# PREFACE

**The Programmer's Helper** (TPH) is a set of programs designed to aid in the development of BASIC programs under the PICK (Tm) Operating Systems. The PICK systems are comprised of the following:

PICK R83 (also known as Classic PICK)
Advanced PICK (Open Architecture)
ULTIMATE
REALITY
REVELATION
UNIVERSE
PRIME INFORMATION

Since varied names are used for the different types of BASIC language found on the above-mentioned implementations and emulations of the PICK Operating System, the term "PICK/BASIC" or "BASIC" shall be assumed in reference to the language name found on your particular system.

The advantages of using a program, or code, generator are many. The primary advantage is, of course, greatly increased programmer productivity. A program can be prepared utilizing far fewer programmer hours for actual programming and debugging. But there are many more benefits besides that of increased productivity. It allows beginning or unskilled programmers to reach levels of proficiency deemed acceptable far more quickly. It also provides these individuals with positive examples of correct programming practices and standards (i.e. - structured coding style, descriptive variable names, system consistency throughout an application, etc.).

A major benefit obtained by the use of program generators is the reduced expense of maintenance programming. This is the type of programming which requires a programmer to go back into programs that are already in production and modify them to meet new criteria or requirements. Program maintenance probably accounts for 80 percent of a company's data processing expenditures.

But perhaps, one of the most overlooked benefits of program generators is that they allow the programmer to spend much more time doing the critical analysis work. In other words, a programmer's time can be spent more effectively on data base design and normalization, understanding

and documenting user requirements, and writing full specifications for s system before any line of code is written. This can be accomplished because the programming tool frees up so much of the programmer's actual time.

The only prerequisite to using **The Programmer's Helper** is a basic understanding of the PICK Operating System's file structure, the use of dictionaries to define file data, a rudimentary knowledge of the PICK/BASIC programming language, and a willingness and desire to learn.

After the programmer designs the file layout and creates the necessary dictionary items to define that file (this step is omitted if the file already existed and was in use), he supplies **The Programmer's Helper** with a brief description of a data entry screen or report format. **The Programmer's Helper** will generate complete, functioning, and self-documenting BASIC programs. The resulting generated program can then be customized to meet any needs that may be beyond the scope of the generator. The result is a finished program completed in a quicker manner than if it had been written from scratch.

Programs generated by **The Programmer's Helper** have the added benefit of being written in an easy to read, highly structured, style. Unlike many of the other code generators, **The Programmer's Helper** does not produce "Spaghetti" coded programs (programs that are unstructured, hard to read, and inefficient). The generated code is very compact and efficient, and makes no use of the **GOTO** or **RETURN TO** statements. Variable name usage in the generated program is derived from the data dictionary definitions making the program self-documenting. When needed, subroutines (external BASIC programs) are used for modularity and ease of maintenance.

A last benefit of **The Programmer's Helper** is that the programs generated can run "stand alone". This means that there is no "run-time" to purchase, or no complex series of subroutines to include with your programs. Programs may be moved from one PICK based machine to another with a minimum of effort. And since the code is written in a very generic style, there should never be compatibility problems between systems.

But **The Programmer's Helper** does more than just write BASIC programs. A report definition can be converted into an ACCESS statement for the preparation of simple reports. Plus, a forms generation utility is also included, which allows for the design and printing of formatted reports (useful for checks, invoices, sales orders, and any other kind of pre-printed forms). Users that are intimidated by ACCESS can use the reporting function to painlessly design and develop reports with little or no knowledge of ACCESS.

**CONGRATULATIONS!** By purchasing **The Programmer's Helper** you are on your way to a more productive career. We hope that the results you obtain with **THE PROGRAMMER'S HELPER** are as rewarding to you as they were to most of our customers.

# 1 The Menu System

## 1.1 Introduction

**The Programmer's Helper** is comprised of five major segments, or tools. These are:

a. Data Entry Programs

b. Report Programs

c. Forms Processing Programs

d. Menu Building Tools

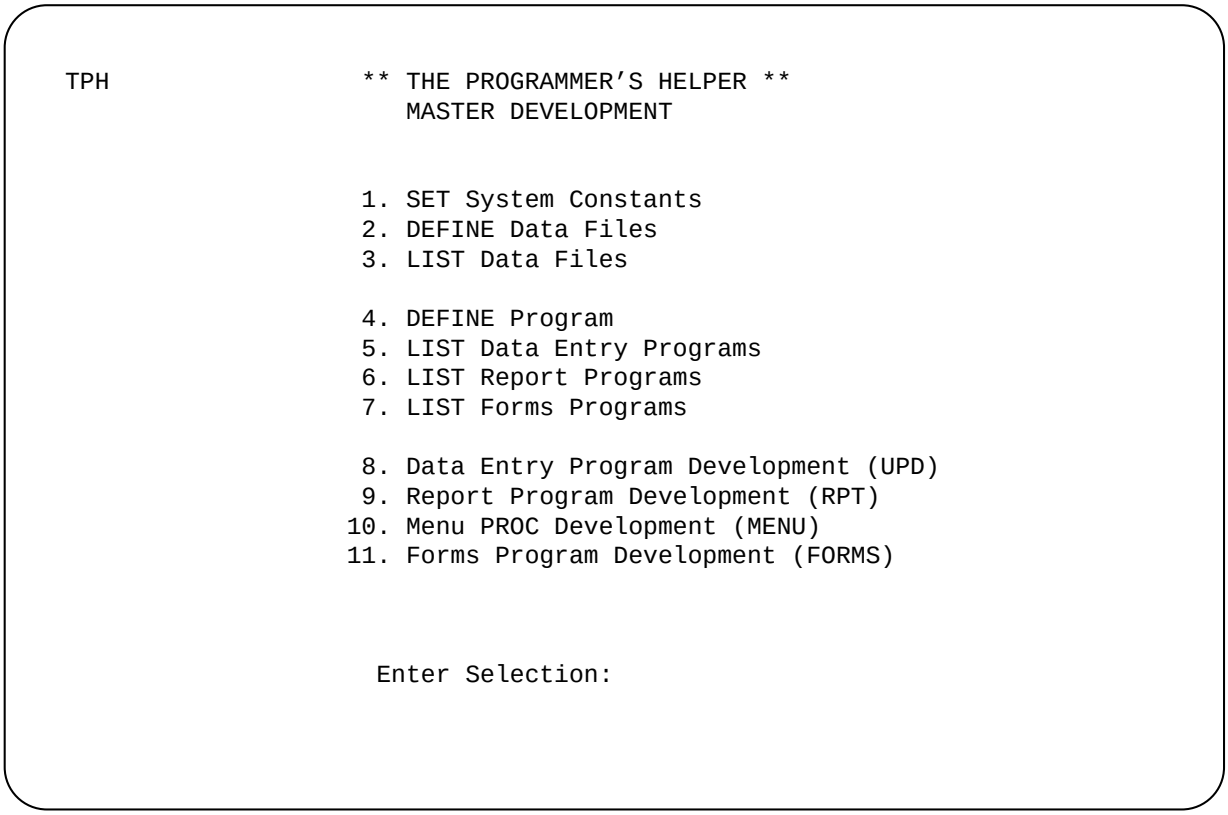
e. Index (B-TREE) Building Tools

The first segment (a) deals with the design and preparation of data entry screen programs. The second segment (b) is concerned with preparing report formats. The third segment (c) deals with the creation of formatted forms processing programs. The fourth segment (d) is used to build user, or application, menus. The Menu Building module allows you to tie together all of the other segments so that the user sees a coherent system and can interact intelligently with it. The final segment (e) allows the user to build and maintain index files that can be used to speed searching.

The following chart shows the chapter in which the specific segment information can be found. Note that there is also a Tutorial included in this manual to help your learning process:

| **To learn more about**... | **See:** |
|---|---|
| Data Entry Programs | Chapter 2 |
| Report Programs | Chapter 3 |
| FORMS Programs | Chapter 4 |
| Menu Building | Chapter 5 |
| Index Files | Chapter 6 |
| Tutorial | Chapter 7 |

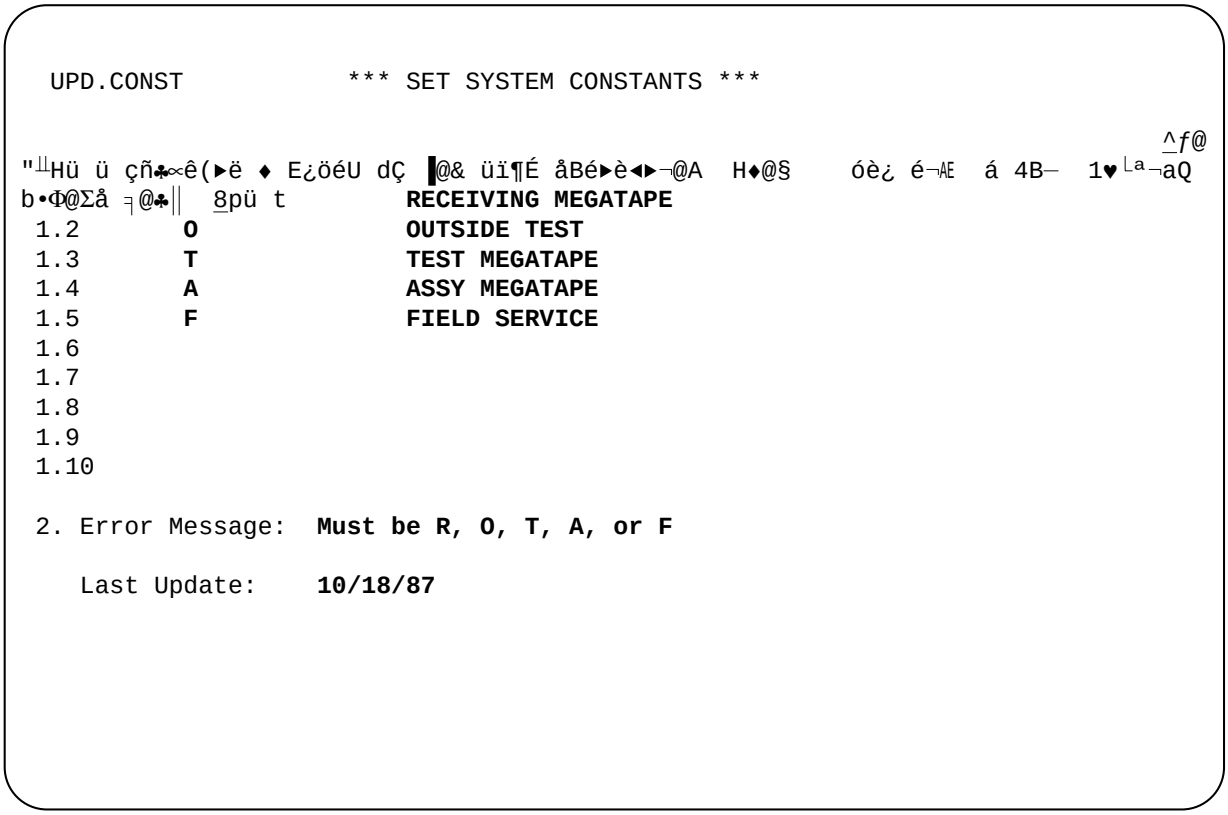
But before moving on to the next chapters, we will spend some time now discussing THE MASTER MENU (see figure 1). The 11 options shown here make up the kernel of **The Programmer's Helper**. Each of the eleven selections will be covered in detail, with respective screen images for your reference.




**The Programmer's Helper Main Menu**

## 1.2 Set System Constants

This function allows you to modify many of the system wide constants that are used and required by **The Programmer's Helper**. If, at any point, you are unsure of any of the answers to the questions on the **SET SYSTEM CONSTANTS** screen, please contact your dealer for further assistance.




**Set System Constants Screen**

Following is a description of each of the fields in the **Set System Constants** screen:

**Data Frame Size**

This refers to the number of character in each frame. This is ONLY used when **The Programmer's Helper** creates data files. For most **R83** versions of PICK, the frame size is 512 bytes. If you are using **Advanced PICK** (also known as Open Architecture), the frame size is 1024 bytes. For some other specific PICK systems such as CIE Systems and ADDS, the default frame size is 2048 bytes. If you are still unsure of the frame size being utilized by your particular system, check your PICK system documentation or contact your dealer for assistance.

**Default Program Indent**

This option allows you to state the number of spaces that you want to indent BASIC structures within a program (**FOR-NEXT**, **LOOP-REPEAT**, **BEGIN CASE-END CASE**, etc.). The value entered here will be the default spacing used when generating a program.

**Maximum Item Size**

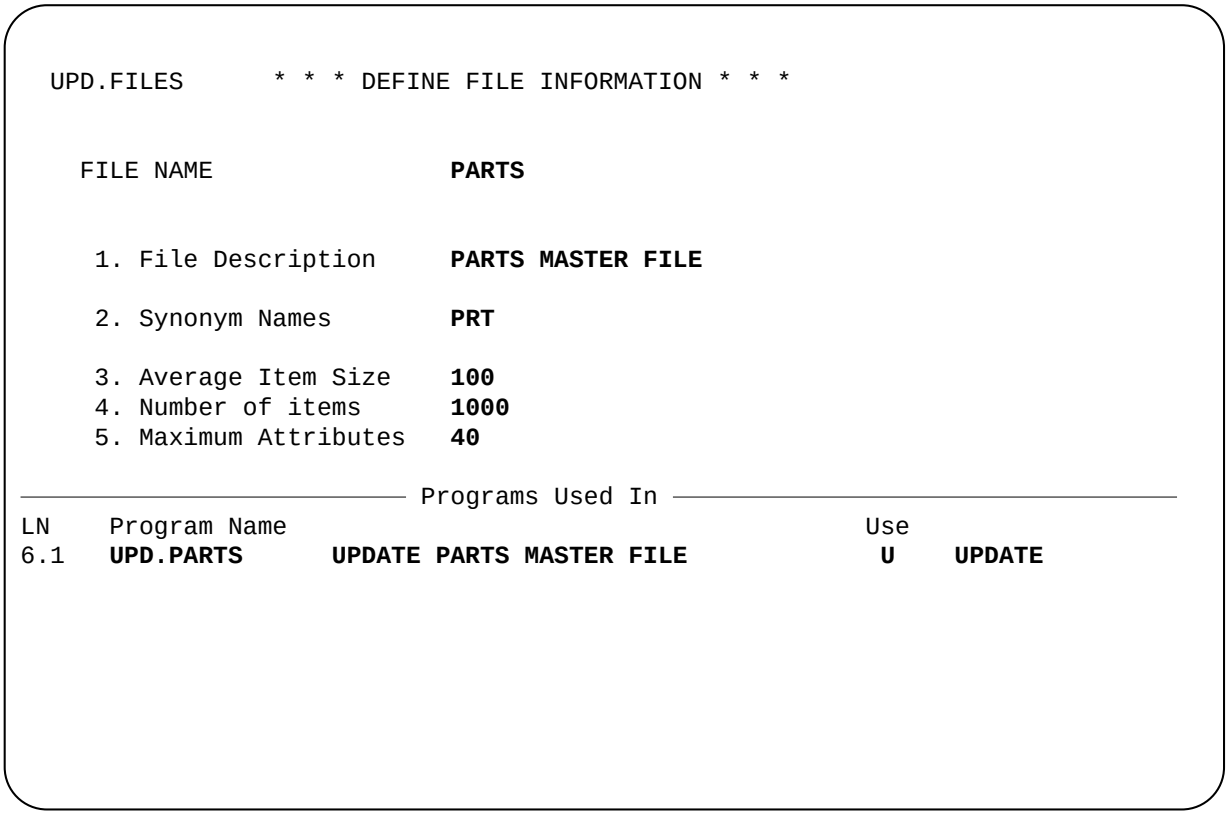
If a generated program exceeds the maximum item size, the portion of the program in excess will be written as follows:
The ID of the program is concatenated to an "*" which is then concatenated to a number counter such as "1", "2", etc. For example, a program named **UPD.CUSTOMER** would be written out as **UPD.CUSTOMER** and **UPD.CUSTOMER*1** if the item exceeded the maximum item size. For most PICK systems, the maximum item size is 32,677 bytes. If your system supports items of unlimited size then enter 99999 at this prompt. If you are unsure of how to fill in this field, please refer to your PICK system manual or call your dealer for assistance.

**INCLUDE/INSERT statement**

Enter the name of the PICK/BASIC statement used for merging other source code items into the program being compiled. Some systems use the command **INCLUDE** while others use **$INSERT**. If your system does not support either of these statements, then press ↵ "enter" to leave null.

## 1.3 Define Data Files

Before **The Programmer's Helper** can be used with a data file, you must first use the **DEFINE DATA FILES** screen to describe the file. This function is used to provide a cross-reference of files and programs in the automatic documentation feature.




**Define File Information Screen**

The **DEFINE FILE INFORMATION** screen is comprised of the following prompts:

**File Name** — At this prompt you enter the name of the file that you want to define. In our example, **PARTS** is the name of the file being used. (Note - the file will be created if it does not already exist)

**File Description** — Enter a brief description of the file that you are describing.

**Synonym Names** — Enter a short name that will help you in reference to the file name. In the sample screen, we use **PRT** as the file synonym name for **PARTS**. (Note - to avoid confusion, please remember to make the synonym different from any other file names that exist in the master dictionary of the account.)

| | |
|---|---|
| **Average Item Size** | This is an approximate figure used to help you size the file correctly. In our example, the average item size is 100. This means that the average record in our file will contain 100 bytes of information. |
| **Number Of Items** | Enter the approximate number of items that you think this file will hold. The example shows a file that will contain 1,000 records. |
| **Maximum Attributes** | This field prompts you for the total number of attributes, or fields, which will be used by this data file. Our example shows that 40 fields will be used by the records in this file. This number is used to DIMENSION the array that is used to MATREAD and MATWRITE the items in the file. |
| **Programs Used In** | This is a multivalued window in which you can enter all of the programs that make use of the file that you are defining. |

## 1.4 Define Programs

Before generating programs, you must tell **The Programmer's Helper** the name of the major data file to use as well as the name of the file in which the generated source code will be placed (known as the source file). The **DEFINE PROGRAMS** screen lets you define this information as well as program titles, program purpose, a list of subroutines called, as well as a list of other files accessed.

```
UPD.PD                  *** UPDATE PROGRAM DEFINITION ***

   Program Name:  UPD.PARTS
                                                        8.Indent:    5
1. Program Type:  ENTRY   DATA ENTRY PGM                9.# Screens: 2
2. Program Title: UPDATE PRICE-BOOK DATA               10.Use INSERT:N
3. Program Desc:  This program is used to update the PARTS master file.
                  You may add, delete or change part numbers.

4. Data File:     PARTS            PARTS MASTER FILE
5. Source File:   BP
   Pgms Called:   6.1   UPD.MODEL        UPDATE PRODUCT MODEL FILE
                  6.2   UPD.PRODLINE     UPDATE PRODUCT LINE FILE
                  6.3   DISPLAY.TABLES   DISPLAY/UPDATE LOOKUP TAB

-----------------------------Other Files---------------------------------
 LN    FILENAME         DESCRIPTION                 ITEM ID ATTR       USE
 7.1 MODELS           PRODUCT MODEL MASTER            MODEL-NO         X5
 7.2 PROD-LINE        PRODUCT LINE MASTER             PROD-LINE        R
 7.3 CUSTOMER         CUSTOMERS MASTER FIL
 7.4
```

**Update Program Definition Screen**

The **DEFINE PROGRAMS** screen consists of the following:

**PROGRAM NAME** — At this prompt, you enter the name of the program that you will later be generating.

**1. Program Type** — This field identifies the valid program types that one can generate. The valid program types are:

**ENTRY** for data entry screens
**REPORT** for reporting programs
**FORM** for forms printing programs.
**WINDOW** for "window" programs.

**2. Program Title** — This is the title that will appear centered at the top of your program's screen.

**3. Program Desc** This description is used to document both the source code in the program, as well as the automatically generated program documentation.

**4. Data File** This is the name of the major data file that is to be used by this program. (Note - **The Programmer's Helper** does not limit you to just one file being updated. More on this later)

**5. Source File** This field contains the name of the file where the source code will be written to.

**6. Programs Called** This field will be comprised of a multivalued list of all of the programs which will be used, or called, by this routine.

**7. Other Files** Here, you will place the names of all of the other files that will be referenced by this program. This is a multivalued window comprised of the following:

a. Filename - enter the actual name of the file being used.

b. Description - this is automatically pulled up from the file definition process.

c. Item Id Attr. - the item id dictionary definition is placed here for the particular file being used.

d. Attribute to use - if the "use" described below defines a total, then this is the attribute name used to update the total.

e. Use - this field is filled with either "R" for read, no updates, "U" for file updates, or "X" for cross references.

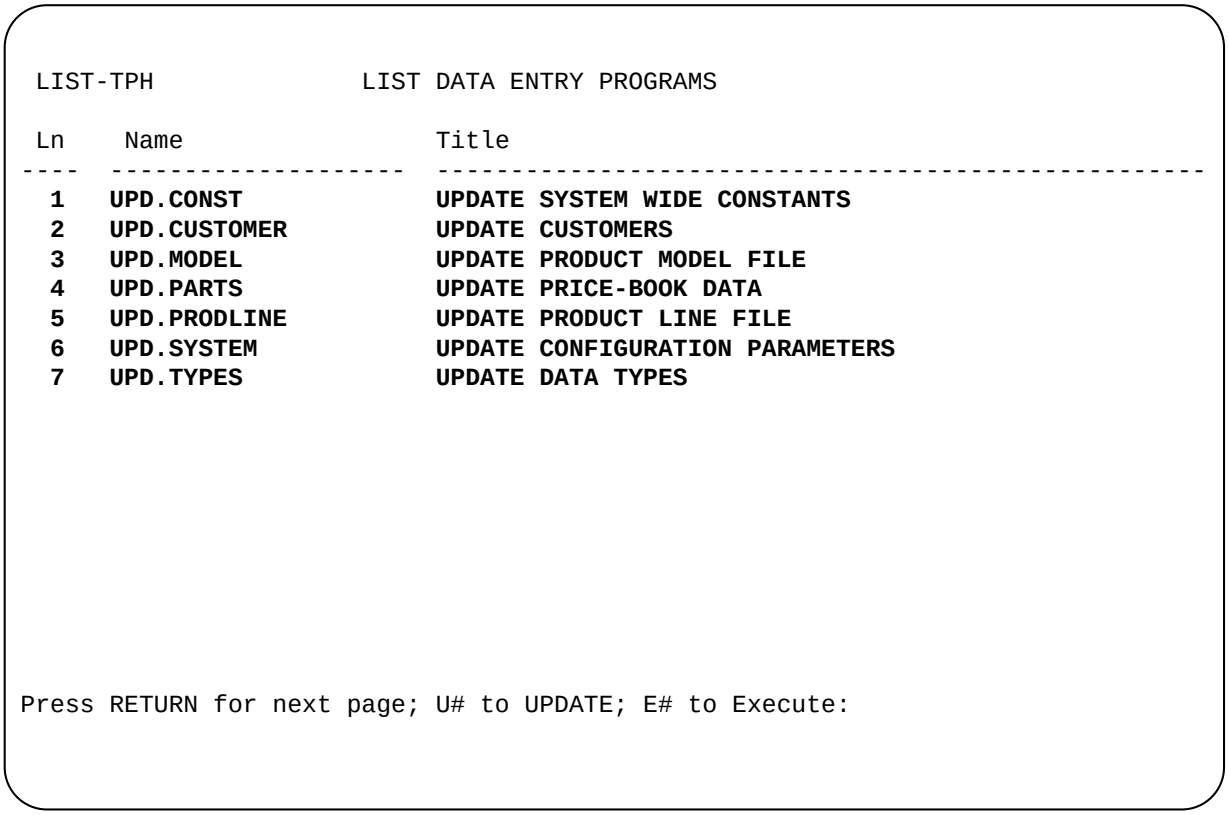
**8. Indent** This field is used to specify how many spaces are to be allowed for each level of code written. The default indentation is set in the **Define System Constants** Screen.

**9. # Screens** This field is used by **The Programmer's Helper** to determine how many data entry screens are to be used per program generated. For example, you could have an Order Entry application with two screens: one for the order header information, and a second for the order detail data.

**10. Use Include** Enter "Y" here if you expect that the program generated will be larger than your maximum item size. If this option is set, **TPH** will write portions of the program in separate items and place an **INCLUDE** or **INSERT** statement in the main program. You must enter the particular keyword used by your system on the **Define System Constants** Screen.

## 1.5 List Data Entry Programs

This function will list all of the data entry programs that have been already defined in the **UPDATE PROGRAM DEFINITIONS** screen. The following screen image shows a typical listing from this routine:




**List Data Entry Programs Screen**

In addition to displaying the current data entry screen programs, this routine also allows for the updating and execution of a desired program that is displayed.

**Update** To update the current program definition for the item desired, you simply type the letter U, and the number of the screen program you wish to update.

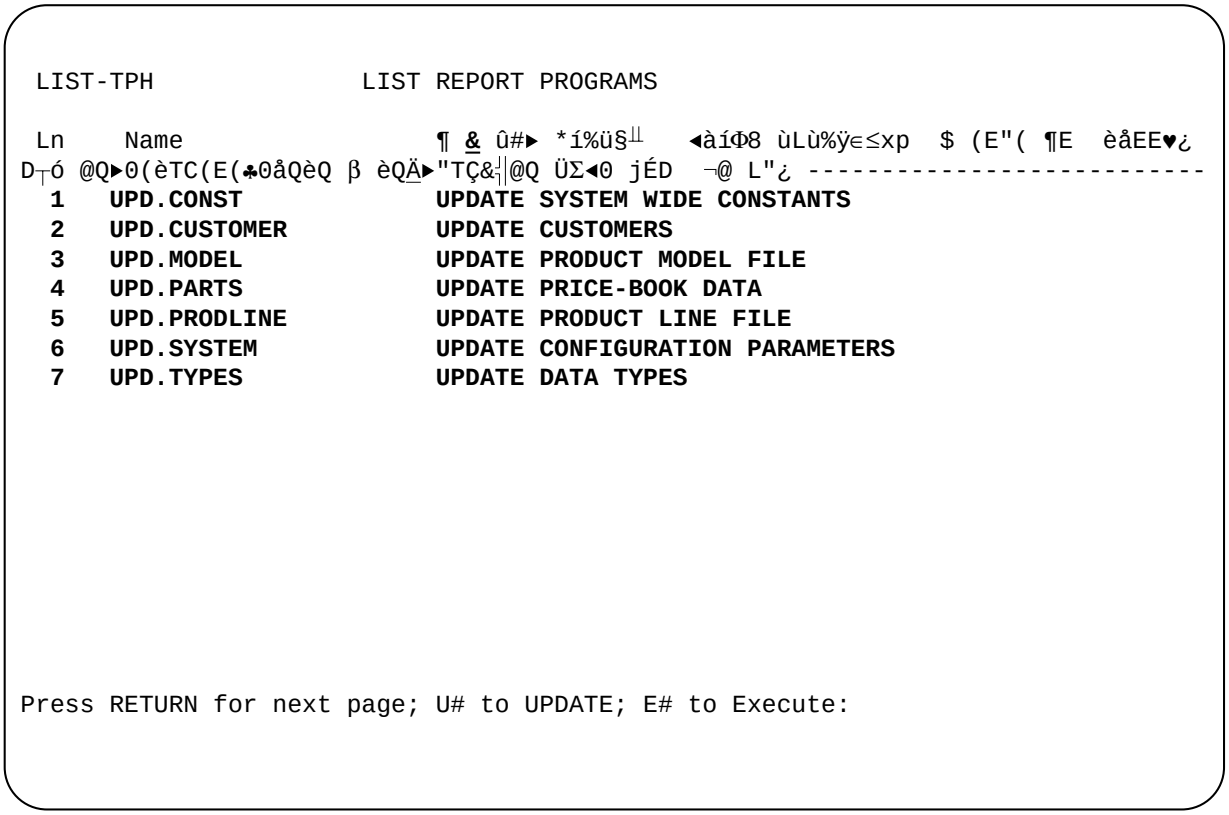
For example: to update the **UPD.PARTS** definition, you would type the following - U4.

**Execute** To execute a data entry program from the data entry screens report, simply press the letter [E] and the number of the program that you wish to run.

For example: to execute the **UPD.CUSTOMER** program, you would type the following - [E2].

## 1.6 List Report Programs

Much the same as the **LIST DATA ENTRY PROGRAMS**, this routine allows you to see all of the reports that have been previously defined for use. The following screen image is an example of this routine:




**List Report Programs Screen**

In addition to displaying the current report programs, this routine also allows for the updating and execution of a desired report program that is displayed.

**Update** To update the current report program definition for the item desired, you simply type the letter [U], and the number of the report program you wish to update.

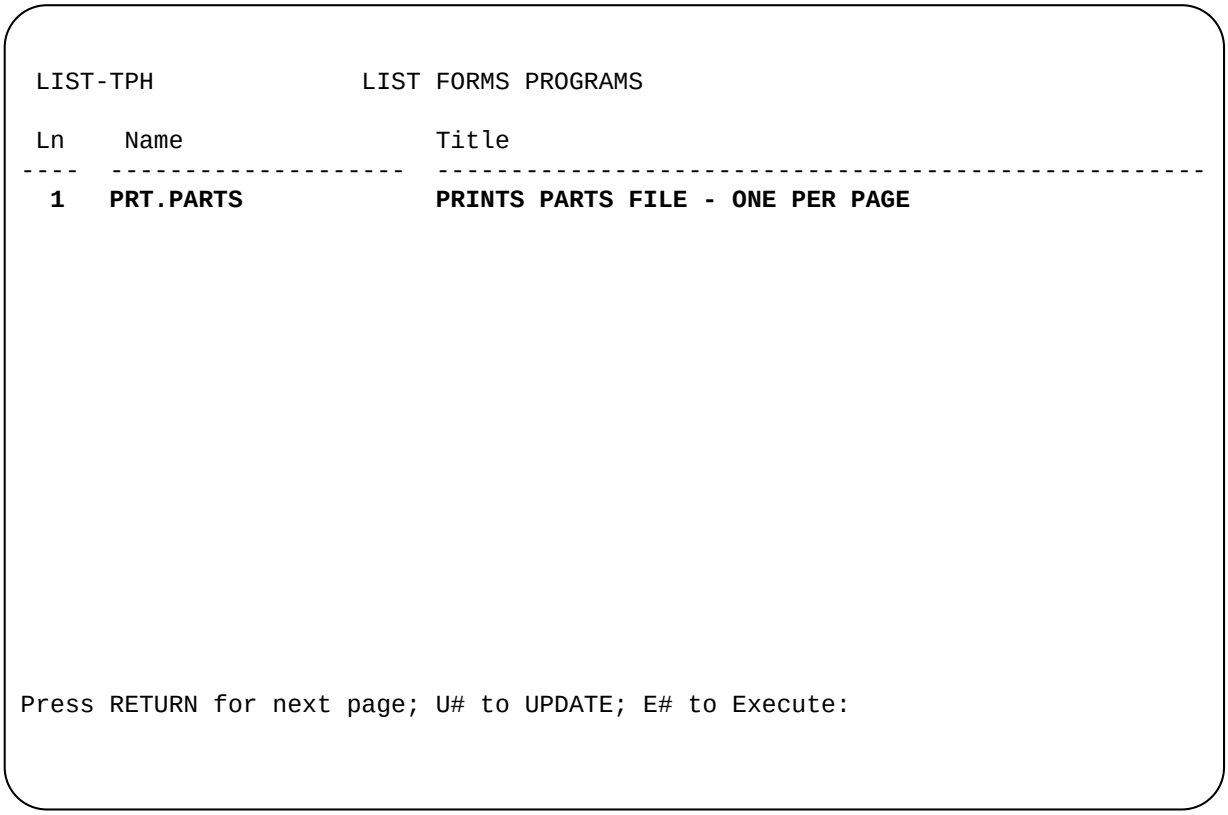
For example: to update the **PARTS-01** definition, you would type the following - [U1].

**Execute** To execute a report program from the report screens report, simply press the letter [E] and the number of the report program that you wish to run.

For example: to execute the **PARTS-01** program, you would type the following - [E1].

## 1.7 List Forms Programs

This function displays all of the forms programs that have been previously defined. It acts in much the same way as the **LIST DATA ENTRY PROGRAMS** routine. The following screen is a sample of this program:




**List Forms Programs Screen**

In addition to displaying the current forms report programs, this routine also allows for the updating and execution of a desired forms report program that is displayed.

**Update** To update the current forms report program definition for the item desired, you simply type the letter (U), and the number of the report program you wish to update.

For example: to update the **PRT-PARTS** definition, you would type the following - (U1).

**Execute** To execute a forms report program from the forms report screens report, simply press the letter [E] and the number of the forms report program that you wish to run.

For example: to execute the **PRT-PARTS** forms program, you would type the following - [E1].

## 1.8 Data Entry Program Development

This tool will probably be the most widely used one in **THE PROGRAMMER'S HELPER**. This is where the actual data entry screen development is handled. This section will be covered in depth in Chapter Two.

## 1.9 Report Program Development

This section will take you into the tool that allows you to create simple reports. This portion of **The Programmer's Helper** will be detailed in Chapter Three.

## 1.10 Forms Program Development

In this module, you can define programs that print formatted forms. This tool is used to design formatted reports or for use with pre-printed forms such as invoices, packing slips, sales order acknowledgement, checks, etc... This will be covered in depth in Chapter Four.

## 1.11 Menu Program Development.

This option allows you to define menus that control the flow of information in your system. It will be covered extensively in Chapter Five.

## 1.12 Index File Development.

This option allows you to define B-Tree indexes. These indexes will automatically be maintained whenever a TPH generated program updates the data file. The definition and use of these indexs are described in Chapter 6.

# 2 Data Entry Program Development

Unlike many other program and application generators, **The Programmer's Helper** does not require an extensive run time module in order to function. The screen definition, or screen picture, can be built using the standard PICK Editor (the **ED** verb), or with the supplied "screen painter". All of the information required to generate a BASIC program is taken from the screen's definition and the data file's dictionary. Once data is defined, there is no further need to answer pages of questions in each screen defining prompt fields. In fact, once the dictionary for a file has been defined, it is just a matter of minutes before a working screen can be generated and put into use.

This philosophy of "what you see is what you get" (**WYSIWYG** as it is referred to in the word processing and desktop publishing arena) with regard to screen definitions has a number of distinct advantages:

- Row and column numbers never have to be calculated. Also, when changes to prompt positions are made, no manual effort needs to be made in reference to column and row positions.
- To line up fields on a screen you merely move them around until the look right.
- Inserting, deleting and moving text can be done quickly and simply.
- The screen can be filled with a lot of descriptive text, or it can be kept very brief and simple.
- YOU ultimately control the way a screen will appear.

**The Programmer's Helper** can also be used to automatically build a default screen based on the dictionary definitions for you. The **AUTO-BUILD** function will select all of the "A" type items in the data dictionary specified and arrange them on the screen in attribute number order. If there are more than 20 attributes in the dictionary, the screen will be arranged in two columns; the first 20 attributes will reside in one column, and the next 20 in the second column. **NOTE:** Any attributes beyond the first forty will be ignored by the **AUTO-BUILD** program. The attributes falling into the over 40 category can, however, be used by **The Programmer's Helper** by simply painting them onto the screen after **AUTO-BUILD** has done its job.

Once a screen definition has been defined, or automatically built, the next step is to "pre-process" the screen to check for any possible errors. The next step is to "test" the screen. This step checks the screen layout and any BASIC functions that may occur. If the above two steps were completed successfully, the following step generates and compiles the actual BASIC program. These procedures can be repeated as often as possible until the program reaches its final format.

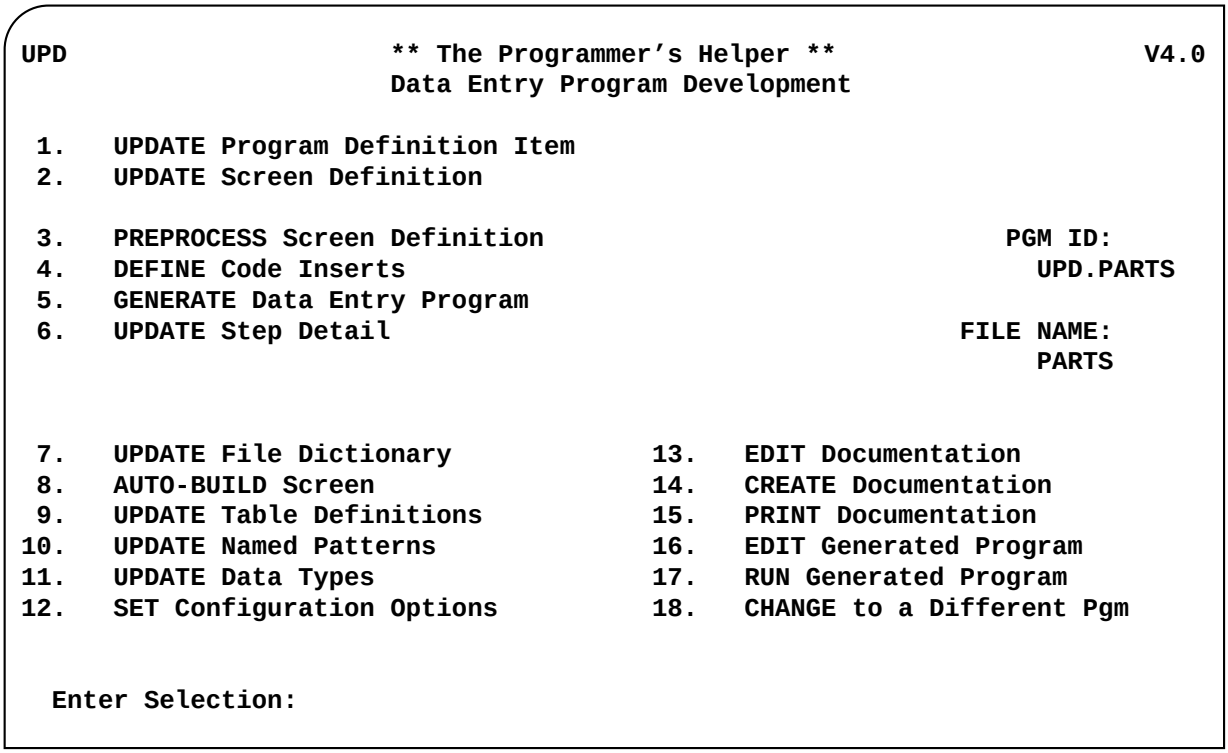
## 2.1 The Development Cycle

The first and foremost step when done creating a new file is to create the dictionary for that file. For the rest of this chapter, it is assumed that the dictionary for a file has already been entered and is complete. If a file has never been used with **The Programmer's Helper** before, you will need to define the file in the **Define Files** function on the TPH Master Menu. Remember, this has to be done first before using the file.

To access the data entry screen development portion of **THE PROGRAMMER'S HELPER** can be done in one of two ways:

| | |
|---|---|
| **From the Master Menu** | Select Option 8 on the Master Menu |
| **From TCL (Terminal Control Language)** | Type **UPD** or type **UPD <program name>**<br><br>For example, if you were going to make a change to an existing program called **UPD.CUSTOMER**, you would type the following:<br><br>**UPD UPD.CUSTOMER** |

If a program name is not supplied with the **UPD** verb, the system will prompt you for one. If a program definition already exists for this program, then you will be taken directly to the development menu. If the program being requested was not previously defined, the system will automatically take you to the **UPDATE Program Definition Item** on the **UPD** menu. When you have finished answering the prompts for the program definition the following screen menu will be displayed:




**Data Entry Program Development Menu**

The following pages will cover, point by point, the screens found in the **Data Entry Program Development Menu**.

## 2.1.1 Update Program Definition (Option 1)

The program definition item contains the name of the main file that will be updated, and the name of the file that will save the code that will be generated in a later step (source file). You can also enter a program title plus two program description lines for documentation purposes.

```
UPD.PD                  *** UPDATE PROGRAM DEFINITION ***

   Program Name:  UPD.PARTS
                                                        8.Indent:    5
1. Program Type:  ENTRY   DATA ENTRY PGM                9.# Screens: 2
2. Program Title: UPDATE PRICE-BOOK DATA               10.Use INSERT:N
3. Program Desc:  This program is used to update the PARTS master file.
                  You may add, delete or change part numbers.

4. Data File:     PARTS            PARTS MASTER FILE
5. Source File:   BP
   Pgms Called:   6.1   UPD.MODEL        UPDATE PRODUCT MODEL FILE
                  6.2   UPD.PRODLINE     UPDATE PRODUCT LINE FILE
                  6.3   DISPLAY.TABLES   DISPLAY/UPDATE LOOKUP TAB

----------------------------Other Files---------------------------------
 LN    FILENAME         DESCRIPTION                 ITEM ID ATTR       USE
 7.1 MODELS           PRODUCT MODEL MASTER            MODEL-NO         X5
 7.2 PROD-LINE        PRODUCT LINE MASTER             PROD-LINE        R
 7.3 CUSTOMER         CUSTOMERS MASTER FIL
 7.4
```

**Update Program Definition**

Fields to document subroutine calls and other file references (such as files used for data validation) are also included on this screen. These fields are automatically updated once the actual program is generated. This feature allows the developer to track the design and scope of a system without "losing" components. Cross-referencing of files are defined in this screen as well.

By selecting the item **Bottom** from the bottom line menu, you will be shown a window where all of the bottom line commands are defined for this program. You may delete any of the commands there or add your own. For example, if you do not want the user to delete records, remove the "delete" command from this window and that code will not be included when the program is generated.

```
-------------------------------------------** Define Standard Bottom Lines **
ln   Word       Description                          Code   Type     Segment ID
1.1  File       File the item                        F       B       FILE
1.2  Edit       Edit a field                         E       B       EDIT
1.3  Delete     Delete the item                      D       B       DEL
1.4  Step       Step thru fields                     S       B       SEQ
1.5  Zoom       Zoom a field to full screen          X       B       EXP
1.6  Next       Display next page of values          N       M       NEXT
1.7  Prev       Display previous page of values      PP      M       PREV
1.8  Add        Add a new value                      A       M       MVA
1.9  DelVal     Delete a value                       K       M       MVD
1.10 Ins        Insert a value                       I       M       MVI
1.11 Exit       Exit without filing                  /*      B       CAN
```

**Define Program Bottom Line Menu**

## 2.1.2 Update Screen Definition (Option 2)

Once the program definitions have been created, the next step is that of building a screen definition. This can be done by either using the Screen Painter supplied with **The Programmer's Helper**, or by using the PICK Operating System line editor (**ED**). Other editors, such as word processors could also be used (i.e. the **JET-EDIT** verb of the **JET (Tm)** word processing system).

The Screen Painter is an excellent tool for positioning prompts and text. The PICK editor is more suited to relocating large blocks of text. A typical screen format is defined in the following screen:

```
UPD.PARTS          * * *  UPDATE PRICE BOOK INFORMATION  * * *

 0. Part Number             ;PART-NO

 1. Model Number            ;MODEL-NO:TMODELS;X;1;1::UPD.MODEL
 2. Part Description        ;DESC::S3


 3. Avail                   ;STATUS:#PART-STAT;30,+1,+1

 4. Catalog Section         ;SECTION:(1N"-"1A)
 5. Business Area           ;BUS-AREA:#PART-BA
 6. Product Line            ;PROD-LINE:TPROD-LINE;X;2;2::UPD.PRODLINE
 7. Price Effective         ;EFF-DATE

 8. List Price              ;NEW-LIST-PR    9. Old List Price     ;@
10. Dealer Price            ;NEW-NET-PR    11. Old Dealer Price   ;@

12. Ytd Sales               ;YTD-SALES::N  13. Last Updated       ;@
```

**Update Screen Definition Sample**

## 2.1.3 Pre-Process Screen (Option 3)

Since the screen definition is entered as a "text" item using a "text" editor, no syntax or error checking is performed while the screen layout is being defined. The screen pre-processor validates the following:

1. All of the dictionary items being referenced in your screen definition can be found.
2. That there are no missing or duplicate sequential step numbers.

3 Miscellaneous command syntax and detectable typing errors.

Once the screen has been pre-processed, the resulting pre-processed screen is stored in the dictionary of your data file with an item-ID of **%<program>.**

For example, if you are pre-processing a screen called **UPD.CUSTOMER**, the result after pre-processing would be stored in the dictionary of the CUSTOMER file as follows: **%UPD.CUSTOMER**.

### 2.1.4 Define Code Inserts (Option 4)

**The Programmer's Helper** generates a standard, generic, PICK/BASIC program in a highly structured manner. Should special features or edits be required, these can be accomplished through the use of Code Inserts. A Code Insert is just what its name describes: A segment, or piece, of code that is inserted or placed into a program to add functionality that is currently not available directly from **The Programmer's Helper**.

Code Inserts allow you to modify the code that will be generated in step 5. Almost every conceivable part of the program can be modified. Column numbers, line numbers, conversion specifications and the like may be specified as variables so that if, for example, you change your screen layout, your code insert will not also have to be modified.

### 2.1.5 GENERATE Data Entry Program (Option 5)

After the Program Definition, the Screen Definition, the Screen Pre-Process and the code inserts have been defined, you may now proceed to generate an actual PICK/BASIC program.

In actuality, two programs are generated for each data entry screen by this process. The first is a small (about 10 line) program that **CALL**s the actual data entry program. The second

program is the main data entry program and is, in reality, a subroutine. The calling program has the same name as the subroutine except that it has the word **.CALLER** appended to it. For example:

Program Name - **UPD.CUSTOMER**

Calling Program - **UPD.CUSTOMER.CALLER**

### 2.1.6 UPDATE Step Detail (Option 6)

This function allows you to modify the parameters of each data entry step, or screen prompt, without having to use the Screen Painter, and without having to Pre-Process the screen again. If changes are made at this point, the Screen Definition item is automatically updated. Some people prefer this method of defining the screen rather than using the Screen Painter.

Once the initial screen has been laid out, it is entirely possible to do all further screen modifications without ever entering the Screen Painter again. The following illustrates a sample update step screen:

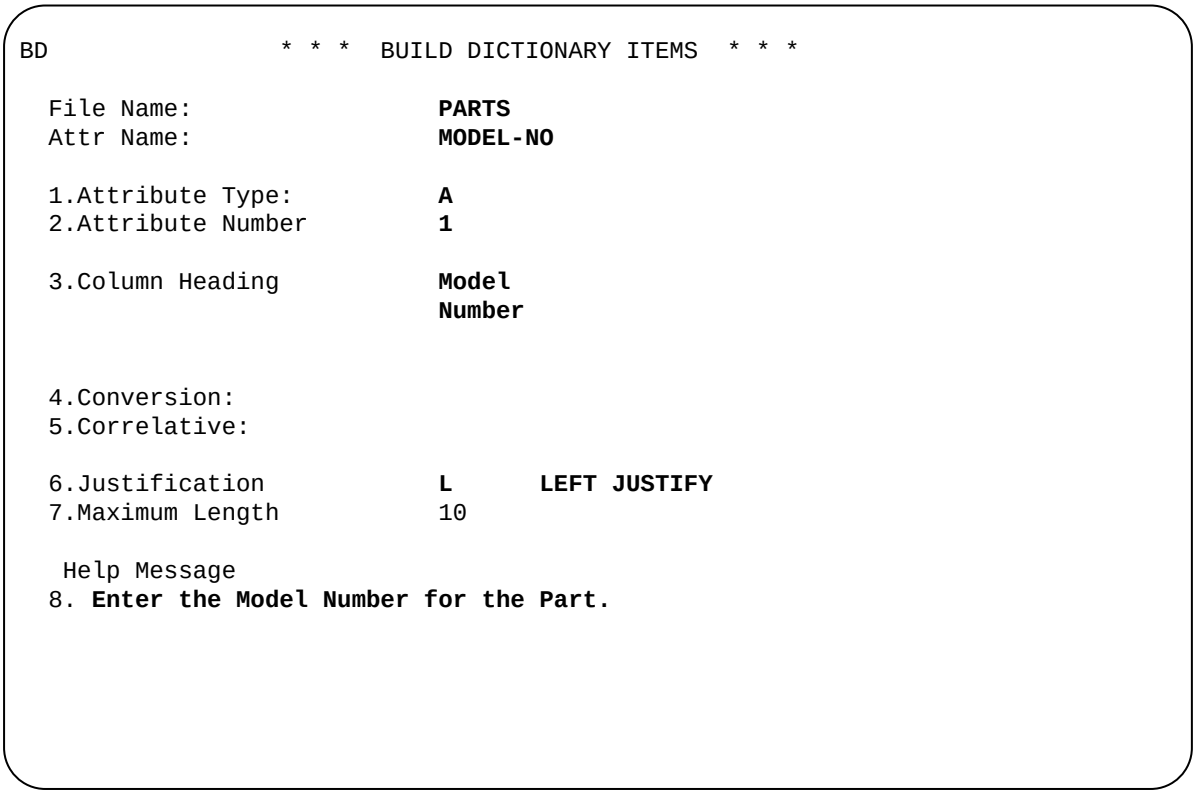
```
 UPD.STEPS              * * *  UPDATE STEP DETAIL * * *
                                                    ◄─ Çà¶ÇT♣àá└pPí♦ó¶ϒÇ► Σ 8üB
ü■ DhE(à0&QéPí áB(PÇ í íTó¶┬ C(E(à0åPèQ a í►é0─YØcL)t ΣÄ8üBü■ pC.ö: r├£Ah♦
└ p•á
 1.Attr Name:       MODEL-NO
 2.Prompt Text:     Model Number
 3.Input Line:       4
 4.Input Column:    21
 5.Prompt Column:    1
 6.Required Field:  N
 7.Validation Type: F  FILE VALIDATION     8.Validation:  TMODELS;X;1;1
 9.Option:          L                     10.Window Size:

11.Desc Len:        25                  14.Update Program:UPD.MODEL
12.     Column:     +17
13.     Line:        +0

15.Default Value:
16.Formula:

   Help Message:
17. Enter a description of the Part Number
```

**Update Step Detail Screen**

## 2.1.7 UPDATE File Dictionary (Option 7)

Since all of the data entry steps used by The Programmer's Helper must have a corresponding attribute dictionary definition item, it is often necessary to make changes to the dictionary items. A simple, yet powerful, dictionary update program is supplied with **The Programmer's Helper** to help you create and/or modify the dictionary items in a file.

You can use your own dictionary update program if you wish, however. Consult your system documentation (usually the ACCESS or ENGLISH reference manuals) for more information on creating and modifying dictionary item definitions.




**Build Dictionary Items Screen**

## 2.1.8 AUTO-BUILD Screen (Option 8)

This tool is used to construct an initial screen definition automatically without having to use the Screen Painter. It works by selecting all the "A" type dictionary items (these are items containing an "A" in attribute 1) from the file dictionary and placing them on the screen in attribute number order. You will be given the chance to exclude attributes from the screen before the screen definition is built.

The following screen illustrates how the **AUTO-BUILD** function works:

```
SCREEN ID:  UPD.CUSTOMER                * Define Fields *

ln    Sel   Attr Name    Prompt          Validation    Options
1.1    Y    CUST-NO      Customer Number                  R
1.2    Y    NAME         Name
1.3    Y    ADDR         Address
1.4    Y    CITY         City
1.5    Y    STATE        ST
1.6    Y    ZIP          Zip             $ZIP
1.7    Y    PHONE        Phone           $PHONE
1.8    Y    SINCE        Customer Since                   N
1.9    Y    YTD-PURCH    YTD Purchases   MR2              N
```

**AUTO-BUILD Sample Screen**

## 2.1.9 UPDATE Table Definition (Option 9)

This function is used to validate a prompt that has a limited number of acceptable responses. This is accomplished through the use of lookup tables. This function creates and maintains these lookup tables. The following screen image is an example of the **UPDATE Table Definition** screen:

```
UPD.TABLE          *** UPDATE VALIDATION TABLES ***

  Table ID:   AREA

 Line     Code            Description
 1.1       R              RECEIVING MEGATAPE
 1.2       O              OUTSIDE TEST
 1.3       T              TEST MEGATAPE
 1.4       A              ASSY MEGATAPE
 1.5       F              FIELD SERVICE
 1.6
 1.7
 1.8
 1.9
 1.10

 2. Error Message:  Must be R, O, T, A, or F

    Last Update:    10/18/87
```

**UPDATE Table Definition Sample Screen**

## 2.1.10 UPDATE Named Patterns (Option 10)

With some fields it is valid to have multiple pattern masks. For example, a ZIP CODE field may have a format of '**5N**' or '**5N"-"4N**' or even '**1A1N1A" "1N1A1N**' for Canadian postal codes. These patterns can all be grouped under a pattern name and data matching any of these formats will be accepted.

A number of patterns are already included with **The Programmer's Helper**. These include: telephone numbers (PHONE), Social Security Numbers (SSN) and Zip Codes (ZIP). Any number of patterns can be created to meet your precise requirements.

The following is a sample of the **UPDATE Named Patterns** screen:

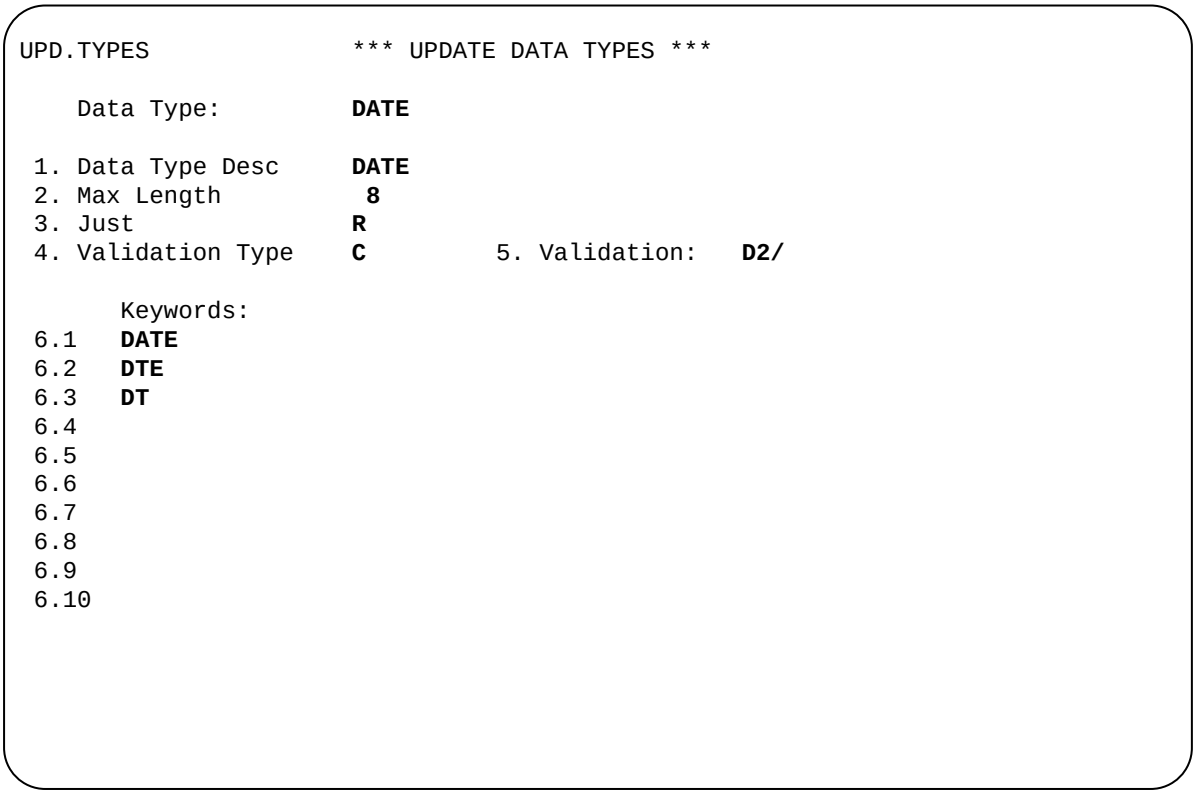
```
UPD.PATTERN            *** UPDATE NAMED PATTERNS ***

   Pattern Name:    ZIP

1. Pattern Desc:    ZIP CODE

  LN    PATTERN                        OUTPUT FORMAT
 2.1    5N
 2.2    5N'-'4N
 2.3    9N                             L(#5-#4)
 2.4    '1A1N1A 1N1A1N'
 2.5
 2.6
 2.7
 2.8
 2.9
 2.10

 3. Error Message:   MUST BE A ZIP CODE
```

**UPDATE Named Patterns Sample Screen**

## 2.1.11 UPDATE Data Types (Option 11)

Data types simplify the task of creating dictionary definition items. For example, you may specify that any dictionary definition items containing the word "DATE" or the word "DTE" will have as a default, a conversion of "D2/", a field justification of "R" and a length of "8".

Data types for **DATE**, **AMT**, and **NAME** are supplied with **THE PROGRAMMER'S HELPER**. Additional data types can be created without limit as you see fit.




**UPDATE Data Types Sample Screen**

## 2.1.12 SET Configuration Options (Option 12)

Throughout **The Programmer's Helper** there are a number of system wide variables that can be changed in order to adapt **The Programmer's Helper** to your own set of screen conventions. You have control over what commands are to be used to file an item, delete an item, etc., and the text of the "bottom line" command prompt. An example of this screen follows:

```
UPD-CONTROL           *** SET CONFIGURATION OPTIONS ***

   CONTROL ID:      UPD-CONTROL

 The 'BOTTOM LINE' listing of commands is defined on the line below:
 1. ENTER COMMAND: X = EXPAND: F = FILE; D = DEL; S = SEQ; * = CANCEL:

 The following 'BOTTOM LINE' will be used for programs with Multivalued Attrs
 2. ENTER: F=FILE; D=DEL ITEM; S=SEQ; N=NEXT PG; K=DEL VAL; A=ADD VAL; *=CANCL:

 3. Character to separate Step Number from Prompt: .

 4. Used ENHANCED (Bar) Bottom Line(Y/N)?          Y
```

**SET Configuration Options Sample Screen**

You may also define what bottom line commands are recognized in each program. This is done by selecting the **Bottom** option from the bottom line menu. Changes made from this program are "global" -- they will effect every program generated unless it has its own custom bottom line.

```
-------------------------------------------** Define Standard Bottom Lines **
 ln   Word       Description                        Code   Type     Segment ID
 1.1  File       File the item                      F       B       FILE
 1.2  Edit       Edit a field                       E       B       EDIT
 1.3  Delete     Delete the item                    D       B       DEL
 1.4  Step       Step thru fields                   S       B       SEQ
 1.5  Zoom       Zoom a field to full screen        X       B       EXP
 1.6  Next       Display next page of values        N       M       NEXT
 1.7  Prev       Display previous page of values    PP      M       PREV
 1.8  Add        Add a new value                    A       M       MVA
 1.9  DelVal     Delete a value                     K       M       MVD
 1.10 Ins        Insert a value                     I       M       MVI
 1.11 Exit       Exit without filing                /*      B       CAN
```

**Global Bottom Line Menu Definition**

### 2.1.13 EDIT Documentation (Option 13)

One of the benefits of **The Programmer's Helper** is that it cuts down on the time and effort involved with producing easy to read, effective user documentation. The **EDIT Documentation** function allows you to modify or elaborate on the text that will be included in the program documentation. This is very useful for describing features that have been added using the "Code Inserts" feature.

### 2.1.14 CREATE Documentation (Option 14)

This step creates a set of operator instructions for each step of your program. The help messages that were entered for each field, as well as information on edits, validations, etc., will be used in creating this documentation.

The resultant documentation will be stored in a file called **TPH-DOC** in **RUNOFF** (see the RUNOFF section of your PICK documentation for more information on this product). This is the documentation that is displayed when you type a double question mark "??" at any prompt.

### 2.1.15 PRINT Documentation (Option 15)

This option will print a copy of the documentation that was created using the **CREATE Documentation** command. It will also print a copy of the screen layout for the program.

### 2.1.16 EDIT Generated Program (Option 16)

This option is used to examine or modify the program that has already been generated by step 5. When this option is chosen, you will be in the PICK EDITOR and all of the editing commands will be available to you. When you are through editing the code, you will be asked if you wish to compile the code at this time.

**WARNING!** If you modify the generated programam using the BASIC editor, these changes be lost if you regenerate the program at a later date using **The Programmer's Helper**. To avoid losing any modifications, use the "Code Inserts" function as much as possible. Program regenerations will always look for any Code Inserts that need to be placed in the code.

### 2.1.17 RUN Generated Program (Option 17)

This option is rather self documenting. If you choose this option, it will run the generated and compiled code for this program. The program, at this stage, will act just as it would when put into a live production setting.

### 2.1.18 CHANGE To A Different Program (Option 18)

This selection will allow you to work on a different  data entry program without having to return back to the main menu or TCL.

## 2.2 The Screen Definition Item

The **Screen Definition Item** determines where on the screen the data entry will take place, and where the prompting text will be located. It is a "snapshot" of what the data entry screen will look like.

The screen definition is composed of 23 lines (each stored as an attribute) containing up to 80 characters (or columns) each. If text appears in column 20 of line 5 of the screen definition, that same text will appear in column 20 line 5 of the finished data entry program.

Following is a sample of a Screen Definition screen for a program called **UPD.PARTS**:

```
UPD.PARTS          * * *  UPDATE PRICE BOOK INFORMATION  * * *

 0. Part Number            ;PART-NO

 1. Model Number           ;MODEL-NO:TMODELS;X;1;1::UPD.MODEL
 2. Part Description       ;DESC::S3


 3. Avail                  ;STATUS:#PART-STAT;30,+1,+1

 4. Catalog Section        ;SECTION:(1N"-"1A)
 5. Business Area          ;BUS-AREA:#PART-BA
 6. Product Line           ;PROD-LINE:TPROD-LINE;X;2;2::UPD.PRODLINE
 7. Price Effective        ;EFF-DATE

 8. List Price             ;NEW-LIST-PR    9. Old List Price     ;@
10. Dealer Price           ;NEW-NET-PR    11. Old Dealer Price   ;@

12. Ytd Sales              ;YTD-SALES::N  13. Last Updated       ;@
```

**Screen Definition Sample**

The data entry screen is logically divided into a number of "steps". Each of these steps represent a prompt for a piece of information. The steps are always numbered sequentially and must begin with zero (which is the ITEM-ID). Each step is comprised of the following format:

**NN.TEXT ;ATTR:CHECK:OPTIONS:UPD-PGM:FORMULA**

or, for example

**7. PROD CODE: ;PROD.CODE:TPRODUCTS;X;;1:R:UPD.PROD**

Each component of the step is described in detail below:

**NN** This is the step number. The step number must be a sequential numeric integer with no gaps allowed. The steps are always executed in numerical order. The step number must always be followed by a trailing period (i.e. **7.**). All step numbers must be unique with two exceptions: 1) the step is a multivalued window, or 2) the step is part of a multiple valued item ID.

**TEXT** This is the prompt text describing the data to be entered. This text can be as terse or descriptive as you like. In the above example, text would be "**PROD CODE:**".

**ATTR** The attribute name must be a valid attribute definition in the dictionary of the file being updated. The dictionary definition item is retrieved to determine the attribute number, the maximum length, and the justification code (left or right justified). If there is a conversion code, it will also be used unless it is overridden by the **CHECK** field following. The attribute name is used as the variable name in the generated program.

In the example, the **ATTR** is: "**PROD.CODE**"

**CHECK** This is a validity check applied on input. It may be an ACCESS (or ENGLISH) conversion, or one of the following special conversions:

**(pattern)** can be any valid BASIC **MATCHES** pattern specification. For example, **(3N'-'1A2X)** will validate a pattern of 3 numeric, a hyphen, one alpha and 2 alphanumeric.

**#Table** this is a table lookup validation. For example, **#CUST.TYPE** will look for a valid customer type code in the **CUST.TYPE** table. The validation tables are maintained using Option 9, **UPDATE Table Definitions**, in the **Data Entry Program Development (UPD)** menu.

**/string** This means that the input must match one of the characters in this string. For example, **/ABC** would require an input of **A**, **B**, or **C** to be accepted.

**$Name** This means that the data must match one of the pattern specifications in the pattern list stored under "name". For example, **$PHONE** will test for an input pattern defined under "**PHONE**" in the TPH-PATTERNS file. These patterns may be maintained by using selection 10 on the **Data Entry Program Development (UPD)** menu.

**Tfile;x;;amc (Translate)** This signifies a translate conversion and is treated as a file validation specification. In the above example, this type of validation is shown by: **TPRODUCTS;X;;1**

**OPTION** Additional options can be specified here, with no limit to the number of options that can be requested. The options can be in any order except that the **H**, **S**, and **V** options must last. The accepted options follow:

**A** This option defines the attribute in question as an "Audit Stamp". This means that the current time, date, port number and user ID (logon name) will be written into the attribute, separated by astericks. The attribute will be non-updatable and you may want to use a "Group Extract" (e.g G0*1) to limit the display to one of the above named data items.

**B** This option generally means that no blanks are allowed during input. This is generally used when processing an Item-ID.

**C** This option will allow data to be entered into new items, but will not allow the field to be changed if it is an existing item.

**D** This option defines the attribute in question as a "Date Stamp". This means that the current system date will be assigned to this attribute when the item is filed.

**Hn** When this option is used, it defines a simple multivalue that is displayed horizontally rather than vertically. This is similar in nature to the **Sn** function, but the multivalues are display left to right instead of top to bottom.

**L** This mean that attribute will automatically have a default of the last valued entered. This is useful if the data tends to be entered in "batches" and many values are repeated from one item to another.

**N** This means that this field is for display only and no updating is allowed.

**R** This option signifies that the input for this field is **required**.

**Sn** This choice signifies that the input will be a simple multivalue of depth "n". A simple multivalue is an atomic set. In other words, this multivalue set is not related to any other multivalued fields.

For example, **S3** will allow for three multivalues to be entered as in a multi line address or multiple phone numbers. Simple multivalues are displayed vertically, one below the other.

**T** This option, when used, means that the default value for this attribute will be today's date.

**Vn** This means that this input field is the controlling attribute of a multivalued window. The optional "n" defines the depth of the window (i.e. the number of lines displayed for this window at one time). If the "n" option is omitted, the window will extend to the bottom of the screen.

This option allows any number of values to be entered regardless of the window size. It also allows for controlling/dependent multivalued attributes as in a line item on an

invoice. There can exist any number of multivalued windows on a screen, in any combination of simple, horizontal or complex multivalues. **The Programmer's Helper** even allows windows to be placed side by side.

**Z** This means that the data entered for the attribute will be zero filled to the maximum length. This is used primarily with older applications that zero fill the item ID.

**UPD-PGM** If any field contains the item-ID from another file, the name of another program generated by **The Programmer's Helper** may be specified, and that program will be invoked whenever the entered data cannot be found within the validation file for this field.

An example of this function would be to call a customer entry program from a sales order header screen if the customer number entered could not be found on file.

This option can also be called from the bottom line prompt by invoking the **ZOOM** (formerly **EXPAND**) command. In the above example, this option is shown by: **"UPD.PROD"**.

**FORMULA** You may specify a step to be calculated from other steps on the screen. The calculated value must have an attribute reserved for it in the data file and have a dictionary definition item. The formula is entered as you would a BASIC expression with one exception: If your attribute name is not a valid BASIC variable (for example, QTY-ORDERED is not a valid BASIC variable), you have to put a "N()" around it, much in the same manner as an "A Correlative." For example:

**10.Extend Price: ;EXT-PRICE::::QTY*PRICE**

but

**10.Extend Price: ;EXT-PRICE::::N(QTY-ORD)*N(UNIT-PRICE)**

Any valid BASIC function may be used. All attribute names used in the formula must appear somewhere on the data entry screen. The attribute will automatically be non-updatable.

Any time you change any of the values used in the formula, the formula will be recalculated and the result displayed.

There is a special expression that may be entered in the formula field to sum all values of a multivalued attribute:

**+<attr-name>**

or, for example:

**10. Total Debits: ;TOTAL-DEBITS::::+DEBIT-AMT**

The "N()" feature should not be used here. Any time a value is added, changed or deleted the total will be automatically updated to reflect the changes.

**COLON Separators**

The colons "**:**" are required only to separate the various parts of the step definition. If your definition does not include a validation, any options, or an update program, then no colons are required. If there is only a validation, only one colon between the attribute name and the validation is required.

Example:

**5. Customer Name: ;NAME (there are no validations)**

**8. Customer PO#...: ;PO::R (required field, no other options)**

**Validation Description**

When your program is running, and a file translate or table validation is specified for this program step, the value received by the translate or the description obtained from the lookup table will be displayed to the right of

the input data on your screen. The length and location of this description can be modified by adding another clause after the validation. This clause has the following format:

**;length,col,line**

where **length** is the length of the description, **col** the column number where the description will be displayed, while **line** is the line on which the description will be shown.

The column and line numbers are treated as "absolute" values unless preceded by a plus "+", or a minus "-" to indicate a value relative to the data entry point. For example:

```
7.PROD CODE:  ;PROD.CODE:TPRODUCTS;X;;1;20,+0,+1
```

This example will limit the description (the result of **TPRODUCTS;X;;1**) to 20 characters, and will display it one line below the prompt (+1) and directly underneath the data entry prompt position (+0).

Another example:

```
10. Customer Type:  ;TYPE:#CUST.TYPE;,20,10
```

This prompt performs a lookup on the **CUST.TYPE** table. The table lookup description will be 25 characters in length (the default), and the description will print  at column 20 line 10. **Note:** the column row positions are relative to the logical top of the screen.

Our last example:

```
10. Customer Type:  ;TYPE:#CUST.TYPE;15
```

This prompt will limit the description from the lookup file to 15 places and display to the right of the input field since no column,row positions were specified.

If you have a length of zero, it will suppress the display of the description, but the validation will be performed.

**Footnotes**

Since individual step definitions can become rather lengthy, it is sometimes impossible to fit all of the edit and validation specifications on the screen without overlapping other text or screen definitions. This is especially true when the step definitions are laid out in multiple columns (i.e. multivalued window).

This problem is solved by the use of a "footnote" page. The footnote page acts as an extension of the first page of a screen. Footnotes are also very easy to use. To use a footnote, place an "at-sign" ("**@**") immediately after the first semi-colon **";"**, in place of the attribute name. This tells the screen compiler to look on the footnote page for that steps screen definition.

The format of the footnote option is as follows:

```
NN.ATTR:CHECK:OPTIONS:UPD-PGM:FORMULA
```

or, for example

```
Screen 1     = 7. Prod Code:   ;@

Footnote     = 7.PROD.CODE:TPRODUCTS;X;1;1:R:UPD.PROD
```

The footnote page can be found by typing "**P**" three times while in the screen painter, or by going to lines 100-120 if you are using the **ED** verb.

**Multiple Part IDs.**

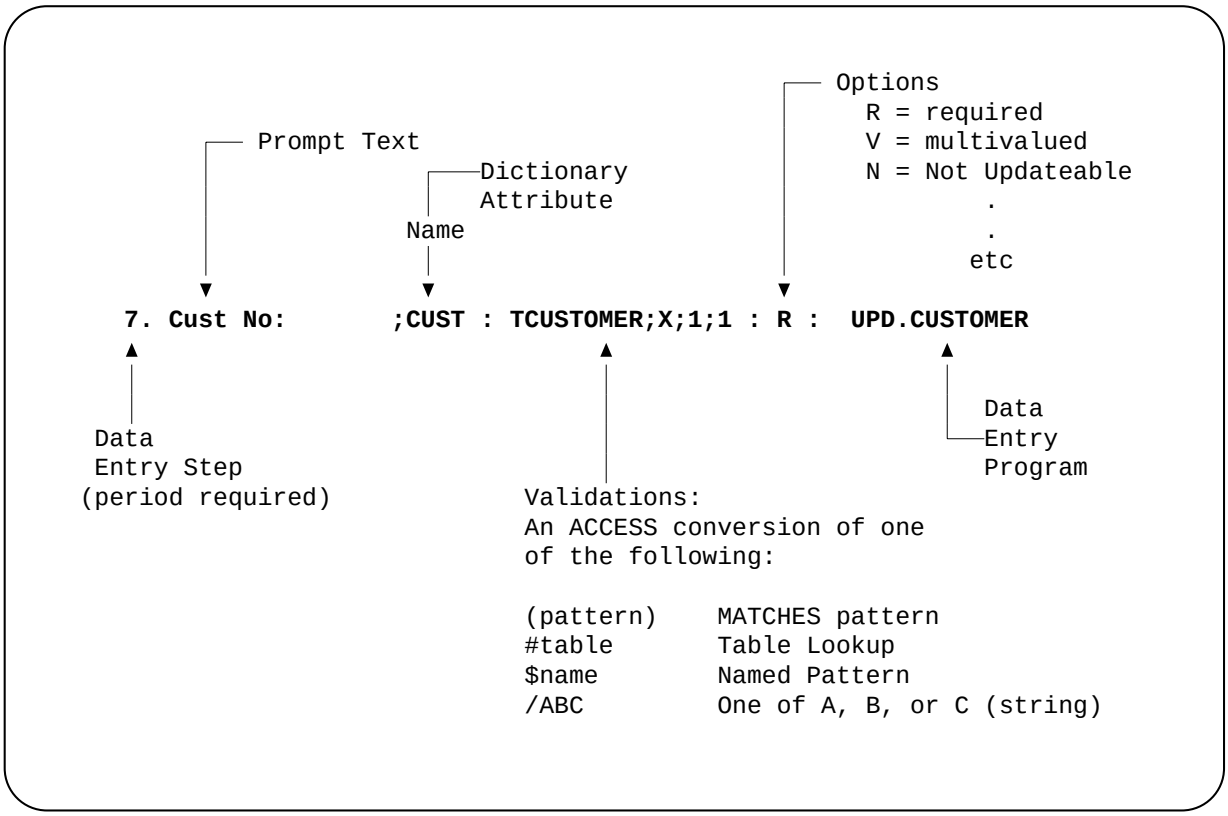
If a file has an Item ID comprised of multiple parts (i.e. **AAA*BBB*CCC**) there has to be one step zero (0) for each of the ID components. Additionally,

each attribute definition item in the file's dictionary must have the appropriate group extraction correlatives (i.e. **G0*1** for the first one, **G1*1** for the second one, and so on...)

**Note:** there can be no other intervening steps between the zero (0) steps. In other words, if an item ID were comprised of an Invoice number and a customer number (10001*12) the step for Invoice number and the one for Customer number would have to be placed sequentially on the screen and in the same order they are found in the item ID.

For more information on group extraction functions, please consult your ACCESS or ENGLISH reference manuals.

The following diagram illustrates a typical screen definition setup and what it means:




**Sample Screen Definition**

**NOTE -** If the options for a prompt include **V**, **S**, or **H**, these must be placed as the last option. Multiple options such as **D** and **N** can be entered together, as **DN**. Obviously, some combinations such as **R** and **N** can cause run time problems. You cannot set an option flag **R** for "required" and another one **N** for "display only" together in the same prompt.

## 2.3 The Generated Program

There are as many ways to structure a program as there are ways to write a murder mystery. Each author possess a certain style and technique for writing, and it does not matter whether one write

a program or a novel. There are no proven "correct" methods for writing. There are, however, some techniques that make for better and more readable programs. The programs created by **THE PROGRAMMER'S HELPER** follow the following guidelines:

- Block structured approach. The code is modular, with each section having a single entry point and a definite function to perform.
- There are no **GOTO**s or **RETURN TO**s. **GOTO** statements do not conform with the standards for structured coding. In complying with structured precepts, **The Programmer's Helper** uses no **GOTO** statements. The use of unstructured branching (i.e. **GOTO**) makes it very difficult to debug a program, thereby increasing dramatically the cost of maintenance.
- Each new level of structure (i.e. **IF-THEN**, **READ-ELSE**, **LOOP-REPEAT**, **BEGIN CASE**, etc...) is indented. This makes the block structure easy to identify and improves the general readability of the program. **NOTE**: the number of spaces that each structure will be indented is controlled by specifying the number of spaces desired in the **SET SYSTEM CONSTANTS** screen of the master menu.
- A generous use of source code comments (including blank comments for "white space") are provided to allow for simple, yet effective, program documentation.
- The use of meaningful variable names. Each screen "step" that is defined will take the dictionary definition name for an attribute and use it as a variable name in the generated program. For example, if the screen step calls a dictionary item named **PROD.CODE**, the variable name referencing this step in the program would be **PROD.CODE**. This feature allows for very simple maintenance of the resulting program.

## 2.4 Multivalued Attributes

**The Programmer's Helper** supports two different types of multivalues: simple and complex. Simple multivalues are composed of only one attribute and a predefined number of allowable values. Complex multivalues allow for multiple attributes defined in a Controlling/Dependent

arrangement, with no fixed number of allowable values.

The majority of this section will deal, in detail, with the complex multivalues. **NOTE**: there is no limit as to the number of simple or complex multivalues that can be used in one screen.

### 2.4.1 Complex Multivalues

There are several conventions that are used for processing sets of multivalued attributes in a data entry screen. **The Programmer's Helper** uses the "horizontal" method. The horizontal method signifies that each horizontal line of the screen represents an associated value. Each column represents one attribute. The leftmost column is the controlling attribute, and the rest are dependent attributes. The area in which the multivalued attributes are displayed is known as the multivalued window.

**NOTE:** the controlling and its dependent attributes must all fit on one line. This means that each set of associated values can only take up one line.

**For more information on controlling and dependent attributes, how they are set up and how they can be used most effectively, please refer to the ACCESS or ENGLISH reference manual that came with your system.**

The controlling attribute is marked by a "**V**" code in the Options section of the data entry screen step. The **V** may have an optional number following it that specifies the number of lines displayed in the window. If no number is specified after the **V**, the window will continue to the end of the screen. NOTE: dependent attributes need not have the V code, as they will share the same step number.

The start of a multivalued window is determined by the starting position of the controlling attribute. For example:

| Ln. | Part Number | Price Code | Qty-Ord |
|---|---|---|---|
| 7. | ;PART-NO:TPARTS;X;1;1:V | 7.;CODE | 7.;QTY-ORD |

In the above example, the attribute **PART-NO** is the controlling attribute and **CODE** and **QTY-ORD** are the dependent attributes. Since there is no specified number after the controlling attribute's **V**, the multivalued window will flow to the bottom of the data entry screen page.

**NOTE:** there may be any number of multivalued windows on a screen; you can even have two separate, unrelated, windows displayed side by side.

### 2.4.2 Simple Multivalues

A simple multivalue is used when the attribute being referenced only contains a fixed number of values, and the values are not related to any other multivalued attributes. Some common examples of simple multivalues would be:

- Multiple phone numbers per customer
- Multiple address lines
- Multiple description lines

An example of a simple multivalue that allows entry of three phone numbers into a customer file could be:

```
7. Phone #:     ;PHONE:(3N'-'3N'-'4N):S3
```

The "**S**" tells **The Programmer's Helper** that this step is a simple multivalue, and that 3 is the maximum number of values allowed. NOTE: the values in a simple multivalue are displayed vertically.

If the "**S**" is replaced by an "**H**", the values will be displayed on a single line, horizontally, each one space to the right of the last. Obviously, both **H** and **S** simple multivalues have to be designed with the screen constraints in mind (i.e. 80 columns by 24 rows).

## 2.5 Program Templates

**The Programmer's Helper** generates programs by combining a series of code segments or "templates". These templates are stored in the **TPH-PGM-SGMTS** file. Each template is composed of a segment of code that may include replacement "variables" and conditional (**IF**) statements. Which program templates will be used for a particular screen will be determined by the screen preprocessor (option 3 on the **Program Development Menu**).

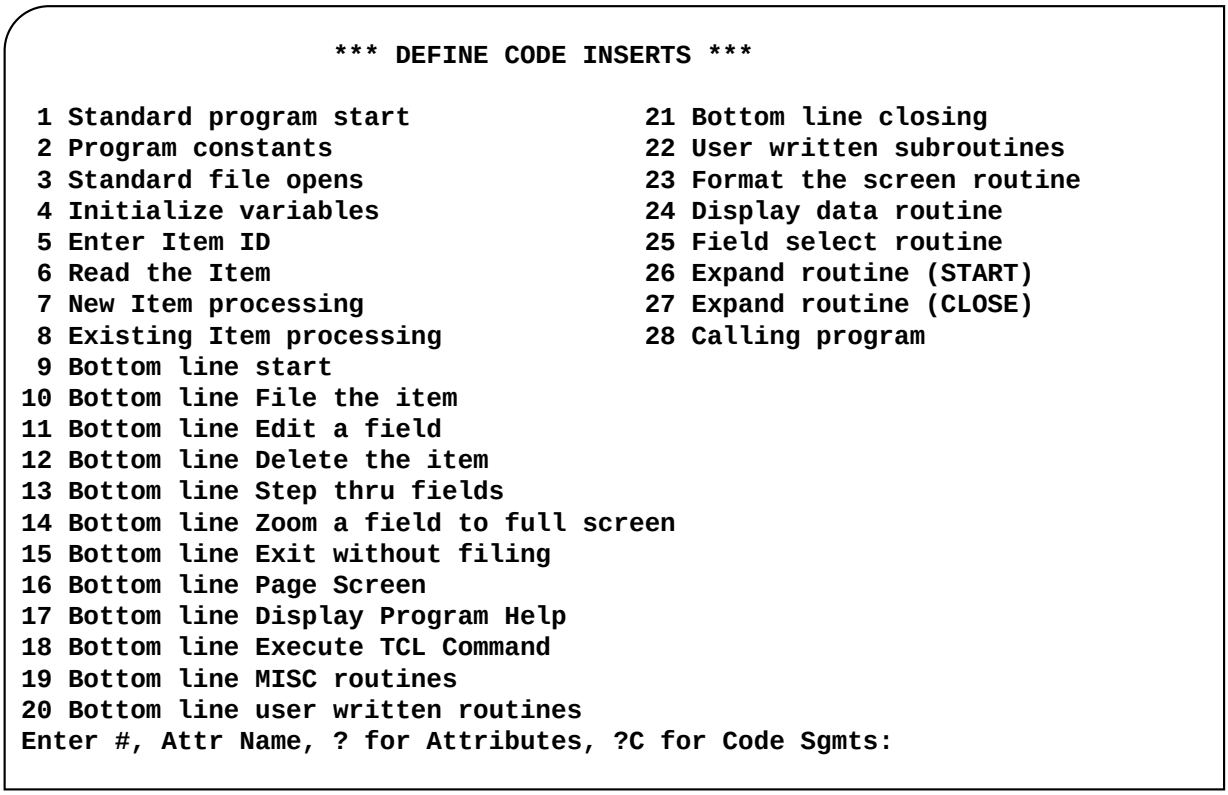
Each of the templates may be customized for a specific program using the **Define Code Inserts** function on the **UPD** menu. The customized templates are then stored in the **TPH-CODE-SGMTS** file. The master templates can be modified also, but great care should be taken when doing this as it may cause incorrect programs to be generated.

When the program code is written, **The Programmer's Helper** first goes to the **TPH-CODE-SGMTS** file to see if a modified template for the program exists. If none is found, **TPH** then reads the default template from the **TPH-PGM-SGMTS** file. By performing the code generation in this manner, all customization performed by the programmer is preserved regardless of how many times the code has been generated.

**Sequence of Events**

| | |
|---|---|
| **1** | The files are created for use by **The Programmer's Helper** |
| **2** | The files are defined for **The Programmer's Helper** |
| **3** | The file dictionaries are created for use by **The Programmer's Helper** |
| **4** | The screen is designed, or painted, in the **UPD** menu |
| **5** | The screen is pre-processed for syntax accuracy |
| **6** | Steps can be modified, or added, through **UPDATE Step Detail** |
| **7** | Program can be tested before compiling or adding code segments |
| **8** | Code Segments can be defined |
| **9** | Program Generation<br><br>a. **TPH-CODE-SGMTS** is checked for user defined templates<br>b. If none are found, **TPH-PGM-SGMTS** is used<br>c. The source code is generated |
| **10** | Compile the source code |

The next screen sample illustrates what happens when the **Define Code Inserts** function is chosen**:**




**Define Code Inserts Screen**

The **Define Code Inserts** screen has a bottom line prompt that asks the user for the following:

**Enter #, Attr Name, ? for Attributes, ?C for Code Sgmts:**

Following is a brief explanation of what happens when any of these options is chosen:

**#** This option means to enter the number on the menu that you wish to customize. For example, if you want to modify the template for number **17. Bottom Line Closing** you would enter the number '**17**' at the prompt.

**Attr Name** This option allows you to enter the screen step name of a field directly. For example, to edit the customer number (**CUST-NO**) code segments, you would simply enter the step name called **CUST-NO** at the prompt

Any screen step name can be used to customize its code segment.

```
                    *** DEFINE CODE INSERTS ***

The following sub-segments are available for the attribute MODEL.NO

 1 Data Entry
 2 Data Display
 3 Field Expand
 4 Cross Ref Add
 5 Cross Ref Delete









Enter Selection Number:
```

**Code Segments Attribute Name Sample**

**? for Attributes** When a question mark [?] is typed, all of the defined steps in the screen are displayed. Once the field you are searching for is identified, you leave this screen and type in the step's attribute name at the **Define Code Inserts** menu prompt.

```
            *** ATTRIBUTES USED IN "UPD.CUSTOMER" ***

CUST-NO                   NAME                      ADDR
CITY                      STATE                     ZIP
PHONE




Press RETURN Please:
```

**Code Inserts Attribute Name Listing**

**?C for Code Segments** Typing a question mark followed by the letter "C" ?C displays a screen report of all of the code segments that have been defined for the program being worked on. This function is for display purposes only.

The following sections will describe the various code segment templates that are available. **NOTE**: these templates reside in the **TPH-PGM-SGMTS** file. The segment name follows the description (e.g. UPD.START).

**Standard Program Start** This section contains the first 40 or so lines of the program. All system constants (i.e. **AM**, **VM**, **CLEAR.LINE**, etc.) are all defined here. You can verify that these are all valid for your system and modify them if necessary. (UPD.START)

**Standard File Opens** This template opens the main data file, the **TPH** support files, and any files that may be used for validation purposes. This can be customized to meet your needs also. For example, security files, special buffer files, mirror image file, etc. (UPD.OPEN)

**Program Constants** This is, in essence, a continuation of the **Standard Program Start** template. The **EQUATEs** for the main data file are appended to this template. (UPD.CONST)

**Initialize Variables** This template is used to initialize program variables before the main program loop is entered. (UPD.INIT)

**Enter Item-ID** This template contains the start of the major program loop and the code to input and retrieve a record's item ID. If any special actions need to be taken before or after entering item IDs they would be placed in this template. (UPD.ITEMID)

**Read Record** This template reads the item from the file and saves any values used for cross referencing or indexing. (UPD.READ)

**New Item Processing** Contains the code to be executed if the requested item ID is not on file (i.e. it is a new record). (UPD.NEW)

**Existing Item Processing** This template contains the code to be executed if the requested item ID is already on file. (UPD.OLD)

The following set of templates deals with the **BOTTOM LINE** commands. These commands may be executed from the screen bottom line prompt during run time. There is one template for each command.

**Bottom Line Start** This template contains the code to start the bottom line loop and to input the bottom line command. (UPD.BOTTOM.START)

**Bottom Line CANCEL Routine** The **CANCEL** command negates the update. (UPD.BOTTOM.CANCEL)

**Bottom Line FILE Routine** This template files the item and exits the Bottom Line prompting loop. (UPD.BOTTOM.FILE)

**Bottom Line EDIT Routine** The edit command really signifies the number of the data entry step to be edited. (UPD.BOTTOM.EDIT)

**Bottom Line DELETE Routine** This template is used for item deletion after entry. Before allowing the deletion, **The Programmer's Helper** asks whether you are sure about deleting this item. You answer with either a yes "Y" or no "N". (UPD.BOTTOM.DEL)

**Bottom Line SEQUENCE Routine** This causes each data entry step to be modified in whatever sequence you specify. For example, let's suppose there are twelve steps in a screen. If you type S8, **The Programmer's Helper** will prompt, in sequence, from step 8 through step 12. (UPD.BOTTOM.SEQ)

**Bottom Line ZOOM Routine** In **The Programmer's Helper**, the **ZOOM** command simply signifies the calling of another program at a certain place in the program. For example, if you are entering a part number that is non-existent, the **ZOOM** function would call the program that updates part numbers if the step was designed to do so. (UPD.BOTTOM.EXP)

**Bottom Line PAGE SCREEN Routine** This controls the page, or screen, number that the data entry operator is currently looking at. This option only works in the case of a multi-screen application. (UPD.BOTTOM.PAGE)

**Bottom Line INSERT VALUE Routine** This and all other multivalue routines are present only if there is at least one complex multivalued window on the screen. It will prompt the user for line number before which the new value will be inserted. (UPD.BOTTOM.MVI)

**Bottom Line DELETE VALUE Routine** This template is used to delete a specific set of values from the multivalued window. (UPD.BOTTOM.MVD)

**Bottom Line APPEND VALUE Routine** — This is used to append a new set of values to the end of a multivalued window. (UPD.BOTTOM.MVA)

**Bottom Line DISPLAY NEXT Window** — This bottom line routine will cause a specific multivalued window to display the next page of values. (UPD.BOTTOM.NEXT)

**Bottom Line DISPLAY PREVIOUS Window** — This will cause the previous page of values to be displayed for a specific multivalued window (UPD.BOTTOM.PREV)

**Bottom Line MISC Routines** — This contains some useful utilities. You may enter a ">" to execute any TCL statment or type a "!" to display the version number and and date generated for the program. If you have any of your own utilities that you want included in every program (such as a pop up calculator or appointment scheduler) you should include them in this template.(UPD.BOTTOM.MISC)

**Bottom Line User Written Routines** — You may add your own bottom line commands in a program by placing them in this template. (UPD.BOTTOM.USER)

**Bottom Line Closing** — This template is used to close out, or end, the bottom line loop. Any special actions that need to be done before exiting the bottom line should be done at this point. (UPD.BOTTOM.CLOSE)

The following templates do not deal with the bottom line prompting. They are miscellaneous routines comprised of the following:

**User Written Subroutines** — If there are any unique subroutines that you wish to add to this program, that can be done so through this template. (UPD.USER)

**Format the Screen Routine** — This template assembles a variable named "**screen**" which displays all of the fixed, non-variable, text on the

screen. In the normal course of events, this subroutine is only called one time. **NOTE:** the variable "**screen**" is in lower, not upper, case. (UPD.SCREEN)

**Display Data Routine** This template displays all of the single, non-multivalued, data on the screen and calls the multivalue display routine for each multivalued window defined. (UPD.DISPLAY)

**Display Multivalued Window** This template will contain the code to display one page from each of the multivalued windows in the screen. (UPD.MV.DISP.START)

**Delete Multivalue** This template will contain the code to delete a specific value from a multivalued window. (UPD.MV.DEL.START)

**Insert Multivalue** This template will contain the code to insert a new value into a multivalued window. (UPD.MV.INS.START)

**Calculate Maximum Value** This template will count the number of values in each multivalued window.

**Field Select Routine** This template is the routine which determines which of the step update routines to call. (UPD.SELECT)

**Expand Routine** This template deals with the logic for calling, or expanding, a screen step so that another program can be invoked. (UPD.EXPAND)

**Add to Cross Reference** This template contains the code to update a cross reference file. (UPD.XREF.ADD)

**Delete from Cross Reference** This template contains the code to delete items from a cross reference file. (UPD.XREF.DEL)

**Calling Program** This routine determines the logic for the calling program sequence of events. (UPD.CALLER)

In addition to the above defined templates, each data entry field is written as a separate, internal (**GOSUB**) routine. This approach to code generation allows for maximum flexibility and customization. The attribute name of the field is used as the template's name. There exist at least two "sub-templates" or "sub-segments" for each data entry attribute, and there may be as many as ten. These sub-segments are listed below:

**Data Entry** This is the main data entry routine. Its is formed from one of four items in the **TPH-PGM-SGMTS** file

a. **UPD.SIMPLE**
b. **UPD.FILE**
c. **UPD.TABLE**
d. **UPD.PATTERN**

Each of these template names may have a "**.SW**" (for simple multivalue) or a "**.HW**" (for horizontal multivalue) appended.

The **UPD.FILE** template is used for the processing if input with a file validation. **UPD.TABLE** is used to for input with a table lookup validation. **UPD.PATTERNS** deals with input with matching named patterns. The **UPD.SIMPLE** template is used for the processing of all other fields.

**Display Data** This is the piece of code that goes into subroutine 2200 or subroutine 2300 to display existing data on the screen.

**Cross Reference ADD** If the specified attribute is the index to a cross-reference then this step and the following sub-segment will appear. This template contains the code necessary to maintain the cross reference pointers. If you are using your own indexing software (such as a B-tree package) you would insert the **CALL**s to your software in this sub-segment. (**NOTE:** please check your B-tree reference manual for information on subroutine calls and data passing from the application program to the B-tree routines.)

**Cross Reference DELETE** — This sub-segment contains the code required to delete pointers from the cross-reference file. It contains the code necessary to maintain the cross-reference pointers. As in **Cross Reference ADD**, if you are using your own B-tree package, this section is where you would put the coding hooks to it.

**Attribute EXPAND** — This sub-segment contains the code required to set up the **CALL** to the appropriate data entry program.

The next four steps will only appear if the attribute selected is the controlling attribute in a multivalued window:

**Multivalue Controlling Loop** — This sub-segment contains the code that automatically sequences from attribute to attribute within a window.

**Multivalue Delete** — This template contains any special processing required to delete a value from the multivalued attribute set.

**Multivalue Insert** — This code segment contains the logic necessary to INSERT a new value into the multivalued attribute set.

**Multivalue MAX.VALUE** — This sub-segment contains the logic that counts the number of values within a multivalued window set.

The following step will only appear if the attribute selected is a calculated field.

**Calculate Formula** — This sub-segment contains any special processing that must be done when evaluating the formula for a calculated field.

## 2.6 Replacement Variables

When in the process of editing, or constructing, a code insert, there exist certain variable values which we shall call "**replacement**" variables. These variables are evaluated at the time the program is being generated to see what kind of value they possess.

These replacement variables fall into two broad categories:

1. STATUS variables, and
2. STEP variables.

Status variables are those that are global and applicable throughout the program. For example the data file name, or the record name. Step variables, on the other hand, are valid only within a data entry step and their values will change from step to step.

**NOTE:** all replacement variables begin with a percent sign. i.e. **%PGM**

### 2.6.1 STATUS Variables

| | |
|---|---|
| **%PGM** | **Program name**. This variable holds the name of the program being generated. |
| **%TITLE** | **Program Title.** This variable holds the title of the program as entered in the **Update Program Definition** screen. |
| **%FN** | **Data File Name**. This is the name of the file as it was designated at the time of creation. For example, if the file was called **CUSTOMER**, the **%FN** variable would contain the value **CUSTOMER**. |
| **%FV** | **File variable name**. This is the internal BASIC file variable name used in the **OPEN** statement. For example, if the file name is **CUSTOMER**, the **%FV** variable would contain the value **CUSTOMER.FILE**. All file names used by **The Programmer's Helper** are appended with "**.FILE**" for internal use. (**NOTE:** if the file was originally named with the ".FILE" extension, the code generator ignores it and leaves it as is.) |
| **%RECORD** | **Record Name**. This is the name of the array that will be used to **MATREAD** and **MATWRITE** the item. For example, if the file name used in the program is **CUSTOMER.FILE**, the name of the data array would be **CUSTOMER.REC**. All arrays used to contain file data use the file name with an appendage of "**.REC**". |

**%ARRAY** **Array size**. This contains the dimensions of the array used to **MATREAD** and **MATWRITE** the record. This value is taken from the **Define Data Files** on the Master Menu. If no value was specified, the default value is 100.

**%MPART** **Maximum Part**. This variable contains the maximum number of parts allowed, or contained, in the Item ID. For example, if the item ID looked like "**AAA*BBB**", the **%MPART** variable would contain a two (2), one for value "AAA" and one for value "BBB".

**%LAST** **Last Screen Line Used**. This is the last line on the screen that contains data. For example, if **%LAST** were equal to 12, it would mean that from line 13 to the bottom of the screen no data appeared.

**%MVFOUND** **Multivalue Found**. This is one (1) if there exist any complex multivalue windows in the screen. If simple values (or simple multivalues) only exist, then this value is zero (0).

**%MAXWIND** **Maximum Multivalued Window**. This is the number of complex multivalued windows that appear on one screen.

**%VERS** **Program Version Number**. This is the counter that is increased by one (1) every time the program is generated.

**%XREF** **Cross Reference Flag**. This variable has the value one (1) if there is a cross reference that needs to be updated.

**%BTREE** **BTREE Flag.** This variable has the valie one (1) if the current file has a TPH B-Tree index defined.

**%MAXPAGE** **Maximum Screen Page.** This is the number of separate screens that are linked together.

**%ENHANCE** **Enhanced Bottom Line.** This variable is null if the enhanced ("light bar") bottom line menu is turned off. If the enhanced bottom line is enabled, this variable will have the value one (1).

**%AUDITFV** **Audit File Variable Name.** If this variable is non-null, it contains the internal (file variable) name of the AUDIT file. If there is no AUDIT file then the value is null.

## 2.6.2 STEP Variables

The following replacement variables are called "**step**" variables. Their values will continually change from step to step and they are only defined within a data entry step:

**%STEP** **Step Number**. This variable contains the step number of the attribute being entered.

**%TEXT** **Prompt Text**. This is the prompt text that is displayed for each step. For example, let's look at step 7.

**7. Customer No ;CUST-NO**

The prompt text for this step would be "Customer No ". Prompt text can usually be defined as the text between the period and the semi-colon in a step definition.

**%COL** **Input Column**. This is the data entry input column number for this step.

**%LINE** **Input Line**. This is the data entry input line number for this step.

**%LEN** **Input Length**. This is the maximum allowable input length for this step. This value is derived from attribute 10 of the dictionary definition item for this field. For example, if attribute 10 of the dictionary item **CUST-NO** in the file **CUSTOMER** were "6", the value of **%LEN** would be six (6).

**%REQ** **Required Flag**. This variable contains the value "**R**" if the step is required and cannot be bypassed.

**%TPOS** **Text Position**. This is the column number where the prompt text will be printed for this step.

**%DPOS** **Description Position**. This is the column number where either the table or file descriptions will be displayed.

**%DLINE** **Description Line**. This is the line number where either the table or file descriptions will be displayed.

**%DLEN** **Description Length**. This is the length that will be used when displaying the table or file descriptions.

**%SPEC** **Special Options**. The variable contains the value of the special options field taken from the screen definition item. (i.e. this is where the "**V**" for multivalued steps goes)

**%AMC** **Attribute Number**. This is the attribute number of the step being input.

**%PART** **Item ID Part**. This variable contains the value of the Item ID part being processed by this step. For example, if the item ID is "AA*BBB", the value for **%PART** when processing "BBB" would be two (2).

**%SCHAR** **Separation Character.** This is the character that is used to separate two or more parts of a field. If a "G0*1" correlative is used, the separation character is an "*".

**%ATTR** **Attribute Name**. This is the name of the attribute being input for this step. The attribute name has been converted into a valid BASIC variable name. For example, if the attribute name where **CUST-NO**, the value for **%ATTR** would be **CUST.NO**. Notice that all special characters in variable names are changed to a period.

**%OATTR** **Original Attribute**. This variable contains the value for the unconverted attribute name. For example, if the attribute definition name was **CUST-NO**, the value for **%OATTR** would also be **CUST-NO**. Unlike the **%ATTR** variable, which converts the attribute name to a valid BASIC variable name, the **%OATTR** keeps the same name for the attribute. This could be valuable for creating an ACCESS or ENGLISH statement from within the program dynamically based on pre-specified conditions.

**%TABLE** **Table Name**. This is the name of the lookup table, if any, for this step.

**%EDIT** **Edit Specification**. This variable contains the ACCESS conversion or other validation specification that is passed to the **SCREEN.INPUT** subroutine.

**%LOOKFILE** **Lookup File Variable**. This is the internal BASIC file variable name used to open the lookup file.

**%LOOKATTR** **Lookup Attribute Number**. This is the attribute number to retrieve when performing a file validation.

**%LOOKNAME** **Lookup File Name.** This is the actual physical name of the file being used for file validation.

**%JUST** **Field Justification**. This is the justification parameter from attribute 10 of the dictionary item (usually "L" or "R") for this step.

**%MASK** **Format Mask**. This is the format mask used when displaying the data. It is derived from the justification parameter and the maximum field length value for this step.

**%DMASK** **Description Mask**. This variable contains the mask used when displaying a table or file description for this step.

**%UPDPGM** **Update Program Name**. This is the update program specification from the screen definition for this step. This variable contains the name of the program that can be called to add data to a validation file.

**%UPDSIZE** **Update Program Size.** This is the number of lines required by the update program described above.

**%MV** **Multivalue Flag**. This variable contains the value one (1) if the step is part of a complex multivalued window.

**%SW** **Simple Window Size**. If the step is a simple multivalued attribute, this variable contains the value for the maximum number of entries that are allowed for this window.

**%MVWIND** **Multivalued Window Number**. This variable identifies which window a multivalued attribute is in. The first window is one (1), and so on...

**%WWIDTH** **Window Width**. This variable contains the width of the multivalued window in spaces. It is used to clear data from the window.

**%SUBSTEP** **Substep Number**. For multivalued windows, this is the relative number of the field within the window. For example, sub-step 1 is the controlling attribute, sub-step 2,3 and so on are dependent attributes.

## 2.7 Conditional Statements

Each code insert may include one or more conditional statements. These are very similar to the BASIC **IF** statement and are used to conditionally include, or exclude, portions of the code from the final generated program. The general format of a conditional statement is as follows:

```
%IF %var op const
...code if true
%ELSE
...code if false
%END
```

Another example of a conditional statement could be:

```
%IF %MV EQ 1
help.msg = help.record(%STEP)<1,%SUBSTEP>
%ELSE
help.msg = help.record(%STEP)
%END
```

As in the BASIC **IF** statement, the **ELSE** clause is optional and may be omitted. The **%var** represents any valid replacement variable as defined above. The "**op**" operation may stand for either **EQ** (equals) or **NE** (not equals). The comparison value "**const**" should not be enclosed in quotes, even if the value is non-numeric.

To test for a null value using conditionals use the form

```
%IF %var
```

Here again, **%var** can be any valid replacement variable.

## 2.8 Generated Program Features

The data entry programs that are generated by **The Programmer's Helper** have a number of built in useful features. This section will describe some of these in greater detail.

**Automatic Item ID Generation**

If you type a pound sign [#] in place of the item ID step for any screen, it will automatically assign the next sequential available number as the item ID. Each time the [#] is pressed, the sequential ID counter is incremented by one (1).

The sequential numbers for each file are kept in a **TPH** file called **TPH-LAST-ID**. Each record in the **TPH-LAST-ID** file is comprised of a data file name (i.e. **CUSTOMER**) and the last value used which is located in field one (1).

This feature is very useful for generating new customer numbers, invoice numbers, check numbers, transaction numbers, etc...

**Backup One Field**

If you type a caret [^] at any step prompt other than the item ID prompt, it will back up one program step. On multivalued windows, typing a [^] will back you up one field until you reach the controlling attribute field for the window, at which time it will back up one value.

This feature is of special interest to data entry operators who have made a mistake while entering a step, noticed it, and can immediately backup to that step and correct it.

**Help Messages**

Typing a question mark [?] at any field prompt will display the single line help messages that you entered when defining the dictionary attribute for this step.

If you type two question marks [??], you will see the paragraph from the operator instructions for the current step. A question mark [?] entered at the bottom line prompt will display the documentation for the entire program.

**Additional Step Information**

On step that contain a table validation, a named pattern validation, or a file validation, you may type a tilde [~] to see a displayed list of all the choices available at this prompt.

For table validations, you will see a list of all valid values for the table. For named patterns, you will see a list of all of the valid patterns that will be accepted. For file validations, the validation file will be sorted and the items displayed at the bottom of the screen. (**NOTE:** you wish to disable the file validation display for large files using code inserts since the sort could take a very long time)

In the case of tables and files, you can simply select the item from the listed display by simply typing in the line number of the item that you wish to select.

## 2.9 The Screen Painter

The screen painter supplied with **The Programmer's Helper** is essentially a one page word processor. The operator can perform the following functions with the screen painter: move text all around the screen, insert text, delete text, and so on...

The screen painter is a very useful tool that allows you the capability of getting your screen to look "just right". The screen painter commands are all single characters which take effect the moment the character is pressed. The following sections will explain the screen painter commands in detail.

### 2.9.1 Cursor Positioning Commands

**Move Cursor**

The numeric keypad on your terminal is used as a "compass" to move the cursor around. The following illustrates this:

[8] moves the cursor up one
[2] moves the cursor down one
[4] moves the cursor one left
[6] moves the cursor one right
[1] moves diagonally left and down one
[3] moves diagonally right and down one
[7] moves diagonally left and up one
[9] moves diagonally right and up one

The space bar [Space Bar] will also move the cursor right one position.

**Cursor Home**

The number zero [0] will move the cursor to the upper left corner of the screen. This is known as the **HOME** position.

**Cursor Bottom**

The period [.] will move the cursor to the bottom left corner.

### 2.9.2 Text Insertion/Deletion Commands

**Type Text** The letter "T" [T] will allow you to enter text until the return [↵] key is pressed. Any existing text that you write over will be overwritten. If a typing mistake is made, the backspace key [←] can be used to correct it.

**Insert Text** The letter "I" [I] will also allow you to enter text until the return [↵] key is pressed. However, existing text will be moved to the right as new text is being entered. To backspace, you must use the underscore key [_]. **NOTE:** text that is moved beyond the right screen margin WILL be lost!

**Delete Text** The letter "D" [D] will delete the character that is at the present cursor position. All text that is to the right of the deleted character will be moved one space to the left.

**Erase Text** The letter "E" [E] will replace the character at the current cursor position with a blank. No other text will be moved or affected.

**Insert Line** The plus key [+] will insert a blank line if the cursor is at the left edge of the screen. If the cursor is in the middle of a line, the plus key will cause the line to split from that point, moving the split text down to the next line. As lines are inserted, the bottom line will begin rolling off the screen and be lost.

**Delete Line** The minus key [-] will delete the current line if the cursor is at the left edge of the screen. If the cursor is in the middle of a line, the text to the left of the cursor will be joined with the text to the right of the cursor on the following line. The letter "K" [K] will also delete a line. Deleting lines adds blank lines to the bottom of the screen.

**Scroll Left** The regular slash [/] key will move a portion of the screen to the left of the cursor down one line. The left portion of the last line will roll off the screen and is lost. All text to the right of the current cursor position remains unchanged.

**Scroll Right** The backslash [\] key will move a portion of the screen to the right of the screen down one line. The right portion of the last line will roll off the screen and be lost. All text to the right of the current cursor position remains unchanged.

### 2.9.3 File Commands

**File Screen** The letter "F" [F] will save the current screen to disk and exit the screen painter. If there was an existing screen with the same name as the one being edited, the existing one will be overwritten. Analogous to the "FI" command in the system editor.

**Save Screen** The letter "S" [S] will save the current screen to disk but will remain in the screen painter to allow for further processing. If there was an existing screen of the same name, it will be overwritten. Analogous to the "FS" command in the system editor.

**Exit Screen** The letter "X" [X] will cause you to exit from the screen painter without saving the current screen to disk. Upon exiting, the screen will revert to its previously saved state. If the screen had never been saved before, then the screen will simply be non-existent after issuing the exit command. Analogous to the "EX" command in the system editor.

### 2.9.4 Miscellaneous Commands

**Transpose** The letter "Z" [Z] will cause the character at the current cursor position to be exchanged with the character that follows it. Typing [Z] again will return the same characters to their originals positions before the first transposition.

| | |
|---|---|
| **View On/Off** | The letter "V" Ⓥ is used to turn the display of the column and row numbers of the current cursor position on or off. Typing Ⓥ once again will restore the column and row counters at the bottom of the screen. |
| **Page** | The letter "P" Ⓟ will cause you to switch between the "footnote" and the regular page for this screen. |
| **Compile Screen** | The letter "C" Ⓒ will invoke the screen pre-processor or "compiler". The screen will be filed first. |
| **Generate Program** | The letter "G' Ⓖ will invoke the screen generator as if it was selected using item 5 on the **UPD** menu. The screen will be filed first. |
| **TCL Command** | The greater than sign ⓢ will allow you to enter a TCL command. This command will be executed and the you will be returned to the painter exactly where you were before. The screen will be filed first. |

## 2.10 The SCREEN.INPUT Subroutine

**The Programmer's Helper** uses a very efficient way of testing all the data that can ever be input into a generated program. All data entry input validations are performed by an external (cataloged) subroutine named **SCREEN.INPUT**. This subroutine performs the following validations:

- Applies any conversion to the existing data
- Displays the converted data padded with dots to indicate the maximum number of characters allowed for input.
- Accepts the input.
- Performs specified edit checks on the entered data:

  • Valid Date
  • Valid Time
  • Valid Number

- Valid Translate Conversion
- Do a Pattern Match
- Do a Table Lookup

- Displays the entered data after the conversion has been applied to it. This display will be padded with blanks to remove the input prompt dots that were displayed earlier. Fields with a numeric conversion, such as **MD** or **MR**, will be displayed using right justification.

- Display a "help" message if a question mark [?] is entered.

Additionally, the subroutine will recognize the character for [CTL-F], which signifies the right arrow cursor key on many terminals, to mean "leave the character as is". This is used by a data entry operator to change part of a field without having to reenter the whole field again.

If your application requires a different set of data entry conventions, the **SCREEN.INPUT** subroutine can be modified to recognize whatever conventions you wish. One occasion in which the subroutine must be modified is when you change the **CANCEL** command or **BACKUP** command strings when setting the configuration options. You must also change the **CANCEL.COMMAND** and **BACKUP.COMMAND** equate statements in the subroutine to reflect your change. **The Programmer's Helper** initially uses an asterisk [*] as the **CANCEL** command which voids a transaction or cancels an update. The caret [^] is used as the **BACKUP** command which allows for movement to the previous field step.

**The SCREEN.INPUT routine can be used generically for any programs that you use to validate data entry input, whether the program was generated by THE PROGRAMMER'S HELPER or not.**

**The Programmer's Helper** may be modified to use a different generalized input routine by changing all of the references to **SCREEN.INPUT** in the "master templates". The master templates are found in the **TPH-PGM-SGMTS** file. Caution should be exercised when performing this kind of change as it will affect the validity of the generated program.

The following will deal with the **SCREEN.INPUT** program parameters:

**DEFAULT** This is the default value to be used if no data is entered at the step prompt. The default is usually used when the data entry operator hits the ↵ key. It should be in internal format.

**LEN** This is the maximum input length allowed.

**CONV** This variable contains the ACCESS conversions or other validation specifications as described in previous sections.

**POS** This variable contains the input position on the screen in the form of the BASIC **@** function. For example, if the position were at column 20 and row 12, the **POS** variable would contain **@(20,12)**.

**HELP** This variable contains the help message to be displayed when a question mark is entered.

**OPTIONS** Other options, such as "R" if required, or "B" if blanks are not allowed.

**RESULT** This variable contains the value of the data entered, after any and all input conversions have been applied to it. For example, if a date were entered as "02/06/89", the **RESULT** variable would contain that date in internal system format.

**STATUS** This variable stands for status code. These codes consist of the following:

0 - Normal Return
1 - Cancel
2 - Exit Sequence Mode
3 - Backup Step
4 - Extended help ?? requested
5 - Repaint Screen (e.g. a TCL command executed.)

## 2.11 Window and Multipage Programs

There exists in **The Programmer's Helper** a special type of data entry program called a "win-

dow", which can be used to create multi-paged screens. This "window" program works in much the same way as a regular data entry program except for three things:

1. Window programs do not perform item ID processing at all. Window programs are called from a main driving program which contains the item ID processing.

2. These programs assume that the data from the main data file has been read in and is available for processing.

3. When an item is filed in a window program, that item is not written to disk, rather, control is passed back to the calling program. The calling program handles all of the file updating.

The following example is intended to help clarify the concept of "window" programs and their use:

> Let's assume that you have a Customer Update program that consists of a large number of fields. Rather than trying to fit a display of all of a customer's open invoices on the main screen, you could build a "window" which would only display open invoices. This "Open Invoice" window could be invoked by typing OPEN at the bottom line screen prompt. The Customer Update program would pass all the customer data to the Open Invoices window without forcing the Open Invoice program to re-read the data.

A window program is defined exactly in the same way as a regular data entry program except for one exception: In the **UPDATE Program Definition** screen, the program type field will replace **ENTRY** with **WINDOW**.

When defining a program that has multiple pages (i.e. Order Entry header and Order Entry Line Items screens), each page is constructed as a separate program. The first, or main, page has a program type of **ENTRY**. The second and subsequent pages have a program type of **WINDOW**. The second and following pages must have as a program name the main program with a page number appended:

1. **UPD.CUST** (main program - entry type)
2. **UPD.CUST.2** (second program - window type)
3. **UPD.CUST.3** (third program - window type)

.
.
.

Each subsequent screen in the above example receives a screen number that is appended to the end of the program name and that is incremented by one for each new screen page.

**Each program must have the appropriate value entered in the "Number of Screens" prompt on the Update Program Definitions Screen.**

# 3 Report Program Development

The philosophy for designing report programs is very similar to that discussed in the previous chapter for data entry programs. That is, to provide a simple, visual representation that resembles the output of a finished program as much as possible. There are, of course, some problems when trying to represent a printed report on a CRT screen. The most obvious of these problems is that most CRT's have a maximum of 80 displayable columns, while many reports contain 132 columns. **The Programmer's Helper** solves this problem by dividing the screen into two (2) "windows", one of which represents the left half of the report, while the other represents the right half.

Once a report format has been defined, a report can be produced by either converting the definition into an ACCESS statement or by generating a BASIC program.

The ability to construct ACCESS statements from visual representations of reports makes **The Programmer's Helper** an ideal tool for people who want to prepare ad-hoc reports but are unfamiliar with the more complex syntax of the PICK retrieval language.

## 3.1 The Development Cycle

As is the case with Data Entry program development, the first step in report creation is to create the file and its dictionary definitions. To access the Report Program section of **The Programmer's Helper**. Select the Report Program Development item from the Master Menu or simply type the following at TCL:

**RPT**
**or**
**RPT <program name>**

If you do not supply a program name, you will be prompted for the name of the program that you wish to define. If the program definition already exists, you will be sent directly to the Report

Program menu. If the program definition is non-existent, you will automatically be taken into the **Report Program Definition** screen (option 1 on the menu). When the report program definition has been entered, the following menu will appear:

```
RPT                    ** THE PROGRAMMER'S HELPER **                    V3.2
                       Report Program Development


                  1. UPDATE Program Definition Item
                  2. UPDATE Report Definition
                                                          PGM ID:
                  3. DEFINE Code Inserts                 11.

                  4. EXTRACT File Test Data                FILE NAME
                  5. RUN Sample Report                   12.

                  6. GENERATE Report Program

                  7. UPDATE File Dictionary

                  8. EDIT Generated Program
                  9. RUN Generated Program
                 10. CHANGE To A Different Program


                        Enter Selection:

                                                  !gêbÖLh θü A♦ƒ#(è ◄ ▲▲^äèIuáC
àèA e á¶"¶  @(E(♣0ÄQèQ a í¶¬¶┬ AŸ C(AêÉ2ä5∞9Ä#áe╚p •⌐e╚0 ♥áu∞ å!Ç4n ├ ♦►&ìπ┐
```

**Report Program Development Menu**

The following sections will deal with each of the **Report Program Development** menu options.

### 3.1.1 Update Program Definition

The program definition item contains the name of the file being used to prepare the report, the report title, and the file name in which the generated program will be stored. Two description

lines may also be entered to keep track of which program does what.

```
UPD.PD.RPT           *** UPDATE PROGRAM DEFINITION ***

      Program Name:  PARTS-01

   1. Program Type:
   2. Program Title:
   3. Prgram Desc:


   4. Data File:
   5. Source File:
   6. Indent:
   7. PROC File:

--------------------------- Other Files ----------------------------------------
 LN     FILENAME       DESCRIPTION                      ITEM ID ATTR      USE
 8.1
 8.2
 8.3
 8.4


 DI=Update File Dictionary
```

**UPDATE Report Program Definition Screen**

The **UPDATE Program Definition Screen** is comprised of the following fields:

**Program Name** — This is the name of the report program that will be used to invoke the report.

**Program Type** — In the case of a report program, the program type will always be **REPORT**.

**Program Title** — This is the title that will appear at the top of each page in the report.

**Program Description** — This is a two line description that is used for internal documentation purposes. It lets you know what this report is being used for. This field is optional.

| | |
|---|---|
| **Data File** | Used to define the name of the main data file from which information is to be retrieved. For example, if you wanted to create a report of customer names and addresses, you would enter **CUSTOMER** as the data file. |
| **Source File** | This is the name of the file in which the generated source code will be placed. |
| **Indent** | This is used to specify how many spaces to indent in the actual generated source code. |
| **PROC File** | This is the name of the file used to store the PROC that will run the generated ACCESS program. |
| **Other Files** | This is for cross-reference and documentation purposes, and works the same way as the **Data Entry Program Definition**. See Section 2.2 for more details. |

### 3.1.2 UPDATE Report Definition

Once the program definition has been created and entered, the next step is to construct the report format or Report Definition Item. This is accomplished via a program that attempts to show you, on the screen, what the finished report will actually look like. This function will be discussed in greater detail in a subsequent section.

### 3.1.3 Extract Test Data

After the Report Definition has been created, it is now time to test the format, or "look" of the report. This is obviously not very efficient when dealing with very large files in a database.

**The Programmer's Helper** solves this problem by offering a very convenient utility that allows for the testing of a report quickly and without having to go through a complete file and having to print a large report.

The **EXTRACT Test Data** function will randomly select records from the data file designated so that a small report can be generated and tested. **THE PROGRAMMER'S HELPER** will prompt you for the number of items that you want in your list. Once the number of items desired has been entered, a SELECT list will be generated and saved for later use.

### 3.1.4 RUN Sample Report

Before actually generating a BASIC program, it is generally a good idea to test the layout of the report. This **RUN Sample Report** function will convert the report definition into a PICK ACCESS or ENGLISH statement, and run a report using the sample data extracted in the **EXTRACT Test Data** function.

The statement constructed will be stored in the PROC file defined in the **UPDATE Program Definition** item with the program name as its ID. Once generated, this PROC may then be customized as desired or needed.

### 3.1.5 Define Code Inserts

This function is the same in nature as the one used in the Data Entry Program Development. It allows you to insert special or customized code within your generated BASIC programs in a number of places. This feature allows you to modify and regenerate your report program definition frequently without losing any of the customization that has been done.

### 3.1.6 Generate Report Program

Once the sample report has the desired appearance, the final step is to generate the BASIC program. Running this feature will also construct the appropriate PROC and place it in your PROC file with the program name as the item ID. Your BASIC program as well as the generated PROC can now be further customized, if desired, to match any specific requirements that you may have.

### 3.1.7 Other Menu Options

The remaining menu options on the **Report Program Development** menu are exactly the same in nature as those found in the **Data Entry Program Development** menu. To reference these, please consult the following sections:

**UPDATE File Dictionary - see Section 2.1.7**
**EDIT Generated Program - see Section 2.1.16**
**RUN Generated Program - see Section 2.1.17**
**CHANGE Programs - see Section 2.1.18**

## 3.2 Editing the Report Format

When defining a new report format, **The Programmer's Helper** will prompt you for a list of attribute names that will be included in the report, the function of each attribute (i.e. is it a control break, a TOTAL field, or simply a field to print), and whether it is a single or multivalued attribute. If you are editing an existing report format this step will be skipped.

The next step will show an approximation of the printed report displayed on your CRT screen. This screen display consists of numbered columns of information, each corresponding to an attribute. A partial example is shown below:

| | | |
|---|---|---|
| **Column Number** | **1.** | **2.** |
| **Column Heading** | **CUSTOMER NUMBER** | **CUSTOMER NAME** |
| **Field Pattern** | **999999** | **XXXXXXXXXXXX** |
| **Field Type** | **(BRK)** | **(PRT)** |
| **Attr Name** | **CUST-NO** | **CUST-NAME** |

Following is a brief description of each of items that are defined for each attribute in a report layout:

**Column Heading** This field is comprised of up to three (3) lines of the column heading as defined in attribute 3 of the dictionary definition item.

**Field Pattern** This is a pattern indicating the type of data that will be printed in the column. Fields with no conversion will display a string of "X's" to fill the column. Numeric fields with an **MD** or **MR** conversion will display as all "9's" with the appropriate decimal place. Date fields will be displayed as "09/09/99" or "09 AUG 99" according to the date conversion specified.

For example, if the customer name in the customer file were right justified with a length of 30, the screen format would look like this:

**CUSTOMER NAME**
**XXXXXXXXXXXXXXXXXXXXXXXXXX** <- 30 "X's"

If the year to date sales for a customer were 10 long and with 2 decimal places, the screen format would display:

**YTD SALES**
**9999999.99** <- 10 positions, 2 decimal places

**Field Type** This indicates the type of field that will be printed. There are three (3) available types of fields. These are:

1. **BRK** - used for control breaks
2. **TOT** - used for fields that need totalling
3. **PRT** - used for any fields that are to be printed

Additionally, all multivalued fields will be flagged as follow:

1. **BRK***
2. **TOT***
3. **PRT***

The asterisk (*) that is appended to the field type denotes that the field is a multivalue.

**Attribute Name** This is the attribute name found in the file dictionary for the data file being used. If the column width is shorter than the attribute name, the name will be truncated the amount of length necessary for it to fit properly.

The Report Screen formatter also supplies a number of commands for editing purposes. These are:

**INSERT** Entering an [I] at the command prompt will allow you to insert a new field before an existing field.

**DELETE** Entering a [D] will allow you to delete a field.

**APPEND** Entering an [A] will allow you to add a field to the end of a report.

**SWAP** Entering an [S] will allow you to exchange the position of two fields.

**CANCEL** Entering an asterisk [*] will cancel any changes made to the report format and return it to the way it was before the EDIT function was selected. The cancel command is similar in nature to the **EX** command in the PICK EDITOR.

**FILE** Entering an [F] will cause all changes to be saved and will return you to the **Report Program Development** menu.

## 3.3 Program Templates

As with the data entry programs, the report program is constructed by combining a number of program "templates" from the **TPH-PGM-SGMTS** file. Following is a description of the templates used for building report programs:

```
                    *** DEFINE CODE INSERTS ***

 1. Start of Program
 2. Open Files
 3. Define Constants
 4. Initialize Control Breaks and Totals
 5. Read Records Routine
 6. Start of Main Processing Loop
 7. Close of Main Processing Loop
 8. Final Totals
 9. Print Routine
10. User Written Subroutines









 Enter #, ATTR NAME, ? For Attributes, ?C for Code SGMTS:
```

**Defining Code Inserts for Reports**

| | |
|---|---|
| **Standard Program Start** | This code segment contains the first 25 or so lines of the report program. Standard constants such as AM, VM and so on are defined here. |
| **Standard File Opens** | This template opens the main data file and any other files that are required by the program. |
| **Program Constants** | This template is used to define any unique constants. The **EQUATE**s for the primary file are appended to this template. |
| **Initialization** | This template is where the break values and totals are initialized. Any special initialization that is required should be placed in this code segment. |
| **Read Records** | This template is where the primary data item is read. |
| **Start of Loop** | This is the first, or top half, of the major program loop that gets the next item ID in sequence from a SELECT list. |
| **Close of Loop** | This is the bottom half of the major program loop. It is primarily the **REPEAT** statement for **LOOP...UNTIL** found at the "Start of Loop" template. |
| **Program Finish** | This template is where any special end of file processing should occur. |

## 3.4 Replacement Variables

When editing or constructing code inserts, certain values may be expressed as "replacement" variables which are evaluated when the program is generated.

These replacement variables fall into two broad categories:

1. STATUS variables, and
2. STEP variables.

Status variables are those that are global and applicable throughout the program. For example the data file name, or the record name. Step variables, on the other hand, are valid only within a report field step and their values will change from step to step.

All of the replacement variables described in **Section 2.6** are valid for use with reports. Some of them, however, will contain a null value (such as **%UPDPGM**, **%XREF**, etc). There is one replacement variable that is found in the report section but not in the data entry section. That variable is:

**%READNAME** **Read Attribute Item ID Name**. This variable contains the name of the attribute that will be used as an item ID when performing a file translate. If the attribute being used as the item ID does not appear on the report, it will have a value of "**%RECORD(%AMC)**"

# 4 FORMS Program Development

A "forms" program is a cross between a data entry screen and a regular report. The form layout is defined in much the same way that a data entry screen would be laid out by using the screen painter (or PICK EDITOR). Instead of writing a data screen, however, it generates a report that uses an entire page for each item processed. Some of the more common examples of forms programs would be: packing lists, invoices, purchase orders, work orders and so on...

## 4.1 The Development Cycle

As when developing data entry programs, the first step is to create the file and its corresponding dictionary items. To access the Forms Program section of **THE PROGRAMMER'S HELPER**, select the **FORMS Program Development** item from the Master Menu, or simply type the following statement at TCL:

**>FORMS**

**or**

**>FORMS <program name>**

If a program name is not supplied, you will be prompted for the name of the program that you wish to define. If the program definition already exists, you will be sent directly to the forms menu. If this program was not previously designed, you will be sent directly to the **UPDATE Forms Definition** screen (option 1 on the forms menu).

The following figure shows what the **FORMS Program Development Menu** looks like:

```
FORMS                  ** THE PROGRAMMER'S HELPER **                  V3.2
                         Forms Program Development


                  1. UPDATE Program Definition Item
                  2. UPDATE Forms Definition

                  3. COMPILE Forms Definition

                  4. DEFINE Code Inserts                          PGM ID:
                  5. GENERATE Forms Program

                  6. UPDATE File Dictionary                       FILE NAME:

                  7. EDIT Generated Program
                  8. RUN Generated Program

                  9. CHANGE To A Different Program


                        Enter Selection:
```

**FORMS Program Development Menu**

In the next few sections we will deal with the options that are available on the **FORMS Program Development Menu**.

### 4.1.1 UPDATE Program Definition

The Program Definition Item, as in all the other **TPH** tools, contains the name of the file name used to prepare the report, the report title, and the name of the source file which will contain the generated code. The following screen is an example of a typical Forms Program Definition Item:

```
UPD.PD.RPT           *** UPDATE PROGRAM DEFINITION ***

      Program Name:  PARTS-01

   1. Program Type:
   2. Program Title:
   3. Prgram Desc:


   4. Data File:
   5. Source File:
   6. Indent:
   7. PROC File:

--------------------------- Other Files ----------------------------------------
 LN     FILENAME       DESCRIPTION                       ITEM ID ATTR      USE
 8.1
 8.2
 8.3
 8.4


 DI=Update File Dictionary
```

**UPDATE FORMS Program Definition Sample Screen**

Section 3.1.1, **UPDATE Program Definition**, describes the use of the **UPDATE Program Forms Definition** program since it is exactly the same one.

## 4.1.2 UPDATE Forms Definition

As in all of **The Programmer's Helper** tools, once a program has been defined, the next step is to define (paint) the layout for either the screen or the report. The same screen painter is used throughout **THE PROGRAMMER'S HELPER**. The format for each step in a FORMS program is exactly the same as that used in Data Entry Programs. The only exception is that field validations such as **Required Field**, **Date Stamp**, etc. do not apply to item used in FORMS.

### 4.1.3 COMPILE Forms Definition

This is essentially the same exact function as **COMPILE Screen Definition** in the Data Entry Development Menu. Its purpose is to ensure that all your attribute names are valid, and that there are no missing steps.

### 4.1.4 DEFINE Code Inserts

This function allows you to generate special, or customized, code into the generated BASIC program in a number of places. This feature allows you to modify the forms definition without losing any of the customized features that you may have added.

# 5 MENU Program Development

Once all the data entry screens, reports and forms have been designed, the ideal situation is to call these from a menu. Menus make it much easier on the end user and do not force them to remember all the program names at TCL when they need to be run.

Programs generated by **The Programmer's Helper** can easily be placed into either its own menu format, or you can use whatever menuing system you own to easily access your work. When you use **The Programmer's Helper** menu utility, it creates a simple menu program that can later be modified as per your particular needs.

## 5.1 The Development Cycle

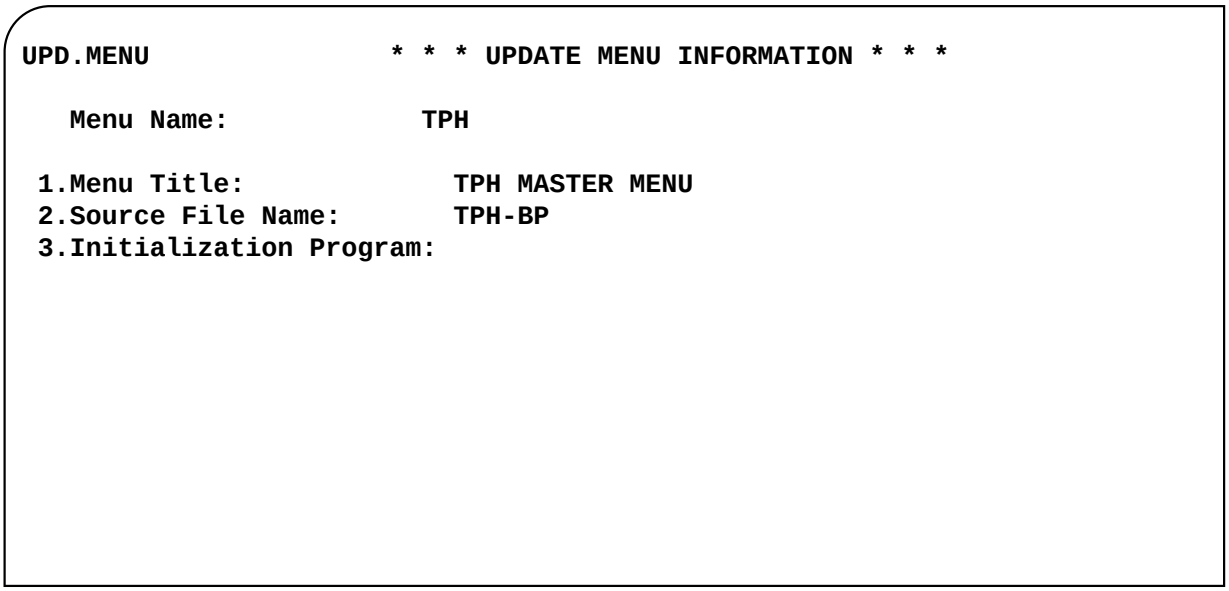
To access the Menu Program Development portion of **The Programmer's Helper**, select the **Menu Program Development** item from the Master Menu, or type one of the following statement while at TCL:

**MENUS**

**or,**

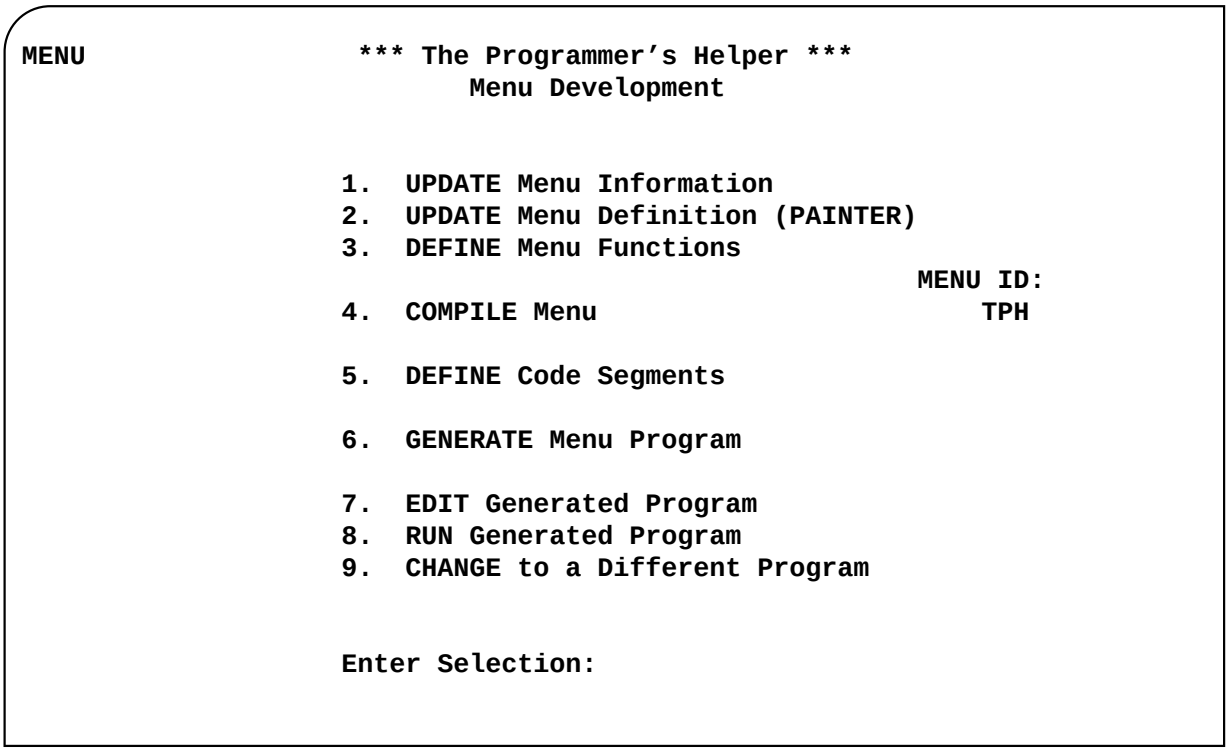
**MENUS <menu-name>**

If a menu name is not specified, you will be asked for the name of the menu that you wish to define. If the menu definition already exists, you will be sent directly to the Development Menu. If the menu was not previously defined, you will be taken directly to the **UPDATE Menu Definition** function (option 1 on the menu). The following figure shows the **UPDATE Menu Definition** screen:




**UPDATE Menu Definition Screen Sample**

Once the menu has been correctly defined, the following menu will appear:




**MENU Development Menu**

The **MENU Development** menu is composed of the following sections:

### 5.1.1 UPDATE Menu Information

This screen, also known as **UPDATE Menu Definition**, is where you tell **THE PROGRAMMER'S HELPER** the title of the menu, the name of the file in which the generated program program will be stored, and the name of an optional Initialization subroutine.

The first two options on the menu are self-explanatory. The third option, Initialization program is executed when the menu is first entered. This program can be used to set up PROC buffers, handle security, or any other functions that your design may call for before allowing user access to the menu.

### 5.1.2 UPDATE Menu Definition

Menu layouts are created using this option in much the same way as are data entry screens and forms. You can use either the PICK EDITOR or the screen painter to develop your menus. See section 2.9 for more information on using the screen painter.

The Menu Definition Item looks very much like the Screen Definition Item used for data entry screens. It is comprised of a number of "steps", with each step representing one menu function. The format of each step is as follows:

**NN.TEXT ;FUNCTION-ID**

or, for example,

**1.UPDATE CUSTOMER FILE ;UPD.CUSTOMER**

The commands for the Menu Definition item are listed below:

**NN.** This represents the step number. When the user types the menu step (or option) number, the corresponding function will be executed.

**TEXT** This is the description of the function that is displayed on the menu screen for this step.

**;** The semi-colon marks the end of the **TEXT** section. Unlike with screen data entry programs, the position of the colon, except in step 0, has no significance.

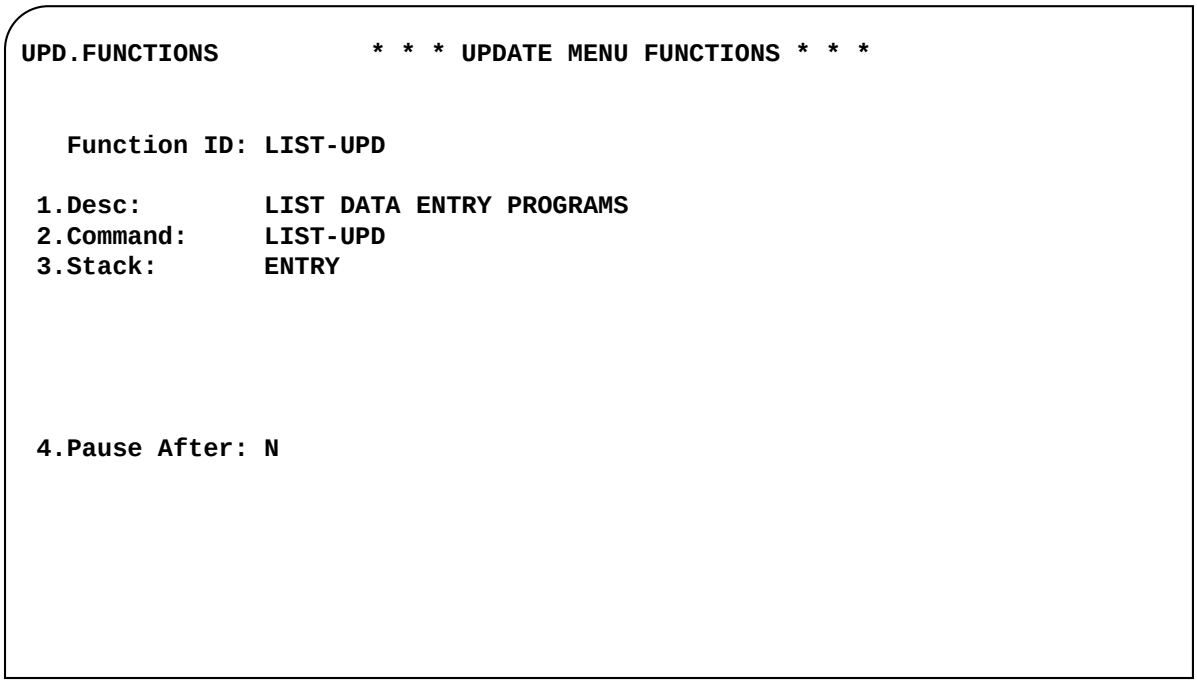
**FUNCTION-ID** This parameter specifies the name of an item in the **TPH-FUNCTIONS** file that contains information on what commands to execute when this menu option is pressed. The definition of these items is described in the next section.

**FOOTNOTES** The **"footnotes"** described in section 2.2 can also be used in the menu creation process to put information on a second page is there is not enough room on the first page.

**Step ZERO** The step numbered with a zero has a special significance in the Menu painter. It is used to denote the text that will be displayed before the user inputs a selection. For example; **Enter Selection: ;** the semi-colon marks where the input will take place. Step Zero in Menu Development is usually placed at the bottom of the menu screen.

### 5.1.3 DEFINE Menu Functions

When designing the Menu screen you associate a menu number with a menu function. The Menu Function contains information about the command to be executed by the menu program.




**Update Menu Functions Screen**

**Function ID** This is the name of the function that was entered when designing the menu definition. It is the ID of the item in the TPH-FUNCTIONS file

**1.Description** This is a line of descriptive text describing what the function does. It is for reference only.

**2.Command** This is the command that will be executed when the menu item is selected. You should enter it exactly as if you were entering it at TCL.

**3.Stack** This step contains any optional parameters that are required by the command. This is similar to the "stack" in PROC and the DATA statement in BASIC.

**4.Pause After?** If you enter a Y in this field the menu program will pause after the command has been executed and will wait for the user to press RETURN before redisplaying the menu.

### 5.1.4 COMPILE Menu

This is essentially the same exact function as **COMPILE Screen Definition** in the Data Entry Program Development Menu. Its purpose is to insure that all of your menu functions have been defined and that you do not have any duplicate or missing steps.

### 5.1.5 DEFINE Code Inserts

This function is similar in nature as the one used in the Data Entry Program Development. It allow you to insert special or customized code within your BASIC programs in a number of places. This feature allows you to modify and regenerate your menu program without losing any of the customization that has been done.

### 5.1.6 GENERATE Menu Program

This option simply generates the menu program once the menu has been defined and painted. The menu program can later be modified for any specific uses that may not have been covered by **The Programmer's Helper**.

### 5.1.7 Other Menu Options

The remaining menu options on the **Menu Program Development** menu are exactly the same in nature as those found on the **Data Entry Program Development** menu. To reference these, please consult the following sections:

| | |
|---|---|
| **EDIT Generated Program** | **- see Section** 2.1.16 |
| **RUN Generated Program** | **- see Section** 2.1.17 |
| **CHANGE Programs** | **- see Section** 2.1.18 |

## 5.2 Program Templates

As with the other section of **TPH**, the menu program is constructed by combining a number of program "templates" from the **TPH-PGM-SGMTS** file. Following is a description of the templates used for building menu programs:

```
                  *** DEFINE CODE INSERTS ***

 1 Start of Program
 2 Open Files
 3 Initialization
 4 Main processing Loop
 5 User written Subroutines
 6 Display additional data
 7 Calling PROC









Enter #, Attr Name, ? for Attributes, ?C for Code Sgmts:
```

**Defining Code Inserts for Menus**

| | |
|---|---|
| **Standard Program Start** | This code segment contains the first 25 or so lines of the menu program. Standard constants such as AM, VM and so on are defined here. |
| **Open Files** | This template opens any files that may be required. |
| **Initialization** | This template calls the initialization subroutine and performs any other required initialization. |
| **Main Processing Loop** | This template is the heart of the program. It displays the menu format, accepts the users input and executes the appropriate command. |
| **User Written Subroutines** | The template is normally empty. You can add any special processing subroutines you wish. |
| **Display Additional Data** | This subroutine is where you can display on the menu screen any other information you wish, such as time/date, port number, user ID, etc. |
| **Calling PROC** | Because standard PICK BASIC cannot directly access the TCL command line, if you have parameters you wish to pass to the menu, they have to be passed through a PROC. This PROC normally just calls the menu program though you can add other features if you like. |

# 6 Index File Development

The PICK system is very good at retreiving data if the exact item ID is known. However, it does not perform as well when only part of the ID or the value of a particular attribute is known and you want to retrieve the entire item. Standard PICK must scan thru the entire file to retreive the desired items. To get around this problem many programmers use schemes to reorganize the data into a form that allows more rapid access. The two most common schemes are inverted files and trees.

The traditional cross reference is an example of an inverted file. The item ID's of all items with a specific value are placed in a list. For example, a Zip Code file could contain a multivalued attribute with the customer ID's for all customers in a particular Zip Code. Knowing the Zip Code instantly gives you a list of customers from which to choose. This technique is the simplest to immplement and usually requires the least amount of overhead. It is also works well when you have to combine many selection clauses (e.g Zip code "90292" or "90326" and with salesman "100"). It works less well if you have partial searches (zip code "90XXX").

The other popular technique is to build a "tree" structure. The keys are stored in sorted order in a way that lets the program quickly determin if a particular record exists, or to retrieve all records in a specified range. Without getting too technical, trees work best when they are "balanced". That means that to find a particular item should take about the same amount of time it takes to find any other item. This is analagous to the selection of the correct modulo when creating a standard PICK file. Luckily, there is a technique that will automatically balance a tree when items are added or deleted. These trees are called "Balanced Trees" or "B Trees." (By the way, the technique was invented and refined in the sixties so it has been around for a long time.). **The Programmer's Helper** uses a variation of this technique that is well suited to the PICK system.

## 6.1 The Development Cycle

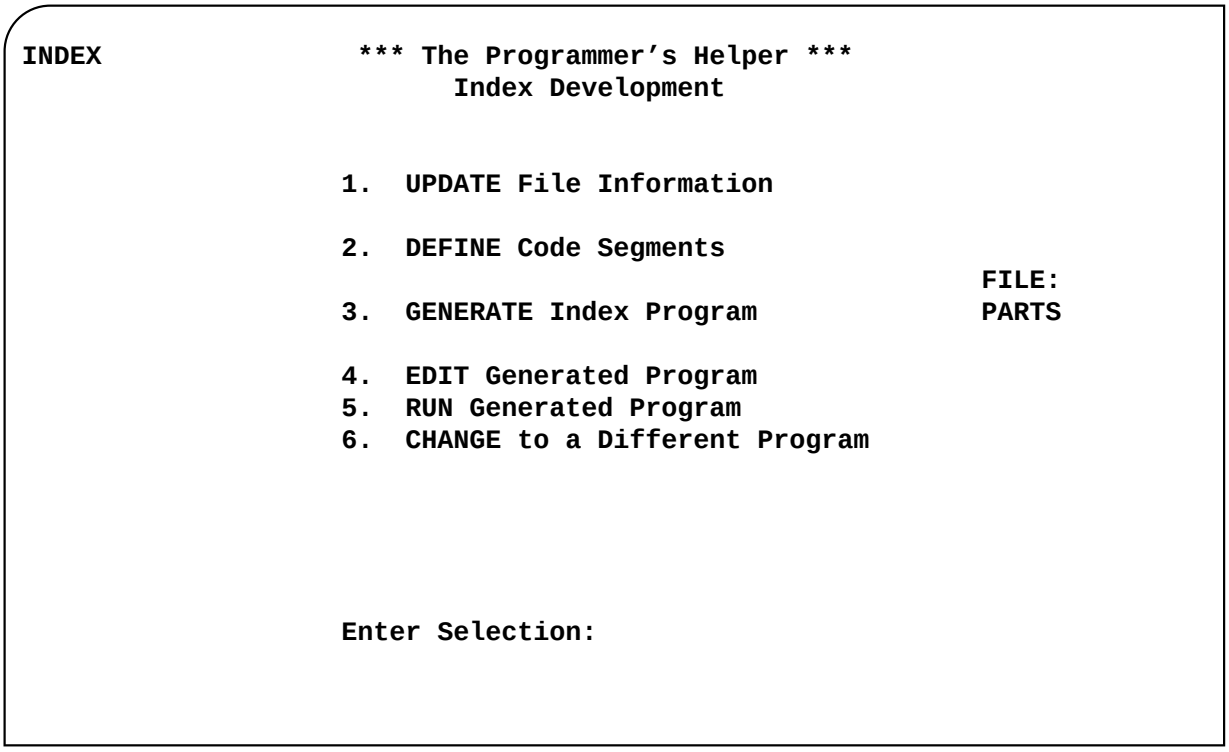
To access the Index Program Development portion of **The Programmer's Helper**, type the following statement while at TCL:

**INDEX**

or,

**INDEX <file-name>**

If a file name was not specified you will be asked for the name of the file for which you wish to define indexes. If the file does not exist or it has not be define for use with **The Programmer's Helper**, you will be taken directly to the **UPDATE File Definition** screen described in section 1.3. You will receive the following menu:




The menus is composed of the following functions.

### 6.1.1 UPDATE File Information (Option 1)

This is the same as the one found on the Main Menu. It is described fully in section 1.3. One of the choices from the bottom line menu is **Index** which will display a special window for entering information about the index file being created.

```
INDEX.WIND            *** DEFINE FILE INDEXES ***

   Data File Name: PARTS

 1.Index Source File Name:  BP

------------------------Attributes to be indexed----------------------------
  ln   Attr Name           Conversion               Options
 2.1   PART-NO
 2.2   MODEL-NO
 2.3   DESC               MCU                      MW
 2.4   STATUS
 2.5

---------------------Attributes to display on Browse Screen----------------
  ln    Attr Name
 3.1    PART-NO
 3.2    MODEL-NO
 3.3    DESC
 3.4    NEW-LIST-PR
 3.5    STATUS
```

Each section of the screen is described below:

Index Source File Name

This portion of **The Programmer's Helper** will create two programs for each file being index. The first program is a subroutine that will update the B-TREE whenever a record in the data file is updated or deleted. The second program will rebuild the entire index from scratch. The Source File Name is the name of a BASIC program file in which these two programs will be written.

Attributes to be indexed

This multivalued window lists all of the fields that will be indexed (and therefor the fields that can be searched). The columns are:

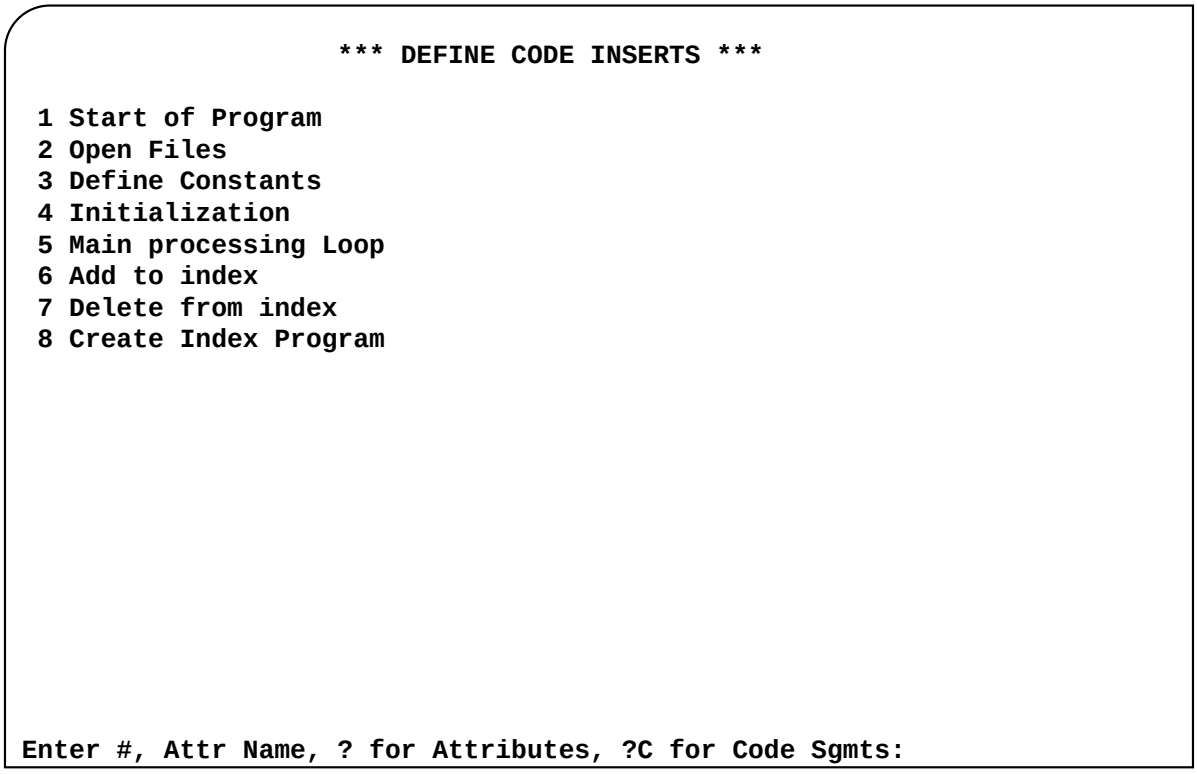
a. Attribute name. This is the name of the attribute as defined in the file dictionary. It should be a "simple" attribute, (i.e. no Correlatives).

b. Conversion. This is a conversion that will be applied to the data before the index is built. Normally it will be "MCU" (Upper Case) for text fields and "MCN" (Numerics Only) for numeric fields.

c. Options. There are only two options supported at this time. If this field contains a "W" the index will be built on each word (separated by spaces) in the field. A "M" signifies that the attribute is multi-valed and that each value should be indexed. Both options may be combined as "MW".

Attributes to display

When peforming a search, a number of attributes from the data file will be shown on a selection screen. The attributes that will be shown are defined here. You may want to use Synonym dictionary items with a shorted length in order to fit all required fields onto a single line.




### 6.1.2 DEFINE Code Segments (Option 2)

This function is the same in nature as the one used in the Data Entry Program Development section of TPH. It allows you to insert special or customized code within your generated BASIC program.

### 6.1.3 GENERATE Index Program (Option 3)

Once the indexes have been defined, this function is used to write the programs that will update the B-TREE index. Two programs are actually written: The first is a subroutine that determines any changes to indexed fields and calls the B-TREE updater for each change. The second program clears the indexes and then rebuilds them. It does this by calling the subroutine for each item in the data file.

The two programs are named **BUILD.filename.INDEX** and **CREATE.filename.INDEX** where **filename** is the name of your data file, with any special characters changed to periods to make it a valid BASIC variable name.

### 6.1.4 Other Menu Options

The remaining menu options on the **Index Development** menu are exactly the same in nature as those found in the **Data Entry Program Development** menu. To reference these, please consult the following sections:

**EDIT Generated Program - see Section 2.1.16**
**RUN Generated Program - see Section 2.1.17**
**CHANGE Programs - see Section 2.1.18**

# 7 TUTORIAL

By now you are pretty eager to begin using **The Programmer's Helper** in a productive way. This tutorial is designed to teach you fundamentals of **THE PROGRAMMER'S HELPER**. After completing this, you will be able to do the following:

| |
|---|
| Create and define files |
| Create Data Entry Screens |

This tutorial will assume that you have a rudimentary knowledge of the PICK Operating System, and so we will not delve extensively into discussions on file structures, dictionary definitions, programming concepts, etc.

Well enough talk, let's get started!

## 7.1 Setting up TPH

We will assume that you have set up **The Programmer's Helper** by following the instructions in Appendix A.

## 7.2 Getting Started

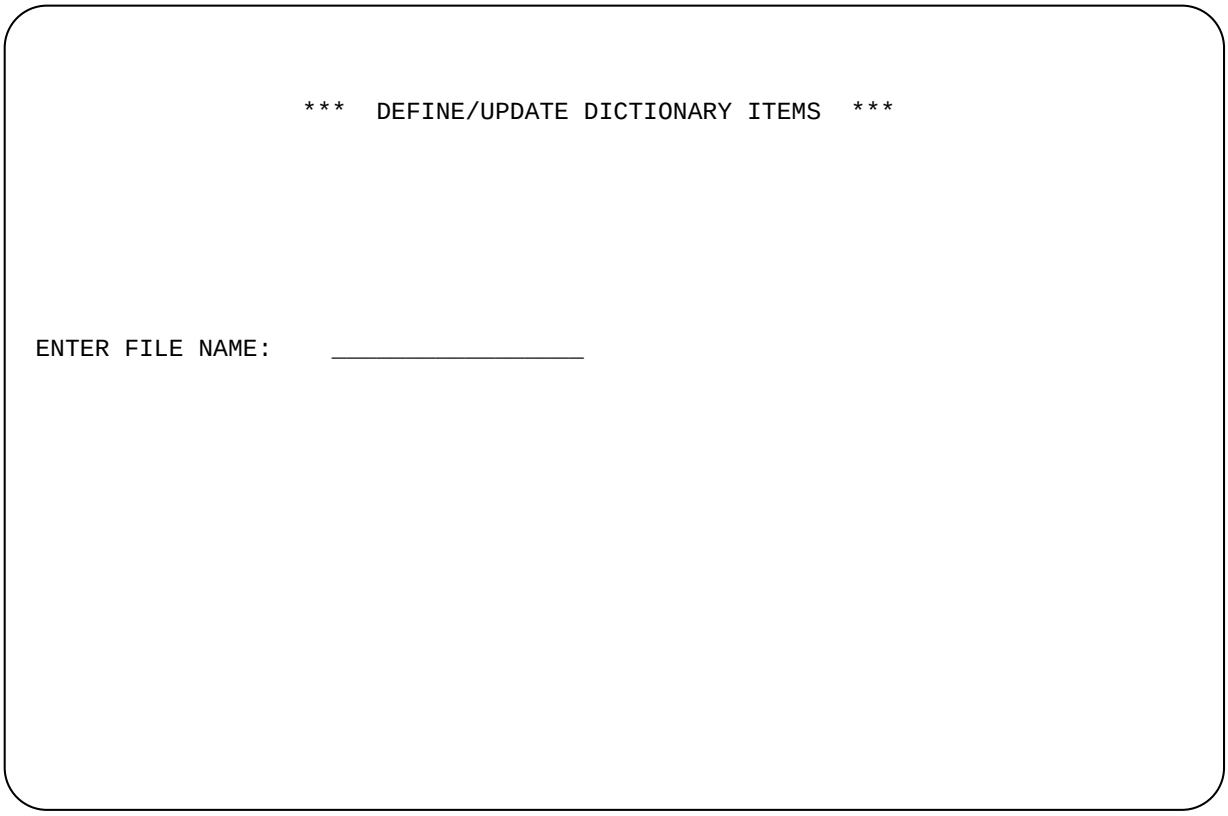
Now log on to the TPH account (usually **TPH4**). The TPH Master Menu will be displayed. Press the RETURN key and you will be taken to TCL. It's time to create some files and some dictionaries so that we can start using **The Programmer's Helper**.

We now want to create files and dictionary items for our application. Let's start by using the **CREATE-FILE** verb. Please type the following:

CREATE-FILE CUSTOMER 3,1 13,1 ↵

This file will be used later oni when we construct a program. Now let's create some dictionary items to define the CUSTOMER file. From TCL type:

BUILD-DICT ↵




At the "**ENTER FILE NAME**" prompt enter the following CUSTOMER ↵. The following screen will now appear:

```
BD               * * *  BUILD DICTIONARY ITEMS  * * *

  File Name:
  Attr Name:

  1.Attribute Type:
  2.Attribute Number

  3.Column Heading



  4.Conversion:
  5.Correlative:

  6.Justification
  7.Maximum Length           10

   Help Message
  8.
```

You are now ready to define the **CUSTOMER** file items. Enter the following attribute information using the dictionary creation program:

| Item | AMC | HEADING | CONV | JUST | MAX |
|---|---|---|---|---|---|
| **CUST-NO** | 0 | CUST NO. | | R | 6 |
| **NAME** | 1 | NAME | | L | 35 |
| **ADDR** | 2 | ADDR | | L | 25 |
| **CITY** | 3 | CITY | | L | 15 |
| **STATE** | 4 | ST | | L | 2 |
| **ZIP** | 5 | ZIP | | L | 10 |
| **PHONE** | 6 | PHONE | | L | 15 |
| **SINCE** | 7 | CUST SINCE | D2/ | R | 8 |
| **YTD-PURCH** | 8 | YTD PURCH | MR2 | R | 10 |

Good work! Now before we can use **The Programmer's Helper** to create a program, we need to define the data files and create a program description. You're doing real well so let's move right along...

## 7.3 Defining Files

In order to define a file, we must be at **The Programmer's Helper** main menu. Let's do this now. At TCL type the following:

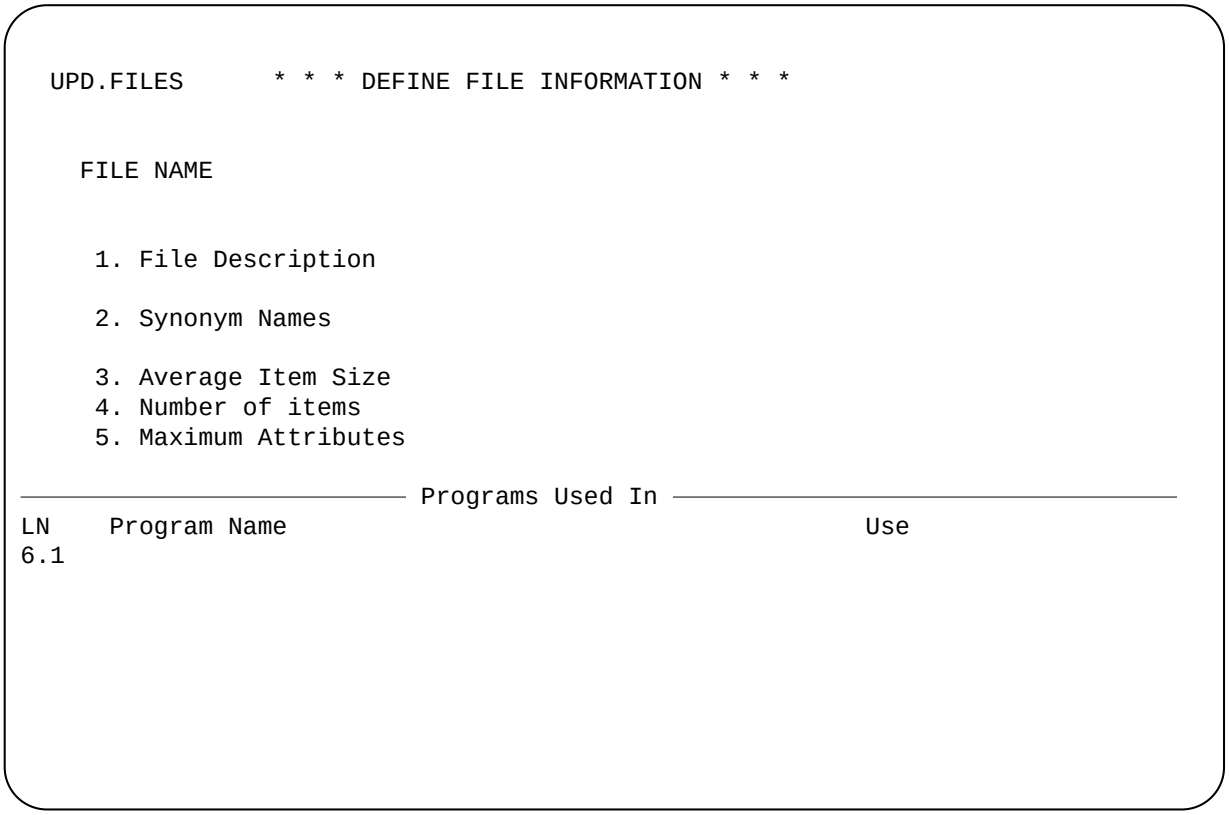
TPH ↵

This command calls up **The Programmer's Helper** main menu. The following illustration gives an example of what the main menu looks like:

```
TPH                  ** THE PROGRAMMER'S HELPER **
                        MASTER DEVELOPMENT


                     1. SET System Constants
                     2. DEFINE Data Files
                     3. LIST Data Files

                     4. DEFINE Program
                     5. LIST Data Entry Programs
                     6. LIST Report Programs
                     7. LIST Forms Programs

                     8. Data Entry Program Development (UPD)
                     9. Report Program Development (RPT)
                    10. Menu PROC Development (MENU)
                    11. Forms Program Development (FORMS)



                      Enter Selection:
```

At this point, before we can create any kind of data entry screen, we need to define the data file. Select option 2 by typing a 2 at the main menu prompt.

If you made the right choice, the following screen should have appeared:




Now let's fill in the prompts as follows:

FILE NAME - enter CUSTOMER ↵

File description - enter CUSTOMER MASTER FILE ↵

Synonym names - enter CUST ↵

Average item size - enter 150 ↵

Number of items - enter 100 ↵

Maximum Attributes - enter 40 ↵

The rest of the screen can be left blank.

Now enter [F] to file the screen. Great, you've just defined the **CUSTOMER** file and it did not take very long at all. Now let's move on.

## 7.4 Defining Programs

After the particular data files you need to use have been defined, the next step is to define the actual program that you will later on be generating. Again, this is a simple step.

From **The Programmer's Helper** main menu select option 8, **Data Entry Program Development**.

At the main menu prompt type [8 ↵].

You will see the following screen:

```
UPD                *** THE PROGRAMMER'S HELPER ***                    V3.2  Ins
                   Data Entry Program Development




 Enter Program Name:   ____________________
```

At the **Enter Program Name:** prompt please type the following:

```
UPD.CUSTOMER ↵
```

If the program already exists you will be taken directly to the **Data Entry Program Development Menu**. Since this is a new program, **The Programmer's Helper** will display the following screen:

```
UPD.PD             *** UPDATE PROGRAM DEFINITION ***

 Program Name:
                                                    8. Indent
 1. Program Type:                                   9. # Screens
 2. Program Title:                                 10. Large Pgm
 3. Program Desc:


 4. Data File:
 5. Source File:
    Pgms Called:    6.1
                    6.2
                    6.3

──────────────────────────────── Other Files ──────────────────────────────────
LN   FILENAME    DESCRIPTION                      ITEM ID ATTR      USE
7.1
7.2
```

The **UPDATE Program Definition** screen allows **The Programmer's Helper** to keep track of programs that have been developed, and is also used for documentation purposes when the documentation is generated.

Now let's fill in the screen prompts so that we can continue with our development.

1. Program type - enter ENTRY ↵

The system will display the following message directly after the "ENTRY" prompt: **DATA ENTRY PROGRAM**.

2. Program Title - enter **UPDATE CUSTOMER MASTER FILE** ↵

This is the title that will appear centered at the top of the actual data entry screen.

3. Program Desc - type the following text

> **This program is used to update the** ↵
> **customer master file** ↵

The next prompt asks for the name of the data file that will be used by this program. This prompt refers to the main data file to be updated. In this case we want to update the CUSTOMER file. So let's type the following:

4. Data File - enter CUSTOMER ↵

After you pressed ↵, you should notice that the following description was printed next to the file name: **"CUSTOMER MASTER FILE"**. This is the description from the FILE DEFINITION screen that you filled out in the previous step.

The next prompt asks for the Source file. This is the name of the file where the program you are going to generate will be placed. Enter the following:

5. Source File - enter BP ↵

The other remaining fields are left blank, so just press the ↵ key until you are at the bottom line prompt. Type F ↵ to file the screen.

## 7.5 Creating Your First Program

After defining your program, **The Programmer's Helper** displays the **Data Entry Program Development** Menu. It looks like the following screen:

```
UPD                    ** The Programmer's Helper **                    V4.0
                       Data Entry Program Development

 1.   UPDATE Program Definition Item
 2.   UPDATE Screen Definition

 3.   PREPROCESS Screen Definition                          PGM ID:
 4.   DEFINE Code Inserts                                    UPD.PARTS
 5.   GENERATE Data Entry Program
 6.   UPDATE Step Detail                                  FILE NAME:
                                                           PARTS


 7.   UPDATE File Dictionary            13.   EDIT Documentation
 8.   AUTO-BUILD Screen                 14.   CREATE Documentation
 9.   UPDATE Table Definitions          15.   PRINT Documentation
10.   UPDATE Named Patterns             16.   EDIT Generated Program
11.   UPDATE Data Types                 17.   RUN Generated Program
12.   SET Configuration Options         18.   CHANGE to a Different Pgm


  Enter Selection:
```

Since we have already completed Option 1, in the previous section, let's design our screen to enter the customer information now.

Screens can be painted, or created, in one of two ways: 1) through the PICK EDITOR utility, or 2) through **The Programmer's Helper** screen painter. The screen painter is, in essence, a one page word processor. There is also a third way in which to design a screen, and it is probably the simplest of the three: let **The Programmer's Helper** do it for you. In fact, why don't we try that now.

At the **Enter Selection** prompt, select Option 10 to AUTO-BUILD the screen:

Enter Selection - type [8]

The following screen will be displayed:

```
AUTO-BUILD          *** THE PROGRAMMER'S HELPER ***                    V3.2
                    Build Screen From Dictionary


WARNING:
   If you continue it will destroy your current screen definition.



   Choose one of the following:
        A for all 'A' type attributes in the file.
        E to enter the attribute names one at a time.
        * to cancel this function


  Selection:
```

For our tutorial purposes, let's choose option "A".

Selection - type [A ↵]

The screen will be replaced by another screen which will display the items that have been selected for the data entry screen. You should now have this on your screen:

```
SCREEN ID:  UPD.CUSTOMER                * Define Fields *

ln    Sel   Attr Name    Prompt          Validation    Options
1.1    Y    CUST-NO      Customer Number
1.2    Y    NAME         Name
1.3    Y    ADDR         Address
1.4    Y    CITY         City
1.5    Y    STATE        ST
1.6    Y    ZIP          Zip
1.7    Y    PHONE        Phone
1.8    Y    SINCE        Customer Since   D2/
1.9    Y    YTD-PURCH    YTD Purchases    MR2
1.10
1.11
1.12
1.13
1.14
1.15
1.16
1.17
1.18

ENTER: F=FILE; D=DEL ITEM; S=SEQ; N=NEXT PG; K=DEL VAL; A=ADD VAL; *=CANCEL:
```

This screen has a "Y" next to all the fields that are selected. At this point you could choose to deselect a field or to add more fields. Looking at the fields that were selected, we can assume that all these fields are to be displayed on our screen. So let's move on.

At the bottom line enter F ↵

The system will ask that you press RETURN to continue. Press ↵.

Ok, now we're back to the **DATA ENTRY PROGRAM DEVELOPMENT** menu. After choosing the AUTO-BUILD function, the actual screen definition has been created for you. In the next section we'll explore the screen painter and look at the screen that we just created.

## 7.6 Using the Screen Painter

Now we will learn how to use the screen painter to modify a screen that was created using AUTO-BUILD. You should be in the **Data Entry Program Development** menu.

At the menu selection prompt enter 2 ↵.

A prompt will appear that says the following:

**USE SCREEN PAINTER (P) OR SYSTEM EDITOR (E) :**

If you chose option 'E' the system editor would be invoked. For our purposes now, let's choose the 'P' option.

At the prompt type P ↵. The following screen will appear:

```
UPD.CUSTOMERS       *** UPDATE CUSTOMERS MASTER FILE ***

  0. CUST NO           ;CUST-NO

  1. NAME              ;NAME
  2. ADDR              ;ADDR
  3. CITY              ;CITY
  4. ST                ;STATE
  5. ZIP               ;ZIP
  6. PHONE             ;PHONE
  7. CUST SINCE        ;SINCE
  8. YTD PURCH         ;YTD-PURCH









POSITION CURSOR          PAINTER 1
                                                  NÉ'└ Σ rrrrrrrrrrv $Ç @\ àü W(E
(à0åPèQ a í¶ó¶┬8C(E(àxë§bL┌▲ítnä_aº╨êÇ9β  ♣│  8pü t ;pβ♥ Φ w∝├•.╨ ∈└ç \á ▌Ç
```

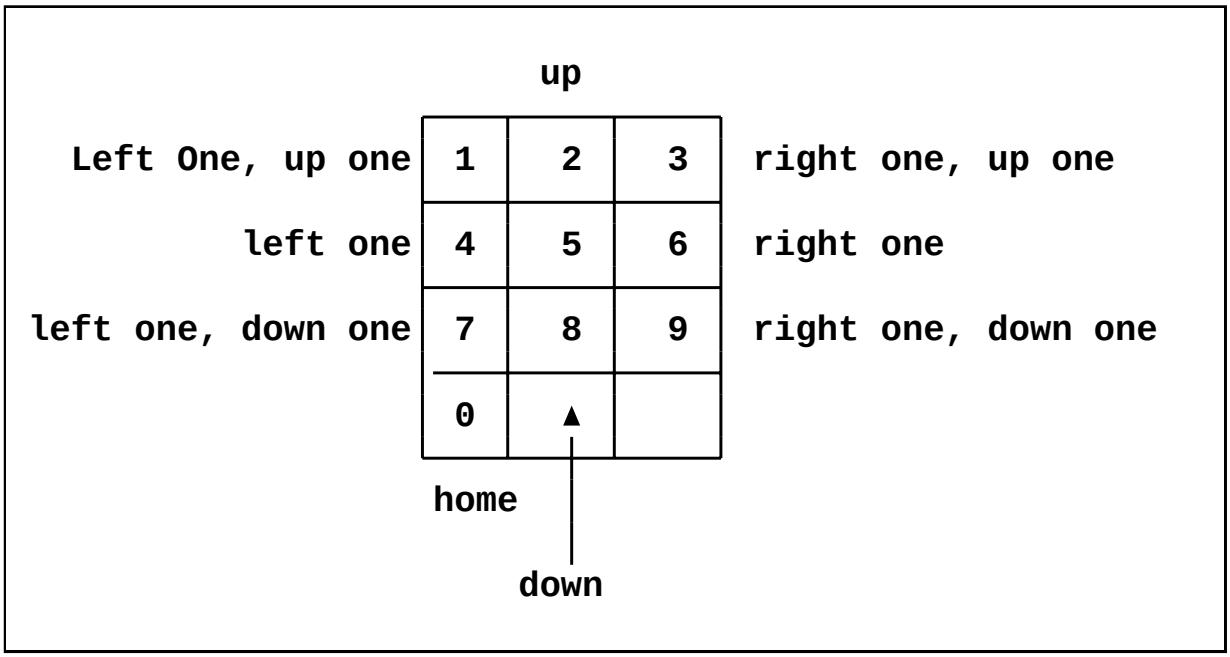
This is the way **The Programmer's Helper** generates screens automatically. If there had been more than 18 attributes, it would have displayed them in two columns.

Each attribute displayed on the screen is considered a data entry "step". The step numbers correspond to the order in which **The Programmer's Helper** will process the data. The semi-colons separate the screen literal from the dictionary item name. The semi-colon also acts as a delimiter that marks the screen position at which data entry input will occur.

The attribute name to the right of the semi-colon will also be used as the BASIC variable name for this field in the generated BASIC code.

Now let's modify the screen to enable us to enter two address lines and three telephone numbers. And we also want to make **SINCE** and **YTD-PURCH** display only fields. Ok, let's get started.

**Note:** To move the cursor in the screen painter, simply use the numeric keypad on the right side of your keyboard as illustrated in the sample picture:




- Use the numeric key pad to position the cursor after the "**R**" of "**;ADDR**". Remember, the number 2 moves the cursor down while the number 6 moves the cursor right.

- press T without hitting "RETURN" and you will notice the bottom line message changing to "**ENTER TEXT**". You are now in the "text entry" mode. In this mode, any text that you write will overwrite any existing text on the screen. Now type the following:

  ::S2 ↵

  Notice that after you hit "RETURN", the **POSITION CURSOR** command appeared again at the bottom line.

  The first colon you entered signifies the end of the attribute name and the start of the validation/conversion field. The second colon marks the end of the validation/conversion field and the beginning of the options field. The "**S2**" is the option in this case, and signifies

that the field is a simple multivalue with 2 allowable values.

- Now that we have specified that the address field is to have two lines, we have to make room for them on the screen. This is done by typing the "plus" key. Let's do that now:

  Press ↵ to put the cursor at the beginning of the next line.

  Now type +, this inserts a blank line under the address line.

  Now type another + to insert one more blank line.

  Every time you press the "plus" sign it inserts a blank line on the screen at the point where the cursor is positioned. **Note: The cursor should be positioned at the beginning of a line to avoid splitting a line in half.**

- Next, we want to allow for up to three phone numbers per customer. This is done in much the same way as what we did with the address field.

  Move the cursor to after the "**E**" in "**;PHONE**".

  Now type T to enter text entry mode.

  Type ::S3↵

  Again you are in the **POSITION CURSOR** mode.

  Press ↵ one more time to move the cursor to the beginning of the next line.

  We now need to add three lines to make room for the additional phone numbers. Type the following to do this:

  +++

- As we said earlier, we want the **SINCE** and the **YTD-PURCH** fields to be non-updateable, or read only. A **"N"** in the options field accomplishes this task.

  Place the cursor after the "**E**" in "**;SINCE**" and type [T] to enter text mode. Now type [::N ↵]. This will make the **SINCE** field a read only, non-updateable field.

  Now, on your own, do the same thing for the •YTD-PURCH field.

- The first stage of our program design is complete. We now have to file our screen away to proceed. Let's do that.

  Type [F], no "RETURN" is necessary to file the screen.

  The three options are "**F**" for filing and exiting the screen, "**S**" for saving and staying in the screen painter, and "**X**" for exiting the screen without saving it.

Your saved screen should now look like this:

```
UPD.CUSTOMERS       *** UPDATE CUSTOMERS MASTER FILE ***

  0. CUST NO           ;CUST-NO

  1. NAME              ;NAME
  2. ADDR              ;ADDR::S2

  3. CITY              ;CITY
  4. ST                ;STATE
  5. ZIP               ;ZIP
  6. PHONE             ;PHONE::S3



  7. CUST SINCE        ;SINCE::N
  8. YTD PURCH         ;YTD-PURCH::N
```

## 7.7 Adding to a Screen Definition

Now let's finish the screen by adding some more refinements. In this section we are going to add some multivalued attributes to the screen. This will require more room on the screen. This will be accomplished in two steps: the first will be to remove some of the spaces between the prompt text and the data entry point. Step two will be to relocate some of the screen fields so that they are laid out in two columns instead of one.

- The easiest and quickest way to remove some of the blank spaces on a screen is through the PICK EDITOR. From the Menu select option two and then select the "E" option instead of the painter option.

  Type [2 ↵]

  Select the editor by typing [E ↵]

  You are now looking at the screen definition item in the EDITOR.

  Type [T ↵] to place you at the top of the screen.

  Type [R99/ ;/; ↵] to remove 5 spaces before the semi-colon.

  That's all it took. Let's file it by typing [FI ↵]

  Now we are back at the **Data Entry Program Development** menu.

- Our next step is to once again use the screen painter. This time, we're going to move some fields around. We will put the **STATE** and the **ZIP** fields on the same line, and we'll move the **SINCE** and **YTD-PURCH** fields to the right column.

  Again, select option two on the menu and then select the painter option.

  [2 ↵ P ↵]

  The **UPD.CUSTOMER** screen is now displayed. Note that the cursor is always in the HOME or (0,0) position when we first start the painter.

  Press [↵] until the cursor is right before the **5.ZIP** step.

  We now need to insert some spaces, so press [I].

  Now press the space bar [Space Bar] once. Did you notice that the entire line was shifted to the right? Now keep pressing the space bar until the "**5.ZIP**" is to the right of

"**;STATE**" on the line above it. Press ↵ and you will again in **POSITION CURSOR** mode.

- We now need to get the **ZIP** field on the same line as the **STATE** field. This is a simple task.

  Type 8 to move the cursor up one line.

  Now type -. The "minus" key will join the two lines.

  **NOTE: the minus key will, if placed at the beginning of a line, act as a delete key and delete the whole line. So care must be taken as to where the cursor is positioned before using this command.**

- If you look closely at your screen, you will notice that there are far too many spaces separating the ZIP literal with the input prompt. So let's remove some of those unwanted spaces.

  Press 6 until the cursor is behind the "**P**" in "**5.ZIP**" and then press D. One space was deleted. Press D again. Another space was deleted. Keep pressing the "D" until there are only two spaces separating the "**5.ZIP**" from the "**;ZIP**".

- Now press ↵. The cursor should be in front of the "**7.CUST SINCE**" field.

  Type I to go into Insert mode.

  Press Space Bar until the **CUST-SINCE** field is on the other side of the screen.

  Press ↵ twice. Once to exit Insert mode and again to place the cursor at the beginning of the next line.

  Repeat the above steps with the **YTD-PURCH** field until it lines up with the **CUST-SINCE** field.

  We now need to move the two fields up on the screen to make room for the multivalues that we will be adding.

Move the cursor just to the right of "**;PHONE::S3**".

Press [-], the minus sign, until the **CUST-SINCE** field is on the same line as the **PHONE** field.

- File the screen by typing [F] and the screen should look like:

```
UPD.CUSTOMERS       *** UPDATE CUSTOMERS MASTER FILE ***

  0. CUST NO     ;CUST-NO

  1. NAME        ;NAME
  2. ADDR        ;ADDR::S2

  3. CITY        ;CITY
  4. ST          ;STATE   5. ZIP  ;ZIP
  6. PHONE       ;PHONE::S3              7. CUST SINCE  ;SINCE::N
                                         8. YTD PURCH   ;YTD-PURCH::N
```

OK, now we are ready to add several attributes to our file. From the menu prompt enter [7]. This will allow us to update the file dictionary. Let's create the following attributes:

| Item | AMC | HEADING | CONV | JUST | MAX |
|---|---|---|---|---|---|
| **PROD-CODE** | 9 | PRODUCT CODE | | L | 10 |
| **EST-PURCH** | 10 | EST PURCH | MR2 | R | 8 |

The attributes that you just created are a multivalued set represent a forecast of how much of each product the customer will buy this year. **PROD-CODE** is the controlling attribute, and **EST-PURCH** is the dependent attribute.

Let's add these two fields as a multivalued set in our **UPD.CUSTOMER** program.

- From the menu prompt reenter the screen painter.

  Type 2 ↵ to enter the painter.

  Select the painter mode by typing P ↵.

  This time, we want to know what column and row the cursor is currently positioned at. This is easily accomplished in the painter by typing the letter "V".

  Type V to see the column,row position on the bottom right side of the screen.

- Press ↵ until the cursor is on row 14.

  Type T to enter text mode.

  Type the following line:

  **Ln Product Code Est Purchases**

  Now press ↵ twice (once to exit text mode and one to move to the beginning of the next line).

  Now type T and the following line so that it lines up with the header line:

**9. ;PROD-CODE::V 9.;EST-PURCH::V**

Press ↵ to exit text entry mode.

As you typed in the information for the multivalued window you probably noticed that the step number 9 appeared twice. This is because all of the multivalues in a multivalued window will all possess the same step number. The **V** option denotes this field as being a multivalued field and is only required on the controlling field. An optional "window size" parameter (e.g. **V5** ) can also be specified, which means that the multivalued window will be 5 deep.

If the window size is omitted, it is assumed that the multivalues will take up the rest of the screen.

Your screen should now look like this:

```
UPD.CUSTOMERS       *** UPDATE CUSTOMERS MASTER FILE ***

  0. CUST NO     ;CUST-NO

  1. NAME        ;NAME
  2. ADDR        ;ADDR::S2

  3. CITY        ;CITY
  4. ST          ;STATE   5. ZIP  ;ZIP
  6. PHONE       ;PHONE::S3              7. CUST SINCE  ;SINCE::N
                                         8. YTD PURCH   ;YTD-PURCH::N

  Ln    Product Code              Est Purchases
  9.    ;PROD-CODE::V           9.;EST-PURCH::V
```

- File the screen by typing F.
- Now let's compile the screen. From the main menu select option three.

  Type 3↵

  This will pre-compile or process the screen definition and check for spelling or syntax errors. Next, you will generate your very first program!

  Select Option 5 by typing 5↵. This option will generate the actual source code for the program.

  When the code is finished, you are asked if you want to compile the code. Answer Y↵.

**The Programmer's Helper** will compile the main program and then create a secondary program with the main program's name and append the word "**.CALLER**". (e.g **UPD.CUSTOMER** and **UPD.CUSTOMER.CALLER** are the two programs that would be generated).

## 7.8 Multiple File References

In this section, we will expand on what we learned in the previous one by using the same screen definition for **UPD.CUSTOMER**. We will be adding a file validation to the screen and an additional data entry window.

In order to complete this tutorial, we will use a file called **MODELS**. This file will be used to validate the product codes entered in the **UPD.CUSTOMER** screen. The file has already been created on the TPH4 account and has had dictionary items created.

Type [TPH ↵] to enter **The Programmer's Helper** main menu, if not already there.

Select option 8, **Data Entry Program Development**, and press [↵].

When asked for the program name, type [UPD.CUSTOMER ↵].

Get into the screen painter by typing [2 ↵] and selecting the painter option [P ↵]

Move the cursor until it is positioned under the second of the two colons on the **PROD-CODE** step.

Press [I] to insert text.

Type [TMODELS;X;1;1]

You'll notice that the text to the right of the cursor is moved to make room for the text you are typing in.

Now press [↵] to exit insert mode.

With your knowledge of the cursor control keys, move the cursor up one line and align the heading line with the new multivalue window line under it.

The translate function that we added will serve two purposes. First, when a product code is entered, it will be checked against the MODELS file to see if it exists. If it does not, then an error message will be displayed and you will be asked to reenter your data. Second, when a valid product code is entered, attribute 1, in our example, from the MODELS file will be displayed to the right of the product code. Any attribute can be pulled from a translated file.

Let's now add one more feature that is very valuable - the ability to create a new item in a different file on the fly. In other words, when the "NOT ON FILE" message is displayed, it would be ideal to be able to create that "not on file" item without having to exit the program that we are currently in. With **THE PROGRAMMER'S HELPER** this can be done quite easily.

- You should still be in the screen painter updating item **UPD.CUSTOMER**.

  Position the cursor so that it is directly behind the "**:V**" in the **PROD-CODE** step.

  Type (T) to enter text and type (:UPD.MODEL)

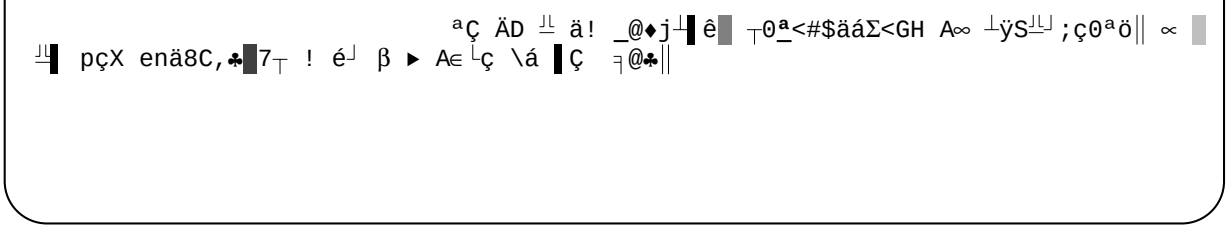
  Now press (↵)

  **UPD.MODEL** is the name of a program generated by **The Programmer's Helper** to update parts in the MODELS file. Now, with this addition, when a product code is not on file you will receive a message that asks "**NOT ON FILE - CREATE IT?**"

  If you type "Y" at the prompt you will be transferred to the **UPD.MODELS** program. If the new program takes less than a full screen's space, it will be overlaid over the current screen.

Let's take a look at what our final screen looks like:

```
UPD.CUSTOMERS       *** UPDATE CUSTOMERS MASTER FILE ***

  0. CUST NO      ;CUST-NO

  1. NAME         ;NAME
  2. ADDR         ;ADDR::S2

  3. CITY         ;CITY
  4. ST           ;STATE   5. ZIP  ;ZIP
  6. PHONE        ;PHONE::S3              7. CUST SINCE  ;SINCE::N
                                          8. YTD PURCH   ;YTD-PURCH::N

  Ln    Product Code                          Est Purchases
  9.    ;PROD-CODE:TMODELS;X;1;1:V:UPD.MODELS   9.;EST-PURCH::V
```



The last step is now to file the item, pre-compile, generate the code and then compile it. Let's continue.

- Type F to file the screen.

  Now select option 3 by typing 3↵

  This step will pre-compile the screen and check for errors.

  Select option 6 by typing 6↵

  This will generate the data entry BASIC code for the **UPD.CUSTOMER** file. When the system asks if you want to compile the program, simply enter a "Y" and the program will automatically be compiled and cataloged, ready for use.

**CONGRATULATIONS**!!! You've just designed, modified and generated your very first program. That was pretty easy, wasn't it? And there's really no limit as to what **The Programmer's Helper** can do for you.

As with all products, practice makes perfect. Keep trying out **THE PROGRAMMER'S HELPER** features and get comfortable with the program. If you should have further questions, comments, or if a point is unclear, please do not hesitate in contacting your dealer for support.

Happy coding!

## Appendix A

# Installing
# The Programmer's Helper

**The Programmer's Helper** is a simple program to install. The following steps will make the installation process as painless and simple as possible. Let's begin:

**FOR FLOPPY DISK USERS**

**1** Turn your machine on

**2** After the system has "come Up", type the following at the **LOGON** prompt:

SYSPROG ↵

If **SYSPROG** is password protected you will be asked to enter a password at this point. Do so. If the system is not password protected, then you will be taken to the system prompt (TCL).

**3** Insert **The Programmer's Helper** diskette into drive A.

**4** For 360Kb drives -

SET-FLOPPY (AS ↵ or,

For 1.2Mb high density diskettes,

SET-FLOPPY (AH

**5** Type T-REW ↵

**6** To load the diskette contents onto your system, now type the following:

ACCOUNT-RESTORE TPH4 ↵

ACCOUNT NAME ON TAPE: TPH4 ↵

The system will begin loading the **TPH-III** account.

**8** Next you must logto the TPH account an run an installation procedure.

LOGTO TPH4 ↵

At the TPH Master Menu, press the return key and you will be sent to TCL. Enter the following:

TPH.INSTALL ↵

This will compile all of the TPH programs and copy some messages to the ERRMSG file.

**9** Once **The Programmer's Helper** has been loaded, put the original diskettes away in a safe place.

The next section will explain how to load **The Programmer's Helper** into your own user accounts. The process is simple:

**1** •Logto your user account. If you are in SYSPROG type the following:

LOGTO <account-name> ↵

?

If you are at the logon prompt type this instead:

<account-name> ↵

**2** Once in your account, make sure that you are at TCL and type the following:

SET-FILE TPH4 TPH-BP ↵

If the system replies "**QFILE Updated**" then proceed to the next step, else, make sure that your TPH installation went OK, or that there were no typos in the SET-FILE command.

**3** Now type RUN QFILE SETUP ↵

**4** Repeat steps 1 through 3 for for any other accounts that you would like to use **THE PROGRAMMER'S HELPER**.

## INDEX

# Table of Contents

# Table of Figures

1) Level: Format Error
   Message: Printer cannot print character (printing fallback character)
   Location: Document Body
      Page: 137  Distance from TOF: 7.308in
      Level: 1  Section: 8  Block: Text #13  Column: 2

# Document Print Summary

## Author: Rick Zanotti/Fred Waltman

## Description: The Programmer's Helper Documentation

Page
- Title ..... 0
- Table of Contents ..... 4
- Document Body ..... 138
- Index ..... 6
- Error Log ..... 1
- Document Print Summary ..... 1
- Total ..... 150

Error
- Warnings ..... 0
- Format Errors ..... 1
- Global Format Errors ..... 0
- Total Errors ..... 1

Copies ..... 1
Print Quality ..... Final
Automatic Hyphenation ..... Yes
Force Document Expansion ..... Yes

Document Revision ..... 1.236
Total Words in Document Body ..... 24454