the user will be asked to type in a value for the window, for each data entry printed. Thus, the word processing text used as a target document might include the window |SALUTATION|, where SALUTATION is not a valid field name in the source file. When DIWF encounters this window, and failing to find a SALUTATION field in the source file, it will ask the user to enter SALUTATION text which then immediately gets incorporated into the output in place of that window. Notice that we are referring to two files-the document file which contains the word processing text and the print from file which DIWF will use to try to fill-in data for the windows.

This entry point is used when the calling program knows both the document file and the entry within that file which contains the desired word processing text to be used as a target document.

Input Variables

DIWF The global root at which the desired text is stored. Thus, in our example, if APPOINTMENT REMINDER is the third document in the Form Letter file (stored in ^DIZ(16001,)) and the word

processing field is stored in subscript 1, you can:

```
S DIWF="^DIZ(16001,3,1,"
```

DIWF will then automatically use this entry and the user will not be asked to select the document file and which document in that file.

DIWF(1) If the calling program wants to specify which file should be used as a source for generating output, the number of that file should appear in the variable DIWF(1). Otherwise, the user will be asked the "Print from what FILE:" question.

After this point, EN1^DIP is invoked. You can have the calling program set the usual BY, FR, and TO variables if you want to control the SORT sequence of the data file.

Output Variable

Y Y will be -1 if:

- There is no data beneath the root passed in DIWF.
- The file passed in DIWF(1) could not be found.

^DIWP: Formatter

Call ^DIWP to format and (optionally) output any group of text lines.

Before calling ^DIWP, you should kill the global ^UTILITY(\$J,"W").

^DIWP works in **two modes** (based on whether the DIWF input parameter contains "W" or not):

1. In ^DIWP's **"accumulate" mode**, repeated calls to ^DIWP accumulate and format text in ^UTILITY(\$J,"W"). After you have finished accumulating text, if you want to write the text to the current device, you should call ^DIWW. ^DIWW writes the accumulated text to the current device with the margins you specified in your calls to ^DIWP and then it removes the text from ^UTILITY.
2. In ^DIWP's **"write" mode**, if the text added to ^UTILITY(\$J,"W") by ^DIWP causes one or more (that is, n) line breaks, n lines are written to the current device (and the remaining partial line is stored in ^UTILITY). This leaves one line of text in ^UTILITY once all calls to ^DIWP are completed. To write the remaining line of text to the current device and remove it from ^UTILITY, call ^DIWW.

Input Variables

X The string of text to be added as input to the formatter.

The X input string may contain | -windows, as described in the Formatting Text with Word Processing Windows topic in the Advanced Edit Techniques chapter of the *VA FileMan Advanced User Manual* (e.g., |SETTAB(9,23,44)|). The expressions within the windows will be processed as long as they are not context-dependent; that is, as long as they do not refer symbolically to database field names. Thus, |TODAY| will cause today's date to be inserted into the formatted text, but |SSN| will be printed out as it stands, because it cannot be interpreted in context.

DIWL The (integer-valued) left margin for the text. Set this to a positive number, 1 or greater. Do not change the value of DIWL if you are making repeated calls to ^DIWP to format text.

- DIWR** The (integer-valued) right margin for the text.
- DIWF** A string of format control parameters. If contained in DIWF, the parameters have the following effects:
- Cn** The text will be formatted in a **Column** width of **n**, thus overriding the value of DIWR.
 - D** The text will be in **Double-spaced** format.
 - In** The text will be **Indented** n columns in from the left margin (DIWL).
 - N** Each line will be printed as it appears in the text (**No-wrap**). If DIWF contains N, the value of DIWR will be ignored. See the *Advanced Edit Techniques* chapter in the *VA FileMan Advanced User Manual* for details about word wrapping.
 - R** The text will be in **Right-justified** format.
 - W** If the DIWF parameter contains "W", ^DIWP operates in "**Write**" mode. If the DIWF parameter does not contain "W", ^DIWP operates in "accumulate" mode. See above for the discussion of these two modes.

When making repeated calls to ^DIWP, don't mix modes. Use "write" or "accumulate" mode, but don't switch between them.
- | Word processing windows (material within **vertical bars**) will not be evaluated. The window will print as it exists in the word processing field.

^DIWW: WP Print

Use ^DIWW to output to the current device the remaining text left in ^UTILITY(\$J,"W") by ^DIWP.

The ^DIWW entry point is designed to be used in conjunction with the ^DIWP entry point. Using ^DIWP, you can accumulate and format text in ^UTILITY(\$J,"W"), in one of **two modes**:

3. In ^DIWP's **"accumulate" mode**, repeated calls to ^DIWP accumulate and format text in ^UTILITY(\$J,"W"). When you have finished accumulating text, you should call ^DIWW to write the text to the current device. ^DIWW writes the accumulated text to the current device with the margins you specified in your calls to ^DIWP and then removes the text from ^UTILITY.
4. In ^DIWP's **"write" mode**, if the text added to ^UTILITY(\$J,"W") by ^DIWP causes one or more (that is, n) line breaks, n lines are written to the current device (and the remaining partial line is stored in ^UTILITY.) This leaves one line of text in ^UTILITY once all calls to ^DIWP are completed. To write the remaining line of text to the current device and remove it from ^UTILITY, call ^DIWW.

%DT: Introduction to Date/Time Formats

This introduction pertains to all of the %DT calls which follow. Please read this first because it is relevant to all of the %DT calls.

%DT is used to validate date/time input and convert it to VA FileMan's conventional internal format: "YYMMDD.HHMMSS", where:

- **YYY** is number of years since 1700 (hence always 3 digits)
- **MM** is month number (00-12)
- **DD** is day number (00-31)
- **HH** is hour number (00-23)
- **MM** is minute number (01-59)
- **SS** is the seconds number (01-59)

This format allows for representation of imprecise dates like JULY '78 or 1978 (which would be equivalent to 2780700 and 2780000, respectively). Dates are always returned as a canonic number (no trailing zeroes after the decimal).

^%DT: Internal to External Date

Introduction to Date/Time Formats: %DT

%DT is used to validate date/time input and convert it to VA FileMan's conventional internal format: "YYMMDD.HHMMSS", where:

YYY is number of years since 1700 (hence always 3 digits)

MM is month number (00-12)

DD is day number (00-31)

HH is hour number (00-23)

MM is minute number (01-59)

SS is the seconds number (01-59)

This format allows for representation of imprecise dates like JULY '78 or 1978 (which would be equivalent to 2780700 and 2780000, respectively). Dates are always returned as a canonic number (no trailing zeroes after the decimal).

This routine accepts input and validates the input as being a correct date and time.

Input Variables

%DT	A string of alphabetic characters which alter how %DT responds. Briefly stated, the acceptable characters are:
A	Ask for date input.
E	Echo the answer.
F	Future dates are assumed.
I	For I nternationalization, assume day number precedes month number in input.
M	Only M onth and year input is allowed.
N	Pure N umeric input is not allowed.
P	P ast dates are assumed.
R	R equires time input.
S	S econds should be returned.

T Time input is allowed but not required.

X EXact input is required.

For an explanation of each character, see %DT Input Variables in Detail below.

X If %DT does not contain an A, then the variable X must be defined as equal to the value to be processed. See Date Fields in the Editing Specific Field Types chapter of the *VA FileMan Getting Started Manual* for acceptable values for X and for the interpretation of those values.

%DT("A") (Optional) A prompt which will be displayed prior to the reading of the input. Without this variable, the prompt "DATE:" will be issued.

%DT("B") The default answer to the "DATE:" prompt. It is your responsibility to ensure that %DT("B") contains a valid date/time. Allowable date input formats are explained in the Editing Specific Field Types chapter of the *VA FileMan Getting Started Manual*.

%DT(0) (Optional) Prevents the input date value from being accepted if it is chronologically before or after a particular date. Set %DT(0) equal to a VA FileMan-format date (e.g., %DT(0)=2690720) to allow input only of dates greater than or equal to that date. Set it negative (e.g., %DT(0)=-2831109.15) to allow only dates less than or equal to that date/time. Set it to NOW to allow dates from the current (input) time forward. Set it to -NOW to allow dates up to the current time.

NOTE: Be sure to kill this variable after returning from %DT.

Output Variables

Y %DT always returns the variable Y, which can be one of two values:

Y=-1 The date/time was invalid.

Y=YYMMDD.HHMMSS The value determined by %DT.

X X is always returned. It contains either what was passed to %DT (in the case where %DT did not contain an A) or what the user entered.

DTOUT This is only defined if %DT has timed-out waiting for input from the user.

%DT Input Variables in Detail

A %DT Asks for input from the terminal. It continues to ask until it receives correct input, a null, or an up-arrow. If %DT does not contain the character A, the input to %DT is assumed to be in the variable X.

E The **External** format of the input will be echoed back to the user after it has been entered. If the input was erroneous, two question marks and a "beep" will be issued.

F If a year is not entered (example 1), or if a two-digit year is entered (example 2), a date in the **Future** is assumed.

EXCEPTION: If a two-digit year is entered and those two digits equal the current year, the current year is assumed even if the date is in the past (example 3).

Example	Current Date	User Input	Date Returned	Returned Without F
1)	July 1, 2000	5/1	May 1, 2001	May 1, 2000
2)	July 1, 2000	5/1/90	May 1, 2090	May 1, 1990
3)	July 1, 2000	5/1/00	May 1, 2000	May 1, 2000

See Y2K Changes below for the behavior of %DT when neither the F nor P flag is used.

I For **I**nternalization, this flag makes %DT assume that in the input, the day number precedes the month number. For example, input of 05/11/2000 is assumed to be November 5, 2000 (instead of May 11, 2000). Also, with this flag, the month must be input as a number.

For example, November must be input as 11, not NOV.

M Only **M**onth and year input is allowed. Input with a specific day or time is rejected (example 1). If only a month and two digits are entered, the two digits are interpreted as a year instead of a day (example 2).

If the M flag is used with the X flag, a month must be specified; otherwise, the input can be just a year (example 3).

M Flag

Example	Date Input	Date Returned	Returned Without M
1)	7-05-2005	invalid	July 5, 2005
2)	7-05	July 2005	July 5, 2000*

*Assuming the current year is 2000 and the F and P flags aren't used.

M Flag (with X Flag)

Example	Date Input	Date Returned	Returned Without X
3)	05 or 2005	invalid	2005

N Ordinarily, a user can enter a date in a purely **N**umeric form, i.e., MMDDYY. However, if %DT contains an N, then this type of input is not allowed.

P If a year is not entered (example 1), or if a two-digit year is entered (example 2), a date in the **Past** is assumed.

EXCEPTION: If a two-digit year is entered and those two digits equal the current year, the current year is assumed even if the date is in the future (example 3).

Ex.	Current Date	User Input	Date Returned	Returned Without P
1)	March 1, 1995	6/1	June 1, 1994	June 1, 1995
2)	March 1, 1995	6/1/98	June 1, 1898	June 1, 1998
3)	March 1, 1995	6/1/95	June 1, 1995	June 1, 1995

See Y2K Changes below for the behavior of %DT when neither the F nor P flag is used.

R Time is **Required**. It must be input.

S Seconds are to be returned.

T Time is allowed in the input, but it is not necessary. See Date Fields in the Editing Specific Field Types chapter of the *VA FileMan Getting Started Manual* for details of how user-input times are interpreted.

X **EX**act input is required. If X is used without M, date input must include a day and month. Without X, the input can be just month-year or only a year.

If X is used with M, date input must include a month. If M is used without X, then the input can be just a year.

Y2K Changes:

If no year is entered, the current year is assumed (example 1).

If a two-digit year is entered, a year less than 20 years in the future and no more than 80 years in the past is assumed. For example, in the year 2000, two-digit years are assumed to be between 1920 through 2019.

NOTE: Only the year, not the current month and day, is taken into account in this calculation (examples 2 through 5).

Example	Current Date	User Input	Date Returned
1)	Sep 15, 2000	3/15	Mar 15, 2000
2)	Sep 15, 2000	1/1/20	Jan 01, 1920
3)	Sep 15, 2000	12/31/20	Dec 31, 1920
4)	Sep 15, 2000	1/1/19	Jan 01, 2019
5)	Sep 15, 2000	12/31/19	Dec 31, 2019

DD^%DT: Internal to External Date

Introduction to Date/Time Formats: %DT

%DT is used to validate date/time input and convert it to VA FileMan's conventional internal format: "YYMMDD.HHMMSS", where:

YYY is number of years since 1700 (hence always 3 digits)

MM is month number (00-12)

DD is day number (00-31)

HH is hour number (00-23)

MM is minute number (01-59)

SS is the seconds number (01-59)

This format allows for representation of imprecise dates like JULY '78 or 1978 (which would be equivalent to 2780700 and 2780000, respectively). Dates are always returned as a canonic number (no trailing zeroes after the decimal).

There are two ways to convert a date from internal to external format—this call and X ^DD("DD"). (This is the reverse of what %DT does.) This entry point takes an internal date in the variable Y and converts it to its external representation.

Example 1

```
>S Y=2690720.163 D DD^%DT W Y
JUL 20, 1969@1630
```

This results in Y being equal to JUL 20, 1969@16:30. (Single space before the 4-digit year.)

Input Variables

- | | |
|------------|---|
| Y | (Required) This contains the internal date to be converted. If this has five or six decimal places, seconds will automatically be returned. |
| %DT | (Optional) This forces seconds to be returned even if Y does not have that resolution. %DT must contain S for this to happen. |

Output Variable

Y Y is returned as the external form of the date.

See also DT^DIO2, which takes an internal date in the variable Y and *writes out* its external form.

^%DTC: Date/Time Utility

^%DTC returns the number of days between two dates.

Input Variables

- X1** (Required) One date in VA FileMan format. This is not returned.
- X2** (Required) The other date in VA FileMan format. This is not returned.

Output Variables

- X** The number of days between the two dates. X2 is subtracted from X1.
- %Y** If %Y is equal to 1, the dates have both month and day values.
If %Y is equal to 0, the dates were imprecise and therefore not workable.

C^%DTC: Date/Time Utility

C^%DTC takes a date and adds or subtracts a number of days, returning a VA FileMan date and a \$H format date. If time is included with the input, it will also be included with the output.

Input Variables

- X1** (Required) The date in VA FileMan format to which days are going to be added or from which days are going to be subtracted. This is not returned.
- X2** (Required) If positive, the number of days to add. If negative, the number of days to subtract. This is not returned.

Output Variables

- X** The resulting date, in VA FileMan format, after the operation has been performed.
- %H** The \$H form of the date.

COMMA^%DTC: Date/Time Utility

Formats a number to a string that will separate billions, millions, and thousands with commas.

Input Variables

- X** (Required) The number you want to format. X may be positive or negative.
- X2** (Optional) The number of decimal digits you want the output to have. If X2 is not defined, two decimal digits are returned. If X2 is a number followed by the dollar sign (e.g., 3\$) then a dollar sign will be prefixed to X before it is output.
- X3** (Optional) The length of the desired output. If X3 is less than the formatted X, X3 will be ignored. If X3 is not defined, then a length of twelve is used.

Output Variable

- X** The initial value of X, formatted with commas, rounded to the number of decimal digits specified in X2. If X2 contained a dollar sign, then the dollar sign will be next to the leftmost digit. If X was negative, then the returned value of X will be in parentheses. If X was positive, a trailing space will be appended. If necessary, X will be padded with leading spaces so that the length of X will equal the value of the X3 input variable.

Examples

Example 1

```
>S X=12345.678 D COMMA^%DTC
```

The result is:

```
X=" 12,345.68 "
```

Example 2

```
>S X=9876.54,X2="0$" D COMMA^%DTC
```

The result is:

```
X="      $9,877 "
```

Example 3

```
>S X=-3,X2="2$" D COMMA^%DTC
```

The result is:

```
X="      ($3.00) "
```

Example 4

```
>S X=12345.678,X3=10 D COMMA^%DTC
```

The result is:

```
X="12,345.68 "
```

DW^%DTC: Date/Time Utility

This entry point produces results similar to H^%DTC. The difference is that X is reset to the name of the day of the week—Sunday, Monday, and so on. If the date is imprecise, then X is returned equal to null.

H^%DTC: Date/Time Utility

H^%DTC converts a VA FileMan date/time to a \$H format date/time.

Input Variable

X (Required) The date/time in VA FileMan format. This is not returned.

Output Variables

%H The same date in \$H format. If the date is imprecise, then the first of the month or year is returned.

%T The time in \$H format, i.e., the number of seconds since midnight. If there is no time, then %T equals zero.

%Y The day-of-week as a numeric from 0 to 6, where 0 is Sunday and 6 is Saturday. If the date is imprecise, then %Y is equal to -1.

HELP^%DTC: Date/Time Utility

This entry point displays a help prompt based on %DT and %DT(0).

Input Variables

- %DT** The format of %DT is described in the %DT section of this chapter. The help prompt will display different messages depending on the parameters in the variable.
- %DT(0)** (Optional) The format of %DT(0) is described in the %DT section of this chapter. This input variable causes HELP to display the upper or lower bound that is acceptable for this particular call.

NOW^%DTC: Date/Time Utility

NOW^%DTC returns the current date/time in VA FileMan and \$H formats.

Output Variables

%	VA FileMan date/time down to the second.
%H	\$H date/time.
%I(1)	The numeric value of the month.
%I(2)	The numeric value of the day.
%I(3)	The numeric value of the year.
X	VA FileMan date only.

S^%DTC: Date/Time Utility

This entry takes the number of seconds from midnight and turns it into hours, minutes, and seconds as a decimal part of a VA FileMan date.

Input Variable

% A number indicating the number of seconds from midnight, e.g.,
SP(\$H,"",2).

Output Variable

% The decimal part of a VA FileMan date.

Example

```
>SET %=44504 D S^%DTC W %  
.122144
```

YMD^%DTC: Date/Time Utility

Converts a \$H format date to a VA FileMan date.

Input Variable

%H (Required) A \$H format date/time. This is not returned.

Output Variables

% Time down to the second in VA FileMan format, that is, as a decimal. If %H does not have time, then % equals zero.

X The date in VA FileMan format.

YX^%DTC: Date/Time Utility

This entry point takes a \$H date and passes back a printable date and time. It also passes back the VA FileMan form of the date and time.

Input Variable

%H (Required) This contains the date and time in \$H format which is to be converted. Time is optional. This is not returned.

Output Variables

Y The date and time (if time has been sent) in external format. Seconds will be included if the input contained seconds.

X The date in VA FileMan format.

% The time as a decimal value in VA FileMan format. If time was not sent, then % will be returned as zero.

%XY^%RCR: Array Moving

This entry point can be used to move arrays from one location to another. The location can be local or global.

After the call has completed, both arrays are defined. They are identically subscripted if the %Y array did not previously exist. If the array identified in %Y had existing elements, those elements will still exist after the call to %XY^%RCR. However, their values may have to be examined because an identically subscripted element in the %X array will replace the one in the %Y array, but an element which existed in the %Y array (but not in the %X array) will remain as it was.

Input Variables

- %X** The global or array root of an existing array. The descendents of %X will be moved.
- %Y** The global or array root of the target array. It is best if this array does not exist before the call.

Example

To move the local array X(to ^TMP(\$J, you would write:

```
>S %X="X(" S %Y="^TMP($J," D %XY^%RCR
```