ULT

UPDATE

THE ULTIMATE CORP.

U P D A T E
L A N G U A G E
D O C U M E N T A T I O N

0 1 O C T 1 9 8 5

TABLE OF CONTENTS

CHAPTER 1

INTRODUCTION

1.1 Introduction

The purpose of the UPDATE language is to provide a generalized methodology for updating Ultimate databases via non-procedural, non-programmed UPDATE statements. UPDATE statements normally provide a screen-oriented and interactive approach toward editing data fields.

A programmed language is called "procedural" because the user has to specify both what the process is to achieve, as well as how to achieve it. With a non-procedural language, such as UPDATE, the user does not specify how the process is to act, but merely what it is to achieve. The significant advantage of a non-procedural language is simplicity since the user does not have to be a programmer in order to create the desired function.

1.2 How the UPDATE language works

The UPDATE language utilizes the dictionary of the file(s) being updated to define the nature of data storage, validation, etc. The dictionary definitions are an extension of those used by the RECALL language, as described later.

The UPDATE language processor is invoked by an UPDATE statement at the TCL level. This statement calls into effect the set of dictionary definitions that are used to define and validate the affected fields.

1.3 Definitions

Term	Definition
UPDATE statement	This is the TCL statement used to invoke the UPDATE processor. It contains the set of UPDATE expressions and the options or connectives.
UPDATE expression	An UPDATE expression consists of an attribute name from the dictionary of the file being updated, along with elements that specify the cursor location of the field, whether it is part of a "window" or not, the "tag" associated with the field, etc.
Primary file	The filename explicitly specified in the UPDATE statement. Just as in RECALL, an UPDATE statement may reference only one file explicitly. The dictionary of this file contains information used by the UPDATE processor (the USING connective of RECALL is invalid in an UPDATE statement).
Secondary file	More than one file may be updated by the UPDATE statement since the primary file dictionary can have links established to other files, which are known as secondary files.
Field	An attribute in the item being updated; a field may be either single or multiple valued. The UPDATE language does not allow sub-multiple values.
Controlling field	The controlling field is a multi valued attribute which has a 'C' code defined in the V/STRUC field of its dictionary definition item. It is the primary field used in an Associated data structure, and may have one or more Dependent fields associated with it.
Dependent field	A dependent field is a multi valued attribute which has a 'D' code defined in the V/STRUC field of its dictionary definition item. There may be one or more dependent fields associated with each Controlling field; the entire set is called an Associated data structure.
Generated field	A definition in the dictionary that has an AMC of zero and a Function (RECALL style, Algebraic or F;) used to create a new data value, which may be mathematical compuation, string manipulation, concatenation, etc.

1.4 Database aspects

RECALL and UPDATE are both dictionary-driven languages, where the dictionary definitions serve to define the database. Since RECALL is a retrieval-only language, it can have unlimited freedom in its data definitions. RECALL cannot affect the database adversely if its definitions are not consistent.

UPDATE cannot afford the same freedom, since the integrity of the database can easily be affected if improper dictionary definitions are used. A major goal of the implementation of the UPDATE language is to ensure against inadvertant destruction of the database, especially since both dictionary definitions and the UPDATE statements that use them can be changed so readily.

For this reason, the dictionary definitions used in UPDATE must follow more rigid patterns than used in RECALL. Also, the dictionary definitions must be validated by a utility program, UPD-VALIDATE, before use. This utility checks the entire set of definitions in all related files for consistency, and also performs some "pre-compilation" steps which improve the efficiency of the UPDATE processor.

In general, existing RECALL definitions can be used by UPDATE in a "display-only" mode (which is non-destructive of data); definitions used by UPDATE to actually affect data must be validated before use.

The format of such dictionary definitions is described later.

1.5 Design considerations

The following considerations should be kept in mind when designing applications using the UPDATE language.

1.5.1 Using multivalues

Try to make the primary file have as much data as possible, including "associated" multivalues such as is typical in a "line-item" of invoices and orders. Storing such information in secondary files is possible, but considerably less efficient.

If it is absolutely necessary to create a "detail line-item" file, there is an alternative to using the secondary file technique built into the UPDATE language. It may be more efficient to use the "after read" and "before write" subroutines to copy the secondary information as multiple values into the primary file item for the duration of the update.

1.5.2 Concatenated values

Concatenated values may be used freely for item-id's, both in the primary and the secondary files.

Attributes may have concatenated values as well, though the processing time increase may not make their use worthwhile if an alternative is available.

Secondary attributes defined as concatenated must be stored as a complete value; that is, you cannot store into only a segment of a secondary concatenated value.

Segments of a concatenated value must use the same separator; that is, a value such as A*1111*9877 (three segments) is acceptable, but A*1111#9877 is not (where "#" is meant as a separator).

1.5.3 AMC and VMC limits

AMC limitations: The UPDATE processor can update files with a maximum of 99 attributes. Do not try to use UPDATE for existing files with items containing more than 99 attributes, or the data will be truncated.

VMC limitations: An attribute defined as multivalued can have up to 99 multiple values.

Sub-multiple values are not supported.

1.5.4 Secondary file limits

Up to nine secondary files can be defined.

1.5.5 Secondary file data storage

You cannot store into a secondary file Controlling-Dependent relationship. Therefore, the best way to design secondary files is to make them have single-valued attributes except when a "cross-index" type of file is being updated. A cross-index file typically has a field or fields with multivalues, but they are not "associated" fields.

1.5.6 Generated field limits

Up to twenty computed fields may be defined.

CHAPTER 2

SYNTAX

2.1 The UPDATE Statement

The UPDATE processor is invoked via a TCL statement. The UPDATE syntax is an extension of the RECALL syntax (and may also be used to generate reports only).

The general form of an UPDATE Statement is:

```
UPDATE  filename
    update.expression.for.item.id    { update.expression .... }
                  { modifiers }
                 { ( options ) }
```

As can be seen, the minimum UPDATE Statement consists of the UPDATE verb, the filename and one update expression for the item-id. The filename specified in the statement is called the "primary file". The UPDATE language can be used to update other files as well; these are called "secondary files" and the dictionary definitions in the primary file contain links that define the nature of these secondary updates.

UPDATE expressions are used to position the updated fields and are described below. A maximum of 70 expressions may be specified.

Standard RECALL modifiers such as HEADING, etc. are valid in an UPDATE statement, although the meanings of some of these may be different somewhat in the UPDATE context.

2.2 Interaction with Select lists

The UPDATE statement may be preceded by a SELECT, SSELECT, GET-LIST or QSELECT statement.

In this case, the first item-id from the select list is brought up for editing; when that item is filed, the item-id still displays and may be re-edited by using the Backspace key at the IDO prompt. If a RETURN is entered, the next item from the select list is brought up, and so on until the list is exhausted.

An 'EXT' or 'STOP' entered at the Edit request line unconditionally exits from the UPDATE language.

2.3 UPDATE Expressions

An UPDATE expression is the main element used to create a screen-oriented UPDATE statement. The general form of an UPDATE expression is:

```
@(x,y) : "tag field" : @(x) : attributename [substring expression]
        ...optional.......                    ...optional........
```

where:

(x,y) is the cursor address where the literal "tag field" is to be displayed.

"tag field" is a literal field that is used to describe the datum on the screen. This field and the next are optional.

(x) is used to specify the leftmost position of the datum. If absent, the field starts just after the "tag field".

"attributename" is a name from the dictionary of the primary file being updated. If this is absent, the expression is used only to generate a literal on the screen.

The following rules apply to the 'x' and 'y' values.

'y' should be in the range 1 through 21. Rows 22 and 23 on the screen are normally reserved for the EDIT line request and error messages, respectively.

'x' should be greater than 3 and in a range such that the entire tag and/or field will fit in the terminal width. The UPDATE processor automatically prefixes a two digit or single character field identification (field-id) to the tag field.

The optional "substring expression" is similar to that used in BASIC, and is used to limit the width of the output, mainly for descriptive fields, without having to define a synonym dictionary attribute definition. If a substring expression exists, the associated field is automatically considered "display only", that is, it cannot be updated and the cursor will bypass it during the update process.

Spaces may be included between elements of an UPDATE expression for clarity, but are not needed.

Examples of UPDATE expressions are:

```
@(10,12) : "Address line 2" : @(25) : ADDR2
@(15) : NAMEX
@(3,20)  : DESCRIPTION [1,20]
@(5,11)  : 'This is only a literal string.........
ZIP
```

2.4 The WINDOW and END-WINDOW connectives

These two connectives are used to specify the screen format of multi-valued attributes and of Associated data sets. A single valued attribute requires only the 'x' and 'y' cursor locations to locate the field on the screen. A multi-valued attribute also needs a specification indicating the number of displayed multiple values.

A subsection of the screen may be designated as a "window" by using the WINDOW and END-WINDOW connectives. The WINDOW connective specifies the 'x' and 'y' cursor locations of the top left-hand corner of the window, and the 'y' cursor location of the last row of the window.

Each 'window line' may use one or more rows on the screen; the default is one row per line. The number of rows per line is also specified in the WINDOW parameters. Its format is therefore:

```
WINDOW @(x,y,z,r) : 'literal' : @(x) : 'literal' ...
                    ..... optional ..................
```

Note that the syntax of the WINDOW parameters is analogous to a BASIC 'FOR row = y TO z STEP r' statement. The optional section of literals will print next to the field-id (see next section) assigned to the window.

Each WINDOW connective may be followed by one or more UPDATE expressions; such expressions use the 'x' cursor as usual to specify the column location where the field begins. The 'y' cursor location is, however, optional and defaults to 1, which is the first row of each 'line' of the window. In the case of multiple-row window lines, the 'y' location may be used to specify which row in the line the field is to appear.

For example, on a window set up by a 'WINDOW @(3,10,17,2)', which has two rows per window line, the cursor location '@(30)' would position the field at x-cursor 30 on the first row; the location '@(30,2)' would do so on the second row.

All UPDATE expressions between a WINDOW and its corresponding END-WINDOW connective are considered to be part of that window, and must reference multi-valued data. Single-valued expressions should not be specified within the scope of these connectives. The END-WINDOW connective has no parameters.

The UPDATE statement may contain any number of WINDOW-END-WINDOW pairs.

An example of a window:

```
Column  ->          1         2         3         4
           12345678901234567890123456789012345678901
 Row
  4         ...ff.....................................
  5         ...01.aaaaaaaaaaaaaaaaaa...bbbb..cccccc.
  6         ...02.aaaaaaaaaaaaaaaaaa...bbbb..cccccc.
  7         ...03.aaaaaaaaaaaaaaaaaa...bbbb..cccccc.
  8         ...04.aaaaaaaaaaaaaaaaaa...bbbb..cccccc.
  9         .........................................
```

The fields are represented by 'a' and 'b' and 'c'. 'ff' is the field-id for the first field in the window. This window may be set up by the following UPDATE expressions:

```
... WINDOW @(7,5,8)   @(7):attribute.a   @(28):attribute.b
    @(34):attribute.c   END-WINDOW ...
```

2.5 Field identification

Each field in the UPDATE statement that can be updated (except the item-id) is assigned a **field-id** by the UPDATE processor. This is normally a number starting from 1 up to 70. The UPDATE language allows up to 70 fields to be updated on one screen. Optionally, the field-id can be the alphabetic characters 'a' through 'z'; see the 'A' option. In this case, the number of fields on one screen are limited to 26.

The field-id is used by the operator when the cursor is to be positioned on a field by explicitly using that number or character from the EDIT line prompt.

The field-id is displayed as follows:

1. **For fields that are not within a window**, the field-id is prefixed to the tag and thus appears three columns to the left of the tag. For example, if the UPDATE expression is:

   ```
   ... @(25,10):"Name":CNAME...
   ```

   and the field-id is "7", the field would appear as:

   ```
           2         3
    col:   01234567890123456789

              7.Name xxxxxxxxxx
   ```

2. **When a window is defined, it is assigned a field-id**, which appears three columns to the left of the 'x' cursor defined in the WINDOW connective. and **one row before the 'y' parameter**. If there is a literal defined with the window, it prints on the same line as the field-id.

   Fields within the window do not have field-ids, since they can be referenced only after going to the first field of the window.

   For example,

   ```
    ...WINDOW @(4,10,18):" INVOICE# INV-DATE"  @(9):INV.NUM  @(17):INV.DATE
                          1         2         3         4
     col:  012345678901234567890123456789012345678901234567890
     row
       9      c. INVOICE#   INV-DATE
      10      01 invnum     mm/dd/yyyy
      11      02 invnum     mm/dd/yyyy
      12      03 invnum     mm/dd/yyyy
   ```

   etc.

2.6 Typical screen format

```
  |-------------------------------------------------------------------------|
 0|Order Entry Program                              OE.02  01 May 1983  10:01:01 |*1
 1|                                                                         |
 2|                                                                         |
 3|  ORDER NUMBER  : 1765                                                   |*2
 4|a.Customer Id   : 943                    Name  : ACME Screw Fasteners Inc. |*3
 5|b.Shipping Addr : 17678 San Pablo Avenue                                 |*3
 6|c.              :                                                       |
 7|d.City          : Berkeley                                              |
 8|e.State         : CA                   f.Zip  : 94000                   |
 9|g.Salesman id   : 178                    Name  : Smith, John             |
10|                                                                         |
11|h.    Inv#   ........Description......... Qty ord Qty shp Price   Extension|*4
12|01    P-100A Screws, 2x12  Galv             100     100    10.20   1,020.00 |*5
13|02    P-150A Screws, 2x15  Galv             150     150    12.50   1,875.00 |*5
14|                                                                         |
15|                                                                         |
16|                                                                         |
17|                                                                         |
18|                                                                         |
19|i.Discount:  150.00       j.Freight:  50.00        Order total:  $2,795.00 |
20|                                                                         |
21|                                                                         |
22|Enter field id to change, * to void, or RETURN to post           .....  .. |*6
23|                                                                         |*7
  |-------------------------------------------------------------------------|
```

*1- Heading line
*2- Item-id of file
*3- Fields with field-identifiers
*4- "Window" fields
*5- Multiple associated values
*6- "Edit request"
*7- Reserved for error messages

UPDATE language Syntax

2.6.1 UPDATE statement for above screen

```
UPDATE ORDER @(3,3) : "ORDER NUMBER  :" : IDO
             @(3,4) : "Customer id   :" : @(20) : CUST.NUM
             @(47)  : "Name   :"        : @(54) : CUST.NAME
             @(3,4) : "Shipping Addr :" : @(20) : SHIP.ADDR1
             @(3,5) : ""                : @(20) : SHIP.ADDR2
             @(3,6) : "City          :" : @(20) : SHIP.CITY
             @(3,7) : "State         :" : @(20) : SHIP.STATE
             @(47)  : "Zip   :"         : @(54) : SHIP.ZIP
             @(3,9) : "Salesman id   :" : @(20) : SALES.NUM
             @(47)  : "Name  :"         : @(54) : SALES.NAME
             @(4,11): "   Inv#   ........Description......... _
Qty ord Qty shp Price  Extension"
     WINDOW  @(3,12,17)
             @(7)   : PART.NUM
             @(15)  : PART.DESC           @(47) : QTY.ORD
             @(54)  : QTY.SHIP            @(61) : PRICE
             @(70)  : EXTEN
    END-WINDOW
             @(3,19): "Discount      :" : @(14) : DISCOUNT
             @(32)  : "Freight:"        : @(42) : FREIGHT
             @(55)  : "Order total:"    : @(70) : ORD.TOTAL
    HEADING "Order Entry Program                      OE.02 'T'"   (A)
```

2.6.2 The HEADING connective

The HEADING connective operates exactly the same as in RECALL and may be used to provide a non-variable heading for the screen. In the absence of the HEADING connective, no heading is displayed.

The standard RECALL codes for time and date may be used in the HEADING. The page number code is reserved to display a Screen number if multiple screens are used. The page number has no direct meaning during interactive updates, but contains the screen number when multiple screens are used and may be displayed as a indicator to the operator.

Headings may be any number of lines in length.

2.6.3 The FOOTING connective

The FOOTING connective has a special function in an UPDATE statement. In its absence, the UPDATE processor displays the default EDIT request on line 22 of the screen:

```
   Enter field id to change, * (EXit), RETURN (FIle)
```

If a different message is desired, the FOOTING connective may be used to display it. The message is automatically displayed on line 22 of the screen, and should contain no more than 70 characters (ten positions are required as a minimum for the data entry prompt on this line).

As with the HEADING, standard codes for time, date and "page" number

(actually the screen number) may be included.

2.6.4 The NEXT-SCREEN connective

The NEXT-SCREEN connective is used exclusively in UPDATE statements, and allows multiple screens on a single update statement, usually used when there are too many update or display fields to fit comfortably on one screen.

NEXT-SCREEN may not appear within a WINDOW - END-WINDOW set, or just after the item-id definitions.

When multiple screens are specified, the operator may move from one screen to another from the Edit Request line, by using the Backspace or Return keys. The Screen number may be displayed by using the "'P'" code in a HEADING statement.

2.7 Options

Options are single characters enclosed in parentheses at the end of the UPDATE statement. For example, '(A)' or '(A,R)'

2.7.1 A option

The UPDATE processor normally uses numbers as field identifiers so that the operator can reference a field from the EDIT line. If the A option is used, lower-case alphabetic characters are used instead of numbers, that is, 'a', 'b', 'c', etc. If this option is in effect, only up to 26 fields can be present.

2.7.2 B option

Normally, the BREAK key is enabled while in UPDATE (except that, after any secondary file items have been written, it is disabled to ensure data-base integrity until the primary item has been written or deleted).

If the BREAK key is to be always disabled while in UPDATE, the B option may be used.

2.7.3 M option

This option is called the "noModify" option; if used, existing items can be displayed but not edited; new items can be created and edited while on the screen.

2.7.4 N option

This option prevents New items from being created; existing items can be retrieved and edited.

2.7.5 R option

Normally, when the maximum number of characters specified in a field are entered, an automatic carriage-return is assumed. If the R option is used, a carriage-return must be entered to terminate the field.

2.7.6 V option

This option is called the View option; if used, existing items can be retrieved for display only; no changes can be made.

2.7.7 X option

The entire primary item can be deleted only if the X option is used.

CHAPTER 3

DICTIONARY DEFINITIONS

3.1 Dictionary attribute definitions

The UPDATE processor is driven by a special set of definitions in the Dictionary of the primary file that is being updated. The Dictionary definitions are an extension of those used by RECALL for listing and reporting purposes.

As far as possible, compatibility is maintained with the existing RECALL structure. Every Dictionary item used in an UPDATE expression must however have a D/CODE of 'U'; an existing RECALL Dictionary item with its already-defined attributes may not be used except as a display-only field.

An UPDATE Dictionary item-id cannot be numeric.

3.2 The ID0 and other item-id definitions

The UPDATE Dictionary definition "ID0" is mandatory and is used to define the item-id of the data file. All fields in this dictionary item are as usual, with the exception that some fields are unused; the UPD-DEF utility will skip over unused fields.

The V/DEFAULT, V/PRESUB and V/POSTSUB

fields (described later) are used

differently in the ID0 item.

In the case of an item-id that is comprised of several concatenated segments, there must be one dictionary definition for each segment. The definitions must be called ID0, ID1, etc. Each such definition must have a Group extract Correlative, and must be in correct sequence with no missing segments.

Example: if the item-id has three segments such as "string*code*date", there would be three dictionary definitions:

Name	ID0	ID1	ID2
Correlative	G0*1	G1*1	G2*1

All ID Dictionary definitions must have an AMC of zero.

3.3 Dictionary attribute 1 - D/CODE

Dictionary definitions used to update fields must have a code of 'U', and must be edited by the UPD-DEF utility rather than the system EDITOR. The UPD-DEF utility automatically stores the 'U' code.

The set of 'U' code Dictionary definitions must also be validated by the UPD-VALIDATE utility (described later) before the UPDATE processor can be invoked. The Dictionary validation performed by the utility ensures that a self-consistent set of definitions have been built for use by the UPDATE processor. Changes in any of the 'U' code definitions require the validation process to be run again.

3.4 AMC - Attribute number

The AMC field in an UPDATE Dictionary definition is identical to that in a RECALL definition, with the restriction that its value in a 'U' code definition must be in the range 0-99.

Zero is reserved for the item-id definitions (ID0, ID1, etc.) and for "generated" fields which have Functions used to generate, but not store, data in the primary file. If a field with a Function correlative has a non-zero amc, the generated value is stored in the primary file. Generated fields are usually used to perform some function on the data (such as multiplication of price by quantity) for storage in secondary files; the function may involve mathematical or string manipulation.

3.5 TAG or HEADING

The tag or heading field (S/NAME) in an UPDATE Dictionary definition is identical to that in a RECALL definition.

It is used to "tag" an UPDATE expression when the user does not specify one. For example, in the expression:

```
@(10,3):PRICE
```

the S/NAME of the PRICE definition would be used to the left of column 10 as a "tag". If S/NAME is null, "PRICE" itself would be used.

For complete control of screen tags, specify the tag in the expression:

```
@(10,3):"List price":@(22):PRICE
```

In order to inhibit the tag entirely, use an explicit null tag:

```
@(10,3):"":PRICE
```

3.6 STRUCture CODE

The structure code (V/STRUC) is particularly important in UPDATE Dictionary definitions. This is where the Controlling-Dependency relationship of multi-valued fields is specified.

Fields that contain **Single values** must have no entry (Null) in their structure code.

Fields that may contain **Multiple values** must have one of the following structure codes:

a. Controlling fields

```
[Internal format]   C(n);dependent1;dependent2;.....

[UPD-DEF display]   C(n)      dependent1
                              dependent2
                              ...
```

"C" defines this field as a Controlling field.

(n) is optional and specifies a limit to the number of allowable multiple values; the maximum number allowed is 99. If (n) is not specified, 99 is assumed.

dependent1, dependent2, etc. are the **names of the fields which are dependent on this controller**. These names must also exist as UPDATE Dictionary definitions, or the dictionary will not be validated.

Example: if the multi-valued PARTNO field controls PRICE, QUANTITY and EXTENSION, (all of which are UPDATE Dictionary definitions), the structure code of PARTNO would be:

```
C          PRICE
           QUANTITY
           EXTENSION
```

b. Dependent fields must have the following code:

```
[Internal format]   D;controller
[UPD-DEF display]   D         controller
```

"controller" is the **name** of the UPDATE Dictionary attribute which controls this dependent field.

c. Multivalued fields which are not related to other fields

```
M(n)
```

Note - the majority of multivalued fields fall into the Controlling or Dependent categories. Multivalued fields which are neither Controlling nor Dependent must have a structure code of "M"; the (n) is optional and specifies a limit to the number of allowable multiple values; the maximum number allowed is 99. If (n) is not specified, 99 is assumed.

3.7 CONVERSION CODES

The conversion codes field (V/CONV) in an UPDATE Dictionary definition is identical to that in a RECALL definition, except that it is more restrictive.

Multiple Conversion codes are allowed. They are processed left-to-right on Output, which is the normal RECALL convention, and right-to-left to verify data entry.

The acceptable Conversion codes are:

a. **Date** - Any Date Conversion such as "D", "D2/" etc. This should be the **last** code if multiple codes are used.

b. **Number** - Any Mask Decimal code, with or without format masking, for example "MR2", "MR2,$#10" etc. This should be the **last** code if multiple codes are used.

   Special notes:
   1. For numbers **without decimal points** such as Quantities, a Conversion of "MR0" will force data entry of valid numbers only. If numbers **with leading zeroes** are to be stored, use instead a Format Mask field such as "MR%%%" or "MR%3"

   2. To force **positive numbers only**, use the 'N' option on the Mask Decimal code, such as "MR0N", "MR2N,$#10", etc.

c. **Format masks** - this may be combined with the Decimal Conversion as shown above, or used by itself. This should be the **last** code if multiple codes are used.

   If used by itself, a Mask may be used to validate text strings; a Mask starts with an "M", followed by an "L" (left-justified), an "R" (right-justified), or a "V" (verify exact match).

   Following this are the standard format codes :

#	- Accept ANY character in this position.
*	- As for "#".
%	- Accept NUMERIC characters only.
&	- Accept ALPHABETIC characters only.
Anything else	- Accept (but do not store) matching character IF IT EXISTS, else IGNORE (that is, special characters are always optional).

Example: For a **telephone number**, the mask "MR%3-%4" may be used, and will accept data in the form nnn-nnnn OR nnnnnnn (n is any number); here the "-" is optional on entry and is not stored.

A **social security number** may be entered via a mask of "MR%3-%2-%4"

Other examples follow:

Value entered	Mask	Value stored	
123-4567	MR%3-%4	1234567	
1234567	" "	1234567	
12345X	" "	**Rejected**	
12345	" "	12345	(see note below)
123-4567	MV%3-%4	1234567	
1234567	" "	1234567	
12345	" "	**Rejected**	
ABC.100	ML&3.%%%	ABC100	
AB100	" "	**Rejected**	
ABC9	" "	ABC9	(see note below)

NOTE: when the value entered is SHORTER than the mask used and the justification is NOT "V", it will be accepted if it "satisfies" the mask from the left or the right, depending on the justification. If a "V" is used, the value must match the mask exactly.

d. **Translate** - A File translate is used to verify and Convert data, and must be one of the forms:

Tfilename;X;iamc;oamc or Tfilename;C;iamc;oamc

only (that is, the codes V, I and O are not allowed). If X is used, a forced verification occurs, that is, if the translated field or item is missing, the data entry will not be accepted. If C is used, a missing value will not cause an error; the untranslated data entry will be stored.

e. **Pattern Matching** - the "P..." code may be used to perform pattern matching beyond that allowed by Format masks.

Example: if a code must begin with the letter "A" or "B" followed by 3 numbers, the Pattern match code would be "P('A'3N);('B'3N)" (note multiple patterns may be specified).

f. **Range Checking** - the "R..." code may be used to force numbers to be in one or more ranges of values.

Example: to ensure that a value is in the range 1-100, the code "R1;100" may be used. **Remember to allow for decimal Conversions**; if the value had two decimals and was entered using a "MR2" decimal mask, you must use "R100;10000".

g. **Time** - Any Time Conversion such as "MT", "MTHS" etc.

h. **Mask Character** Conversions - The "MCU", "MCL" and "MCT" Conversions may be specified, and are used as Output (display) Conversions only; they **do not affect** data storage.

Note that other Conversion codes such as "MX", "MP" etc. are invalid, and will be rejected during Dictionary validation.

3.8 CORR or correlative codes

The correlative field (V/CORR) in an UPDATE Dictionary definition is identical to that in a RECALL definition, except that only Functions and Group extracts may be used. **Multiple Correlative codes are not allowed.**

a. **Group extracts** are used to split fields which have been stored in a concatenated manner. Thus more than one field can be stored within the same data attribute; this is mostly useful to generate a complex item-id. The format of a Group code is:

G n c m

where "n" is the number of segments to skip;
"c" is the character separator;
"m" is the segment copy count, and **must be 1**.

A given set of Group codes referencing the same attribute must use the same separator chararacter, and all segments of the value must be specified (that is, segments cannot be skipped over or left undefined).

Note that each Dictionary attribute has its own Conversion codes for data verification and display, its own external and internal data links, etc.; each segment is indeed a separate entity. Segments of a value may be specified as Controlling or Dependent via Structure codes.

Example: three fields which store data in the form "xxxx*yyyy*zzzz" would use three Dictionary definitions, the first of which would have a Group code of "G0*1"; the second "G1*1" and the third "G2*1".

b. **Function codes** may be "A" (Algebraic format) or "F" format. The value generated by the Function code is stored in the primary file item if there is a non-zero value in the amc; if the amc is zero, the generated value is not stored in the primary item, but is used only for display purposes or to be stored in a secondary file via a file-link.

Generated fields may be used just for data display, like RECALL definitions with Functions. The difference is that a Generated field is automatically recomputed (and redisplayed if necessary) whenever any of the values that affect it are changed. A RECALL definition is only recomputed under the same condition if you explicitly request it by setting up entries in the V/LINK of the fields that affect it (see later)

Example: If the values of PRICE and QUANTITY are to be multiplied so that the result displays and stores in a secondary file, an UPDATE Dictionary attribute would use the Correlative:

```
A;N(PRICE)*N(QUANTITY)
```

If PRICE and QUANTITY were multivalued, their sum may be generated by using the Funtion:

```
A;S(N(PRICE)*N(QUANTITY))
```

3.9 JUSTIFICATION

The justification field (V/TYPE) in an UPDATE Dictionary definition is identical to that in a RECALL definition.

Codes of "L", "T" and "U" are treated as Left-justified; "R" is Right-justified. In all cases, values longer than the defined field length will be truncated.

If a format mask is specified, its justification code will override the V/TYPE.

3.10 MAXIMUM LENGTH field

The maximum length field (V/MAX) in an UPDATE Dictionary definition is identical to that in a RECALL definition.

This value is used to limit the data entry length.

If a format mask is specified, its length will override the V/MAX specified on display.

Example: if V/MAX is 12, and the format mask is "MR2,$#12", the displayed field is 12 characters, and 12 will be accepted on input, which is not correct, and the display will be truncated. Therefore, a V/MAX of 7 is probably better - that is, a maximum entry of 9999.99 (7 characters) will redisplay as "$9,999.99" (9 characters). An entry of 9999999 (7 characters) will redisplay as "9,999,999.00" (12 characters). Note even in this case, the "$" is lost on output. An appropriate Range check should be used if entry values are to be limited.

3.11 MINIMUM LENGTH field

The minimum length field (V/MIN) in an UPDATE Dictionary definition is used to force a minimum data entry. A null or zero V/MIN indicates that there is no minimum length limit.

If the value entered, after masking, is shorter than V/MIN, the data entry is rejected with a message. Note that a non-zero V/MIN does not force the field to be a "Required field"; the V/REQD entry does that.

3.12 REQUIRED FIELD flag

The required field flag (V/REQD) in an UPDATE Dictionary definition is used to prevent a null entry from being accepted.

A value of "R" indicates that this field cannot be set to null.

If a Controlling field is set up as a Required field, at least one controlling value must be entered.

If a Dependent field is set up as a Required field, a value must exist for each corresponding controlling value.

3.13 PRE EXT SUB or pre-data-entry subroutine

This field in an UPDATE Dictionary definition is used differently in the IDO definition than in non-ID definitions.

a. In the IDO definition, an entry in this field is used to call a BASIC subroutine just after the primary item has been read in. The subroutine is called even if the item was not on file.

   It is acceptable to read or write other files in this subroutine, but maintaining the integrity of the database is then the programmer's responsibility.

b. On non-ID definitions, an entry in this field is used to call a BASIC subroutine before data entry.

   The subroutine can determine whether to allow the update of the field or not, depending on special circumstances.

   It is recommended that no other files be written via this subroutine call. This is because there are conditions when the data in this field may change, but the subroutine is not called. For example, if this field is a "dependent" and its "controlling" value is deleted, the appropriate dependent multi-value is automatically deleted also, but since there was no data entry to this field, its subroutine is not called.

Subroutine interface is described later.

The subroutine must be cataloged in the user's Master Dictionary.

3.14 POST EXT SUB or post-data-entry subroutine

This field in an UPDATE Dictionary definition is used differently in the IDO definition than in non-ID definitions.

a. In the IDO definition, an entry in this field is used to call a BASIC subroutine just before the primary item is to be written, deleted, or voided.

   It is acceptable to read or write other files in this subroutine, but maintaining the integrity of the database is then the programmer's responsibility.

b. On non-ID definitions, an entry in this field is used to call a BASIC subroutine after data entry.

   Note that the subroutine call only takes place if there was data entry; a "Return", cursor-control, or "Trap" character entered will not call the subroutine.

   The subroutine can perform additional checks on the validity of the entered data, and accept or reject the entry.

   It is recommended that no other files be written via this subroutine call. This is because there are conditions when the data in this field may change, but the subroutine is not called. For example, if this field is a "dependent" and its "controlling" value is deleted, the appropriate dependent multi-value is automatically deleted also, but since there was no data entry to this field, its subroutine is not called.

Subroutine interface is described later.

The subroutine must be cataloged in the user's Master Dictionary.

3.15 DEFAULT VALUE or automatic value

The default field (V/DEFAULT) in an UPDATE Dictionary definition is used differently in the ID Dictionary definitions than in all other definitions.

a. In the ID definitions (that is, in ID0, ID1, etc.), the purpose of the V/DEFAULT is to automatically generate a new numeric value. This automatic value is maintained in attribute two of a special item in the same Dictionary. The item-id of this special item is stored in the V/DEFAULT field.

   If V/DEFAULT is set up in this way, remember to edit and file the special item into the Dictionary.

   > To get the next automatic value when the cursor is positioned on the item-id request, a LINEFEED (or Control-J character) causes the next value to be generated.

   Example: The ID0 Dictionary definition in the CUSTOMER file contains the value "NEXT.CUST" in its V/DEFAULT field. The item NEXT.CUST is intially stored in the same Dictionary, with attribute 2 set to "1000".

   When the automatic value is first requested, a value of "1000" is returned, and attribute two of NEXT.CUST reset to "1001". The next time, the returned value is "1001" and NEXT.CUST is reset to "1002", etc.

b. In all UPDATE Dictionary definitions other than the ID definitions, V/DEFAULT is used to generate an "automatic" or "default" value on a one-time basis when a field is first referenced. This happens on new records as each field is reached, and additionally on dependent fields whenever a new controlling value is added.

   An automatic value may be overridden by the operator.

   The V/DEFAULT field contains an "A" (Algebraic) or "F" code, just like the Correlative field.

   Example: In order to generate today's date automatically, the following function may be used:

   ```
   A;D
   ```

   Another example: If, after entering a customer number into the primary file's attribute number 3, an address field is to be obtained from the CUSTOMER file's attribute 7, the V/DEFAULT would be:

   ```
   A;3(TCUSTOMER;C;;7)
   ```

3.16 HELP MSG - help message

This field in an UPDATE Dictionary definition is used to assist the data entry operator by printing a "help" messsage on the screen.

The text of the message can be up to 65 characters long, and it is printed whenever the "help" trap character, "?", is entered by the operator.

If additional assistance is needed, a BASIC subroutine may be called (see the next section).

3.17 MORE HELP SUB external subroutine

This field is used to call a BASIC subroutine when the data entry operator requires more assistance than provided for by the V/HELP message.

After the above message is printed, if there is a value in this field, the message:

```
? for more
```

will display; the operator can now enter another "?" to invoke the BASIC subroutine.

Typically, the subroutine may be used to Execute a RECALL statement to list data from a file.

Subroutine interface is described in a separate section.

3.18 EXTERNAL STRUCTURES - secondary file links

The external structures field (F/LINKS) in an UPDATE Dictionary definition is a multi-valued field that contains links to secondary files.

Secondary file items are locked when read, to ensure database integrity in a multi-user environment.

Secondary file items are written when 1) the secondary item-id is changed; 2) (if the secondary item-id is defined from a window) when the cursor moves off the window line; or 3) when the primary item is written or deleted. If any secondary item has been written, voiding of the primary item update is inhibited and the BREAK key disabled, until the update is completed by either writing or deleting the primary item.

Up to nine secondary files may be defined.

Each link consists of three parts:

1. The secondary file-name.
2. A storage code.
3. The name of the Dictionary attribute (in the secondary file) in which to store the data, unless this is a link to the item-id of the secondary file, in which case this value is null.

a. Generating a link to the secondary file's item-id.

There must be a link that generates the item-id of the secondary file from some field in the primary file. UPD-VALIDATE will ensure that such a link exists, and that it conforms to the definition of the item-id of the secondary file.

A link to the secondary item-id has a storage code of "ID".

The field that links to the secondary item-id may be single-valued, multivalued, a controlling or a dependent value, or the item-id of the primary file.

If the secondary file's item-id is composed of concatenated segments, it is necessary to define a "generated field" in the primary file which generates the secondary item-id. The Dictionary definition of the generated field must contain the link, while the definitions of the individual fields in the primary file that compose the secondary item-id must not have any links to the secondary file (see example later).

b. Storing data in fields of the secondary file.

Each field in the primary file that stores data in a secondary file's attribute must have a link.

The field that links to the secondary attribute may be single-valued, multivalued, a controlling or a dependent value, or the item-id of the primary file.

The storage code defines the method used to transfer the data from the primary file to the secondary file.

Storage code	Meaning
SV	Copies the value as a Single Value to the secondary file's attribute.
A	Adds the value to the secondary file attribute, which must be singlevalued and numeric.
S	Subtracts the value from the secondary file attribute, which must be singlevalued and numeric.
MV	Copies the value as a Multiple Value to the secondary file's attribute. This code also deletes the appropriate multivalue in the secondary attribute when the primary value is changed or deleted.
MA	Copies the value as a Multiple Value to the secondary file's attribute. This code does not delete the appropriate multivalue in the secondary attribute when the primary value is changed or deleted. The main purpose of this code as opposed to the MV is when several primary file items may update the same multivalue ("n-to-1 relationship), and when one of these primary values is deleted or changed, you do not want the secondary to disappear. Use this code with care, since it tends to leave "footprints" in secondary files.

Restrictions:

a. You cannot store into a field of the secondary file that is defined as a concatenated value.

b. You cannot store into a Controlling-Dependent relationship in the secondary file.

UPD-VALIDATE will attempt to ensure that the "from" and "to" values match; you cannot add into a Date secondary field, for example.

It is best to set up UPDATE Dictionary definitions for all related files at one time and run UPD-VALIDATE against them all. If any changes are made in a Dictionary that has links to or from other files, they should all be re-validated; this is the responsibility of the applications designer since it is not forced.

Examples:

1. The primary file is ORDER; the secondary file is BOOKING.

   Action required: every time an order is assigned a Customer number, the BOOKING file is updated with an item whose item-id is the Customer number. Also, the item-id of the ORDER is stored in the ORDER# field of the BOOKING file. Since there may be more than one order per customer, the ORDER# field is a multivalued field.

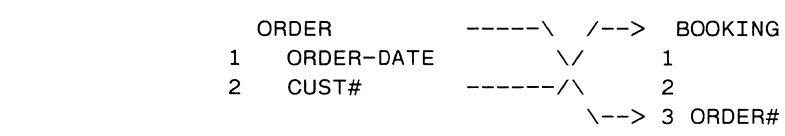
   To establish the link to the BOOKING file's item-id, the F/LINK value in the CUST# definition in the ORDER Dictionary is:

```
BOOKING    ID       (no attribute name)
```

   To establish the data storage link, the F/LINK value in the IDO definition of the ORDER Dictionary is:

```
BOOKING    MV       ORDER#
```

   The file layout is :



   Data base integrity is automatically maintained under all conditions, even if the ORDER item is deleted, or the customer number in the order is changed, etc.

2. Modifying the above, instead of just the customer number, we want to use a concatenation of the CUST# and ORDER-DATE to form the item-id in the BOOKING file.

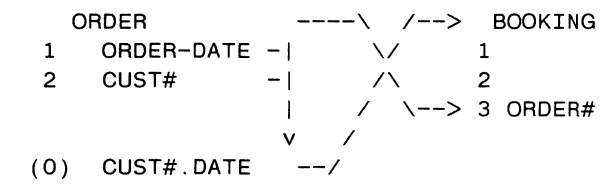
   Since the secondary file's item-id is concatenated, we need to create a "generated field" as mentioned earlier.

   If the generated field is called CUST#.DATE, it will be defined in the Dictionary of the ORDER file with an AMC of zero, and a Correlative of A;N(CUST#):'*':N(ORDER-DATE). Note that Dictionary definitions CUST# and ORDER-DATE do not need F/LINKs; however, CUST#.DATE must have an F/LINK of:

```
BOOKING    ID
```

   (The F/LINK of the IDO item in the ORDER dictionary is the same as before).

The file layout is now:



3. In the same files, let us now assume that there is a Controlling-Dependent relationship; the multivalued PART# field controls the QTY-ORD field.

Action required: as parts are ordered, the quantity ordered is to be subtracted from the ON-HAND field in the INVENTORY file.

Note that the secondary file (INVENTORY) is updated as each line in the window is modified, added or deleted.

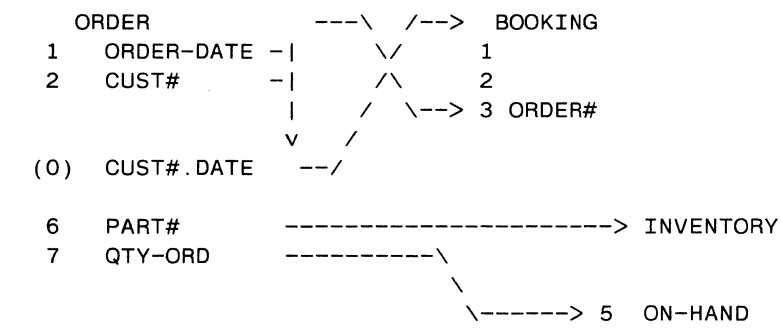
First establish the link to the item-id of the INVENTORY file, which is the PART# in the ORDER file. The F/LINK value in the PART# definition of the ORDER Dictionary is:

```
INVENTORY     ID
```

Next establish the quantity link; the F/LINK value in the QTY-ORD definition of the ORDER Dictionary is:

```
INVENTORY     S     ON-HAND
```

The file layout is now



3.19 VALUE LINKS

The value links field (V/LINKS) in an UPDATE Dictionary definition is a multi-valued field that contains the names of any RECALL definitions affected by this field.

RECALL definitions (with an 'A' code) may be used to display data on the screen. If the RECALL definition has a Function that is affected by a value on the screen, the Function should be re-computed whenever a value is changed. The V/LINKS field ensures that the re-computation takes place.

If the computed value is from an UPDATE definition with a Function Correlative, there is no need to place its name in the V/LINKS field; the re-computation in this case takes place automatically.

Example: Assume that the RECALL definition MARGIN is defined as a Function of PRICE minus COST (that is, it has a Function such as "A;N(PRICE)-N(COST)", where PRICE and COST are UPDATE Dictionary definitions.

To ensure correct re-computation, the V/LINKS fields of both PRICE and COST should contain the value "MARGIN".

3.20 Subroutine calling

In general, the UPDATE processor can handle the majority of data validations and data manipulations. If non-standard processing is needed, a BASIC external subroutine may be called from a number of different areas of the UPDATE processor.

The subroutine must be cataloged in the user's Master Dictionary.

A subroutine may be called under the following conditions:

a. Just after reading in the item from the file (even if the item does not exist on file) - subroutine name is in "PRE EXTernal SUB" field of the IDO Dictionary definition.

b. Just before writing, deleting or voiding the item - subroutine name is in "POST EXTernal SUB" field of the IDO Dictionary definition.

c. Just before accepting data entry from the operator - Subroutine name is in "PRE EXTernal SUB" field of the Dictionary definition for the field.

d. After data entry from the operator has been accepted by any Masking and other Conversion validations, but before storing the data in the field - subroutine name is in "POST EXTernal SUB" field of the Dictionary definition for the field.

e. When the user requests "additional help" - Subroutine name is in "MORE HELP SUB" field of the Dictionary definition for the field.

It is not recommended that subroutines in categories (c), (d) and (e) perform file writes. See the warning under Dictionary description.

3.21 Subroutine interface

The external subroutine interface is the same for all the above conditions; the following must be the first two lines of the subroutine:

```
SUBROUTINE name
$INCLUDE SYSLIB UPD.COMMON
```

The "included" item contains the COMMON variables accessible by the subroutine. The C.ERRCODE variable allows the subroutine to determine the condition under which it has been called (see table later); the C.CLEARSCREEN variable may be set by the user to ONE to cause the screen to be refreshed on return from the subroutine.

Note - the UPDATE processor runs with a PRECISION of ZERO, which is set by the included item UPD.COMMON. The subroutine cannot have a different precision.

3.22 COMMON variables

Warning - Variables marked with an "*" should not be altered by the user subroutine. All COMMON variables from UPDATE begin with "C."

C.DICTPRIMFILE	*	File variable of dictionary of primary file.	
C.PRIMFILE	*	File variable of data section of primary file.	
C.ITEMID	*	Item id of primary file.	
C.ITEM(100)		Primary item.	
C.ITEM.EXISTS	*	0 for NEW items, 1 for existing items.	
C.NO.VOIDING	*	0 if no secondary files items have been written; 1 if any have been written.	
C.AMC	*	Current field attribute number.	|
C.VMC	*	Current field multivalue number.	|
C.OLD.DATA	*	Original value from item (Note 1)	|
C.NEW.DATA		Operator-entered value (Note 1)	|
C.X	*	x cursor location.	| Meaningful
C.Y	*	y cursor location.	| for "Before data
C.VCONV	*	Output conversion code (may be used in OCONV).	| entry" and "After data entry"
C.RCONV	*	Input conversion code (may be used in ICONV).	| subroutines only.
C.MASK	*	Output "mask" code.	|
C.GLOBALF	*	"Options" from UPDATE sentence.	
C.PRIMFILENAME	*	Name of primary file.	
C.DEBUG		If non-zero, UPDATE debugger is in effect.	
C.ERRCODE		See table in next section.	
C.CLEARSCREEN		This variable is zero on entry; the user subroutine may set it to one if the screen is to be refreshed on return to UPDATE.	
C.USER(15)		User-definable for parameter passing. This array is set to null on initial execution of UPDATE.	
C.DELETE.OPT	*	1 if the statement allows item deletion (X option)	
C.VIEW.OPT	*	1 if the statement is inquiry (V option); Also set to 1 when editing an EXISTING item and the Nomodify (M) option is in effect.	
C.ITEM.CHANGED	*	1 if Primary item has been modified since read.	

Note 1: Value is in "internal format"; that is, before Output Conversion for OLD.DATA, and after Input Conversion for NEW.DATA.

3.23 C.ERRCODE variable I/O values

When called	Value on entry to subroutine	Value on exit (set by user)
Just after reading item	0	0 = Accept item. 1 = Reject item (don't allow any updates).
Just before writing item	0 = Writing item 1 = Deleting item 3 = Voiding item	0 = Accept action. 1 = Reject action (return to "Edit line"). 3 = Void all updates (see Note 1).
Before data entry	0	0 = Accept data entry. 1 = Disallow data entry (skip this field).
After data entry	0	0 = Accept and store data. 1 = Reject data entry (repeat data request).
Help request	No meaning	No meaning

Note 1: By setting C.ERRCODE to 3, all updates are voided (as in EXit command in the EDITOR). If the C.NO.VOIDING flag has a value of one, this will not be accepted because at least one secondary file has been written, and database integrity would be lost if the void is allowed. The user may change C.NO.VOIDING to zero at his own risk to force acceptance of a void command. Maintenance of database integrity is then the user's responsibility.

3.24 Examples of user subroutines

3.24.1 Conditional updating

A field can be accessed for change only if attribute 1 of the item contains an "A"; no error message is needed. The following subroutine can be called from the PRE EXT SUB field:

```
SUBROUTINE ACODE.PRE
$INCLUDE SYSLIB UPD.COMMON
  C.ERRCODE = ( C.ITEM(1) = 'A' )
RETURN
```

3.24.2 Complex validation

A set of "From" and "To" quantities is to be entered into a multi-valued discount schedule. For example:

```
From      To
  1       25
 26      100
101      500
```

We need to ensure that the "To" quantity is greater than the "From" quantity, and less than the next "From" quantity. The following subroutine can be called from the POST EXT SUB field of the "To" attribute (assume the "From" attribute number is one less than the "To" attribute number):

```
SUBROUTINE TO.POST
$INCLUDE SYSLIB UPD.COMMON
IF C.NEW.DATA < C.ITEM(C.AMC-1)<1,C.VMC> THEN
   PRINTERR 'To quantity must be > From quantity'; C.ERRCODE = 1
END ELSE
   NEXT.FROM = C.ITEM(C.AMC-1)<1,C.VMC+1>
   IF NEXT.FROM # ''  & NEXT.FROM < C.NEW.DATA THEN
      PRINTERR 'To quantity must be < Next From quantity'
      C.ERRCODE = 1
    END
 END
RETURN
```

A similar subroutine is needed to validate the "From" values.

3.24.3 An extended "Help" routine

The following subroutine may be used to help the operator find a customer number:

```
SUBROUTINE CUSTNO.HELP
$INCLUDE SYSLIB UPD.COMMON
PRINTERR 'Enter customer name or portion'
INPUT SEARCH: THEN
   EXEC 'LIST CUSTOMER-MASTER WITH NAME "[' : SEARCH : ']" (C)'
   PRINT 'Type return to continue':; INPUT SEARCH,1:
   C.CLEARSCREEN = 1
END
RETURN
```

CHAPTER 4

DATA ENTRY

4.1 Modes of data entry

The UPDATE processor first requests the item-id or key of the file. A null entry to this terminates the UPDATE statement.

After entering the item-id, the UPDATE processor operates in two distinct modes of data entry, INPUT and EDIT.

4.1.1 The INPUT mode

The **Input mode** is entered automatically when the key of a NEW item is entered by the operator.

In this mode, the UPDATE processor steps sequentially through the fields as they appear in the UPDATE statement. After each field is entered and validated, the cursor is positioned at the next logical field.

In the case of Associated data sets, if the Controlling value is entered, data entry will be requested for all corresponding Dependent values. If a null is entered at the Controlling value, this section of data entry is terminated.

Backward cursor positioning is allowed during this phase; this allows the operator to correct mistakes even after the field or fields have been entered. The section on Cursor Controls describes this.

4.1.2 The EDIT mode

The **Edit mode** is entered:

1. After all fields have been entered in the INPUT mode for a NEW item, or

2. When an EXISTING item is being updated.

In this mode of operation, control passes back and forth between the EDIT request line and data entry. The EDIT request is a line, typically at line 22 on the screen, of the form:

```
Enter field id to change, * (EXit), RETURN (FIle)
```

The above message is the default generated by the UPDATE processor; a different message may be specified by using the FOOTING connective in the UPDATE statement.

A field id of the forms:

n n.m a a.m

may be entered; "n" is field number; "n.m" is a specific line "m" in a window whose field number is "n"; the "a" forms are used if alphabetic field-ids are used.

In addition, the Editor-like commands 'FI', 'EX' and 'FD' may be entered to File the primary item, Exit without filing, or Delete the primary item. An 'EXT' returns to TCL even if a select list is being used.

The 'DEBUG' command may be entered to turn on the UPDATE Debugger.

4.2 Paging of data within windows

When a window is initially setup by the UPDATE processor, values are stored starting at the first multi-value on file until such values are exhausted or the window is filled. Data in the window may be "paged" in several ways:

1. If the cursor is positioned on the ending multi-value of any field and the LINEFEED key is entered to page down, or on the beginning multi-value and the Cursor-Up or Backspace key is entered to page up.

2. If a specific multi-value is requested from the EDIT request line via the 'n.m' command, where 'm' is the sequence number. For example, entering '7.11' will position the cursor on the 11-th. multi-value of field #7; the window data will be paged up or down if that field was not on display.

4.3 Cursor movement

The movement of the cursor from field to field on the screen is made as simple as possible.

4.3.1 The RETURN key

The **RETURN** key is used as the most frequent termination of data input or to move from one field to another. A RETURN without any data input preceding will maintain the current field value (if any) with no change.

4.3.2 The LINEFEED key

The **LINEFEED** key (equivalent to the Cursor Down key if this exists) is equivalent to the RETURN key if the cursor is on a single-valued field. It has a different action only when within an Associated data set, or when on the item-id field(s).

If **within a window**, linefeed moves the cursor down to the next corresponding multi-value without changing fields (that is, it causes a downward motion of the cursor).

If the cursor had been postioned on the last value of the field as displayed, and there are more values on file, the entire window will "roll up" one value to display the new value. If there are no more values on file, the LINEFEED is ignored.

If on an **item-id field**, linefeed invokes the "get next automatic id" feature if this has been defined in the dictionary

4.3.3 The Up-arrow or Caret key

The Up-arrow or Caret key (equivalent to the Cursor Up key if this exists) is equivalent to the RETURN key if the cursor is on a single-valued field. It has a different action only when within an Associated data set. It then moves the cursor up to the previous corresponding multi-value without changing fields (that is, it causes a upward motion of the cursor).

If the cursor had been postioned on the first value of the field as displayed, and there are previous values on file, the entire window will "roll down" one value to display the new value. If there are no previous values on file, the Cursor Up is ignored.

4.3.4 The BACKSPACE key

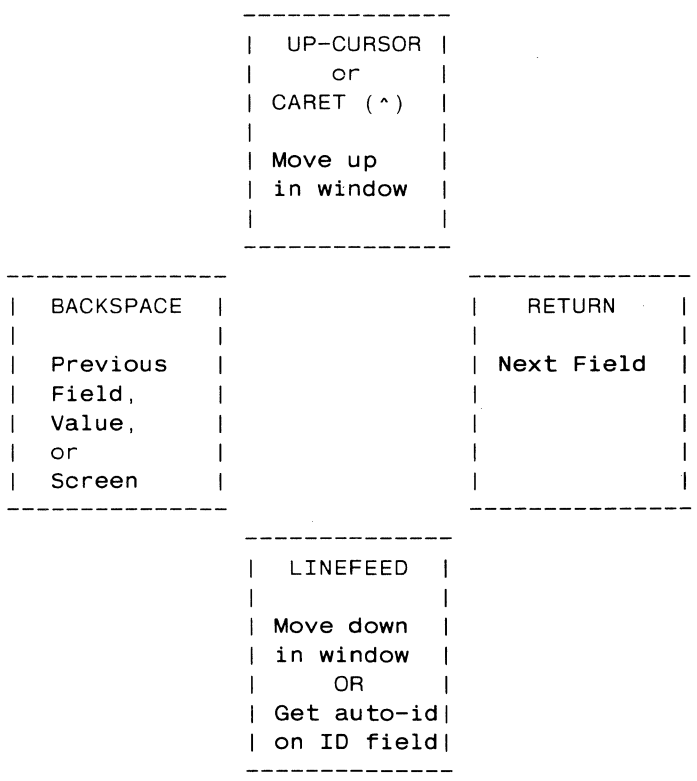
The **BACKSPACE** key (or Cursor Back if it exists) is used normally to erase the last input character.

If, however, the BACKSPACE is entered when the cursor is positioned **on the first character of the field**, it causes the cursor to back up to

the previous logical field, if on a data field; to get the previous item-id if on the item-id field; or to go back to the last field edited if on the Edit line. If the cursor was positioned on the first field of the screen, no action takes place.

BACKSPACE is also used to go to the previous screen from the Edit line when multiple screens are used.

4.4 Summary of cursor movement



4.5 Trap characters

The following are special or "trap" characters that cause the UPDATE processor to take the specified action.

The column headed "location" indicates where the action takes place - when data is entered to a field, or at the "Edit line".

Character	Location	Action
-	Edit line	DELETE entire item; (requires confirmation from operator). 'FD' is an equivalent entry.
	Data entry	DELETE value if not a Required field; if a Controlling value is deleted, all associated Dependent values are also deleted; that is, the window line is deleted.
?	Anywhere	Print HELP message; request if HELP subroutine is to be called, if any.
*	Edit line	VOID updates; accepted only if nothing has been written to a secondary file. 'EX' is an equivalent command.
	Data entry	SKIP field or window line.
!	Anywhere	REFRESH screen.
+	Data entry only while on controlling field within a window	INSERT a new line in the window.
Escape	Data entry	Enters the WORD PROCESSING mode (valid for left justified fields only); only the D (delete character), I (insert text) and R (replace text) are implemented; use RETURN to terminate input/replace text. Control-X quits Word processing mode without making any changes.

CHAPTER 5

UTILITIES

5.1 UPD-DEF utility

You must use UPD-DEF instead of the system's EDITOR to edit UPDATE Dictionary definitions. If any changes are made to a Dictionary definition, UPD-DEF forces re-validation of the Dictionary for safety.

To call UPD-DEF, enter at TCL:

```
UPD-DEF filename
```

You are now in the UPDATE processor, and may enter an item-id to edit. Help messages (using ?), cursor controls, etc. are now in effect.

5.2 UPD-VALIDATE utility

You must run UPD-VALIDATE to validate the Dictionary definitions before using UPDATE, and every time you change any Dictionary items via UPD-DEF.

To validate a Dictionary, enter at TCL:

```
UPD-VALIDATE filename   {(P)}
```

where the optional (P) is used to route messages to the printer.

If validation errors are found, an error listing is generated (see next to generate the same listing).

5.3 LIST-UERRORS

Errors detected by UPD-VALIDATE can be listed by entering at TCL:

```
LIST-UERRORS  filename  {(P)}
```

ULT
THE ULTIMATE CORP.
717 RIDGEDALE AVENUE, EAST HANOVER, NEW JERSEY 07936
(201) 887-9222 TWX 710-996-5862 Telecopier (201) 887-6139