DATAFORM
DATAFORM
DATAFORM
DATAFORM
DATAFORM
DATAFORM

SIMPLIFIED USER'S GUIDE
For Cassette, Diskette, and Disk Systems

DATAPOINT CORPORATION

A SIMPLIFIED USER'S GUIDE

TO

THE THREE DATAFORMS:

DISKETTE DATAFORM

CASSETTE DATAFORM

DISK-BASED CASSETTE DATAFORM

A language for form generation, data checking, computation and data capture employing the Datapoint line of Dispersed Data Processing Equipment.

The text in this manual was entered, edited and typeset using a
Datapoint 2200 Business Processor with a phototypesetter and the Scribe
Text Processing Program.

Table of Contents

CHAPTER ONE

OVERVIEW

Introduction

Placing computer power and communications where business transactions occur (such as a distant sales office) rather than sending the documentation of a transaction via surface or air transporation to a centrally located large computer, has an appeal because of its convenience and its cost-saving features. The concept of Remote Intelligent Data Entry has allowed businesses to cut costs, save time and reduce errors by having a computer at the job location which checks data as it is keyed in.

Anyone wishing to utilize intelligent data entry devices, though, has had to overcome the programming obstacle. Also, an intelligent data entry system requires changes and improvements, during business growth, which means that a manager is committed to numerous programming changes.

DATAFORM provides a solution to the intelligent data entry programming problem. Level I DATAFORM allows fast and easy screen format generation and data capture with fundamental editing capabilities. If more complex operations are required, short subprograms may be written in Level II DATAFORM -- a unique high-level language. Data is stored on diskette, cassette, or disk and is readily available for subsequent transmission to another Datapoint Processor or other computer for processing. For more information on communications see the DATAPOLL communications language manual.

DATAFORM is an intriguing language because it offers enough power and versatility to impress an experienced programmer, and yet offers simplicity of use for the non-programmer. The combination of the Datapoint Processor (1100, 2200, 5500) and the DATAFORM Simplified User's Guide make it easy for an individual who is totally non-computer oriented to jump the programming hurdle. The aim of this book is to do just that -- ease you over the hurdle.

How to Use This Book

This text offers an introduction to DATAFORM, an explanation of its capabilities and sufficient background to write applications programs for all but the most complex data entry operations. If more technical coverage is required, an expanded DATAFORM Program User's Guide offers a complete reference for each of the three DATAFORMS.

This Simplified User's Guide provides information on DISKETTE DATAFORM, CASSETTE DATAFORM, and DISK DATAFORM. If you have very little data processing background, it is recommended that you read all of the sections which pertain to your equipment. However, if you have sufficient data processing background to jump right into the Diskette form and program, go directly to the Step-by-Step Parts of that chapter. If you wish to gain as much knowledge of form generation as you can, but already have programming background, you can work your way through Level I and then jump to the Step-by-Step Level II Part.

DATAFORM is divided into two levels, Levels I and II. All DATAFORMS are highly similar so only a detailed treatment of the Diskette version is given with the features of the others given in separate chapters.

Level I

Reading the Form Generator Section will indicate just how fast screen forms with many editing and error-checking capabilities can be generated. Data can be stored on diskette, cassette, or disk and later transmitted with DATAPOLL or other communications packages. No further work is required to create an operational data entry system.

The text is roughly divided into two major sections. The first describes the vocabulary of forms, form generation concepts, and use of field editing criteria. The second section deals with the steps and operations actually used to create a working data entry system. Once the first section has been read and understood, the second section can be used for practice or as a reference. An Abbreviated Step-by-Step Guide is also included.

Level II

If more complex error-checking or computation on the entered data is needed, the user should read this section, as it deals with the DATAFORM Programming Language.

This straight-forward language allows short programs to operate on the entered data and can perform arithmetic and other sophisticated tasks.

Since Level II DATAFORM is actually programming, a short introduction to programming is offered. The actual steps involved are so short that users lacking programming experience should not become discouraged. DATAFORM'S structure approaches English language instructions and should not be viewed as territory only for professional programmers. If you're still skeptical, go through the book and ask a programmer friend to help you over any troublesome spots.

Some Encouragement

Once an individual has actually gone through form generation and other steps, his general reaction is "That's not hard". It is more a task of explanation than implementation. So take this feeling on faith -- once you've read through the text and run a form, we're sure you'll agree -- that it's easy!

One more thing. If you're planning on building a system using the Level II language features, remember that you'll be working with several different concepts: designing the program system, creating forms, writing the programs, cataloguing, and integrating a complete working data entry system.

Each step of this process involves learning new techniques and following a step-by-step process -- both of which will give you plenty of chances to make mistakes. Don't be discouraged. Any computer

system which works for you as much as DATAFORM does will require some patience and understanding until you know the steps and sequences involved. Once mastered, you'll be competent in a powerful and useful business system.

So read on and enjoy. If you run into questions or problems, your Datapoint Systems Engineer or Account Manager can provide help.

Planning for DATAFORM

In preparing to program the Datapoint Processor, you should examine your business forms to take full advantage of the large and easy-to-read screen. All of the 12 lines and 80 columns may be used for data entry. Often, judicious placement of fields can save the use of two or more forms. The use of more than one form is no problem. It is frequently done purposely to simplify operations.

DATAFORM planning work sheets are available to arrange a suitable screen image. The work sheets also include space to note the various edit and error-checking criteria used.

Outside of the economic considerations, the choice of equipment is mainly one of storage area and speed of operation. The Diskette 1100 and Disk Systems are considerably faster than the Cassette System and have a great deal more storage area for forms, programs, data, etc. The Diskette 1100 System has roughly 1/4 million character storage area per drive; the maximum of four drives would represent storage area of roughly one million characters. The Cartridge Disk System offers roughly a 2 1/2 million character storage area on each cartridge; the maximum of four drives represents about a 10 million character storage area. For comparison, a cassette will store approximately 120,000 characters per side.

CHAPTER TWO

LEVEL I DISKETTE DATAFORM

(NOTE: All DATAFORMS are very similar. The general features of DATAFORM are covered here with emphasis on a diskette-based system. If you're planning a system using cassettes or cartridge disk, read this section first then read the section that covers the specific language.)

Using the Level I DATAFORM - A Sequence Overview

Before we begin the nuts and bolts discussion of what it takes to get an intelligent data entry system up and running let's briefly talk about where we're headed.

The complete Level I start-to-finish sequence involves three parts:

Form Definition
You first decide on what the form will look like on the screen. If you're working from standard company formats, great. If not, just make up a good format, such as a name and address form. A large graph-paper screen sheet format is available for this purpose, although if you're just puttering around, loose placement will suffice. The large screen gives lots of room. For teaching purposes, we will be using one simple format throughout the book.

Form Generation
During this step you will actually create the form on the screen and assign various editing and error-checking criteria. This is the most technical part but actually takes the least time.

Data Entry
This is the result of your work. The operator can load your form, enter data in it, have the data checked for errors and write the data back to diskette. This will be a complete, working intelligent data entry system. When data entry is complete, the diskette will contain a record of all the data, ready to be transmitted to another location or, in some cases, processed locally right in your own office using other programs for the Diskette 1100.

That's all there is to it. It looks easy and it is.

PART I
Determining the Screen Format

Let's first examine the elements of a simple form. Naturally, to make the tasks easier, we'll pick a fairly simple format that lends itself to clarity. The data entry operation involves a vocabulary of its own and we want to cover that first. Probably, the worst part about learning a new programming or system technique is figuring out the terms.

SAMPLE BUSINESS FORM & ITS PARTS
Drawing No. 1

The total information on this form is called a "record," i.e., the information could be referred to as Ruby Tuesday's record. The total group of customer records stored is a "file".

We can suppose, for the moment, that the owners of the Green Earth Garden Shoppe fill out a card on each customer and use the card information for advertising mailings or credit information on those accounts desiring mail orders or a monthly bill.

In any case, our job will be to build a DATAFORM form allowing an operator to load this card data onto a diskette for later transmission to Green Earth's home office in Los Angeles. For the sake of clarity, we'll just place this form loosely on the screen. Naturally, if you were working with a genuine application, far more thought would go into form text placement.

PART II
Features of the Form Generator

Before covering the actual operation of the features, a word of explanation as to just how these features are entered into the machine. Form Generation proceeds in a series of steps as follows.

The system user first creates a screen image by actually typing form text and fields on the screen. After the form is complete, the user begins a series of steps or 'passes'. In each of these passes, the form reappears on the screen and the various criteria are typed in and automatically placed in memory. After all the passes are done, the completed form is written to the diskette for later use. It can also be recalled for revision, if necessary.

Although this sounds intricate, it isn't. The actual process of completing all the passes takes approximately 15 minutes.

Rather than burden our discussion now with the mechanics of the procedures, we'll cover the basic principles first, then give a step-by-step procedure for the passes. That way, you'll have a better understanding of what you're doing when you actually generate a form.

Step 1. Defining Form Text and Field Lengths

The space in which the operator is to type in the information is defined by vertical bars and underlines. Each field must be started with a vertical bar, which counts as a character space and begins the field. Underlines (__) or carets (Λ) are used to define the rest of the field.

In the example below, a colon is used to aid the operator. It is purely esthetic and could have been omitted but, as will be shown later, provides a visual break between form text and the field.

the form text   ┌field   A 10 character field

       ➤NAME: I_____ (9 underlines + 1 vertical bar)

Note that the vertical bar is the first character position of the field. It tells the computer where to begin data entry.

This small form will easily fit on the screen of the Datapoint so we can arrange it in the following manner, as shown on the next page:

DATAPOINT SCREEN

DEFINING FIELDS WITH UNDERLINES & VERTICAL BARS
Drawing No. 2

Note the Area Code and State fields. As an example, we're going to assume that the majority of customers at this location are from Minnesota and have the same Area Code. These fields will be automatically written in to save work but can be changed if an out-of-state customer card is encountered. More on this later.

Up to 126 fields or a total of 249 data character slots may be created for each form, which should be more than sufficient!

Step 2. Defining Field TYPEs

The second step in generating a form is the TYPE definition. This allows the form designer to specify what category of data he will accept in that field. If the operator attempts otherwise, the Datapoint will emit an audio "Beep" and reject the characters. This is a first in a series of 'passes' to provide field editing.

NOTE: During the ensuing editing steps, the forms reappear on the screen and permit the letters to be typed in. We'll show how this is done in a moment.

Now let's define what TYPE restrictions are available:

Code	Meaning
A (Alpha)	Accepts upper case alphabet characters, A thru Z or a space character (blank). The field is left justified (typed from the left border of the field) and unused character spaces are filled with blanks.

D (Digit)	Specifies numbers only for a field. Only numbers 0 thru 9 are acceptable; left justified with blanks filling unused character spaces.
N (Numeric)	Same as Digit but a minus sign (-) and a decimal point are allowed; no plus signs allowed, however. When the field is entered, it is checked to contain one or less decimal points and the minus sign must be in the leftmost position. Also, a maximum of 16 numbers to the left and 8 to the right of the decimal point are allowed.
M (Mixed)	Both Alpha and Numeric types (0-9, A-Z, minus sign and decimal point) can be filled in. The field is blank filled to the right if not completely used.
O (Minus-Overpunch)	Numeric, Minus-overpunched field must conform to the numeric format with the exception of the last digit. Typing a MINUS key instead of the enter key in this type of field will cause the right most digit to be "overpunched" with a minus sign, i.e., a '1' becomes a 'J'. In the REQUIRED pass, do not assign fill control F or B or Keyin-continuous K or X. This feature is sometimes used in applications where a keypunch machine is to be replaced.
No type specified (Blank)	If no letter is used, no restrictions or operations are performed on the field.

Users who have worked with a keypunch will find these terms familiar and common to data preparation and error-checking.

Let's look at our form and see how the TYPE edit codes enhance the data entry operation.

```
  I                                                          80
┌──────────────────────────────────────────────────────────────┐
│                                                                │
1     NAME: A ◄──────────        these fields should have alphabet
2     STREET: M ◄──────              characters only
3     CITY: A ◄────
4     STATE: A ◄───               the street will have mixed numbers
5     ZIP CODE: D ◄──                and alphas
6     TELEPHONE: (D) D - D ◄──     these fields should have numbers only
7     CUSTOMER NO.: D ◄──
8     PURCHASE AMOUNT: $ N ◄──     the dollar value can contain a
9                                    decimal point or be minus in
10                                   case of a refund
11
12
│                                                                │
└──────────────────────────────────────────────────────────────┘
```
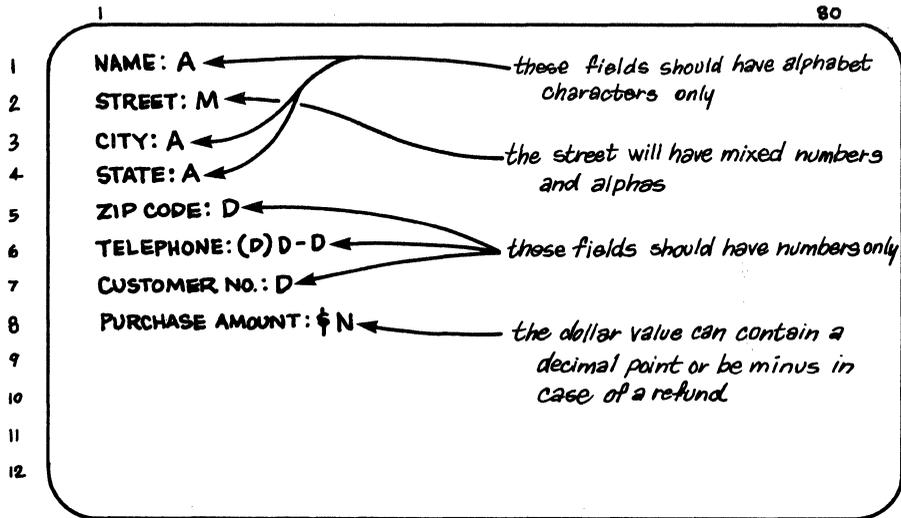
DEFINING 'TYPE' FIELD RESTRICTIONS
Drawing No. 3

Step 3. Assigning Justification and Fill

Now that the field types are set, let's justify our fields so that characters butt up against the left or right edge of the field and fill control them (unused character space filled with zeros or blanks). Let's look at the available options for this second 'pass':

J (Right Justify)   Causes the field to be right justified and the unused character positions on the left, blank filled upon completion of keyin.

Z (Trailing Zeros)   Causes trailing zeros to be added upon completion of keyin.

R (Left Zeros)   Causes the field to be right justified and left zero filled upon completion of keyin.

Now let's apply them to our form. As a convention, Alphabetic and Alphanumeric fields are usually displayed left justified and right blank filled; Numeric and Digit fields are usually displayed right justified and left blank filled. Since our Alpha and Mixed fields are already left justified and right blank filled, we won't change them. However, let's type in 'J' in the Numeric field (Purchase Amounts) to change it to right justified and left blank filled.

DATAPOINT SCREEN

```
                                                          80
NAME: I _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _
                                            all numbers in this field will be
STREET: I_ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _    right justified now

CITY: I_ _ _ _ _ _ _ _ _ _ _ _

STATE: I_

ZIP CODE: L_ _ _

TELEPHONE: ( I_ _) I_ _ - I_ _ _

CUSTOMER NO.: I_ _ _ _ _ _
PURCHASE AMOUNT: $ J_ _ _ _ _ _ _
```

<div align="center">

DEFINING JUSTIFICATION & FILL
Drawing No. 4

</div>

Step 4. Defining Which Fields are Required

In most data entry operations there are certain fields which the operator may elect not to fill in and conversely there are usually several fields which must be filled or the record will be incomplete.

For example, our Green Earth format would certainly be meaningless if the operator neglected to key in the name of the customer. DATAFORM allows the system planner (that's you) to require field entry when needed. A variation of this requirement is to insist that all the spaces in a field must be filled.

Let's examine what options are available in the required pass and then use these in the Green Earth form.

R (Required)

A required field demands the operator fill in at least one character. The field may not be skipped and left empty.

F (Fill Controlled)

This restriction requires that all positions must be filled in. After a fill controlled field is complete, the operator is automatically tabbed to the next field. An option exists in that the field may be entirely skipped if the ENTER key is pressed before any data is entered.

B (Required
& Fill Controlled)

This restriction combines both Required and Fill. The field may not be skipped and must be completely filled in.

P (Program Reserved)	When a DATAFORM program has been created that will perform some operation or computation and perhaps display the result, 'P' prevents the operator from keying into this field. Chapter 3 on the Level II language covers this in detail.
S (Program Reserved & Required)	If a field will be filled by a program, this restriction requires that the program must be executed before that record may be written to diskette. Again, more on this in Chapter 3.
K (Keyin-Continuous)	A field will be automatically entered when the last position in the field is keyed in; otherwise, the Enter key must be pressed to move to the next field.
X (Required & Keyin-continuous)	Same as Keyin-continuous, but at least one character must be entered. Once the last position of the field is keyed in, the next field will be automatically entered; otherwise the ENTER-key must be pressed.
Blank (No Restriction)	If none of the above are used then the operator may bypass the field.

These restrictions, as we said before, are implemented during a form generation operation or 'pass', which will be explained in detail later.

Now let's see how we can put these features to use on our example form:

80

```
 1     NAME: R _____  ← the operator may not bypass
 2     STREET: R _____        these fields
 3     CITY: R _____  ←
 4     STATE: R_              ← these fields must be filled in
 5     ZIP CODE: F___  ←        completely, i.e., all five digits
 6     TELEPHONE: (F__)F__-F___  ←  or bypassed at the beginning
 7     CUSTOMER NO.: _____  ← this field is completely
 8     PURCHASE AMOUNT: $_____   optional - no restrictions
 9
10
11
12
```
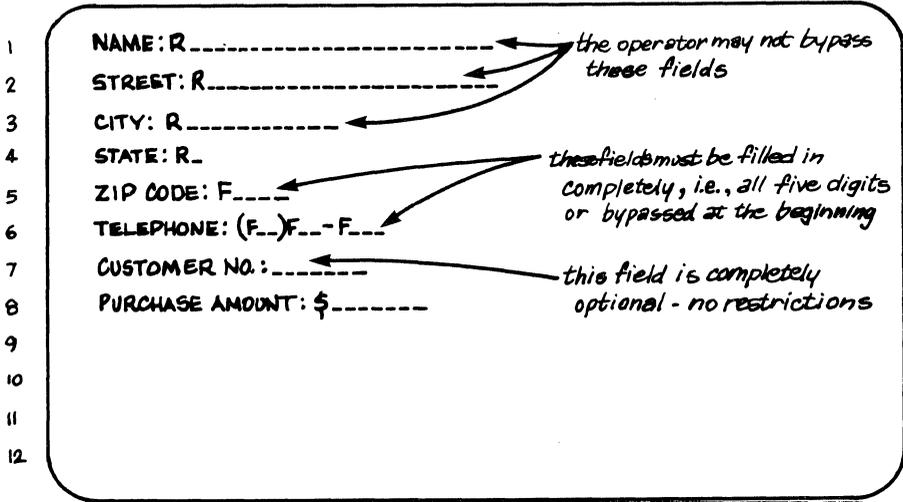
DEFINING REQUIRED OR OPTION FIELDS
Drawing No. 5

Step 5. Assigning Semi-Constant / Constant Fields

In our application, we noted that most of the customers would be from Minnesota and most likely have the Area Code 612. By defining fields as Semi-Constant, the form will be displayed with the anticipated data already entered. The operator has two choices: the field may be accepted as is (i.e., the Semi-Constant characters) and skipped over, or the field may be changed by simply typing in correct data over the displayed data (the Semi-Constant characters).

In cases where a field must not be altered, such as an identifying record of the originating office, a Constant field may be defined. The operator cannot change the contents of this field and it will be written to diskette in that manner each time a form transaction is completed.

We can make use of the Semi-Constant field in our application in the State Field and the Area Code portion of the Telephone Number Field.

DATAPOINT SCREEN

80

```
 1     NAME: I _____
 2     STREET: I _____   ← the state and zip code are
 3     CITY: I _____            semi-constant fields, and
 4     STATE: MN  ←                      may be typed over if needed
 5     ZIP CODE: I____
 6     TELEPHONE: (612) I__-I___  ←
 7     CUSTOMER NO.: I_____
 8     PURCHASE AMOUNT: $_____
 9
10
11
12
```
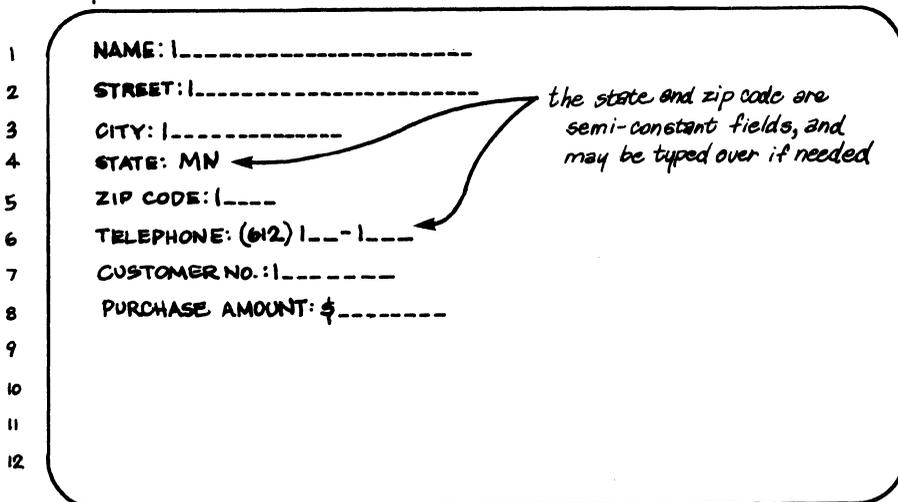
DEFINING SEMI-CONSTANT & CONSTANT DATA
Drawing No. 6

17

For some applications, it might be useful to create a form with all Constant data for use as an identifier at the beginning of each day's operation, or as a set of instructions to the operator. This can be done by making the whole form out of 'Constant' data. Note that at least one field must be left for the operator to fill or the form will hang, locked in place, as there'll be no way to tell it that we're done with it! Fields which are to be used for Constants or Semi-Constants may not be defined with the caret (Λ). Finally, note that neither vertical bars (|) nor underscores (__) may be used as characters inside a Constant or Semi-constant field. For Constant data, a caret (Λ) may be used to enter a blank space.

Step 6. Assigning DATAFORM Programs to Fields

As mentioned earlier, Level II of DATAFORM allows programs to be written in a high-level language that performs a wide variety of tasks. These programs are created in a separate operation which will be covered later. However, the Form Generator Program provides the means to tie these programs to the appropriate form fields, so at least an explanation is due here. If you plan on using a program, now's the time to specify it.

The operational DATAFORM programs are usually short and designed to work on one particular field or any number of fields on the screen. This step or pass of the form generator permits the system designer to type, into a field, a letter, A through Z, which corresponds to a program. Only 26 programs may be assigned to one screen form because of the limits of the English alphabet.

We can assume, for example's sake, that we've written a short program that checks for the correct check digit in the account number, as in the following illustration:
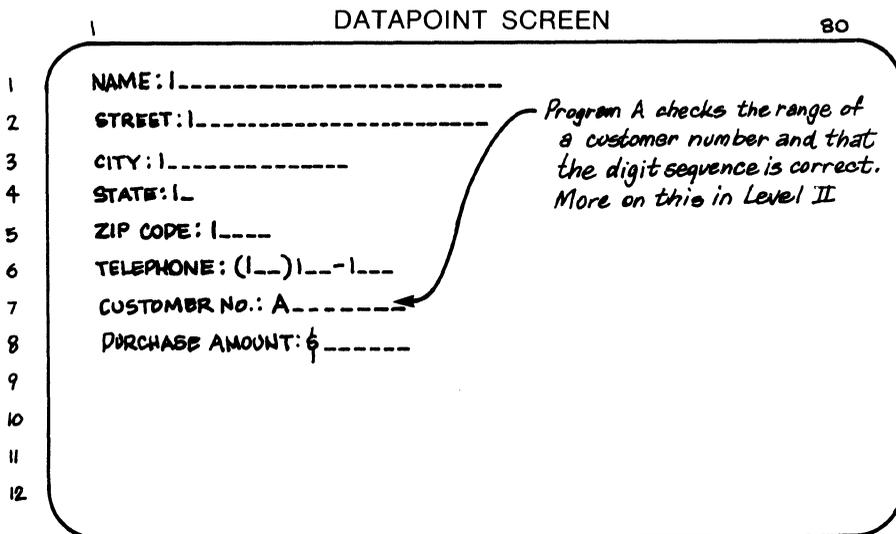
DATAFORM LANGUAGE PROGRAMS MAY BE ASSIGNED
Drawing No. 7

18

Remember that the use of these programs is optional. If you feel the Form Generator features will meet system requirements, this step or 'pass' may be skipped. If you call for a program in this step and then elect not to write it, don't worry. If the Form Generator can't find the referenced program, it continues normal generation and notifies you with a 'Program Missing' message on the screen.

All this will be covered in Chapter 3 under Level II DISKETTE DATAFORM.

Step 7. Displaying Other Forms

In many business applications, a single screen form won't be adequate. Numerous business transactions require much field information to describe the heading, then lists for item entries.

An example of this might be an order for parts. The first screen's format consists of shipping, billing and destination information. After completing that, the next form will contain a list for the ordered parts, with the option of calling up as many screen forms as needed to fill the parts order.
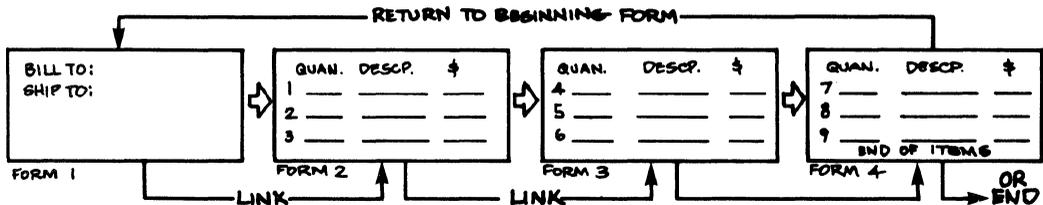
THE FORM GENERATOR PERMITS LINKING TO OTHER FORMS
AUTOMATICALLY OR THROUGH MANUAL OPERATOR CONTROL
Drawing No. 8

The Linking Technique

During the generation of forms, numbers are assigned to each screen form. When, during data entry, the operator has decided that this form is complete, (and DATAFORM agrees!) control keys are pressed (see Part III) which allows the present data to be written to the diskette. After data is written, normally the form in use is redisplayed with the data fields cleared ready for the next record. To have one form link to another, a LINK Number is defined for each form. This number tells which form should be fetched and displayed next. The LINK Number is the Form Number, and like the Form Number, 01 through 99 can be assigned during Form Generation.

It would be helpful to the people receiving the data if you made a Constant field and placed the form number in that field. Therefore, if a form is skipped, they will know what original format was used.

For example's sake, we'll assign arbitrary form numbers ourselves, so that we'll know which form to link with. DATAFORM looks at the link number, which you've assigned to the form, and then finds it. You can even have the last form reference the first, creating a complete loop-around.

Two kinds of linking are available: Manual and Automatic.

Manual Linking

If you'll look ahead to Part IV, you'll see that upon pressing and holding down the DISPLAY key, the numeric pad can perform a wide variety of control functions for data entry, very much as we did for form generation.

As surmised, WRITE DATA control function takes the data from the form and writes it on the diskette. The original form you were using comes back up on the screen, (fields blank) ready to be filled in again. If the LOAD NEXT FORM function key is pressed, the program looks to see what link number has been specified, finds that form and displays it on the screen. So, using this feature, the operator can elect to keep entering data on the same form or call up the next form by pressing the LOAD NEXT FORM key; by hitting the LOAD NEXT FORM function key twice the third form can be loaded; by doing it three times, it loops back to the first form.

During Form Generation time the LINK must be set to the next form to be used by typing "LINK↓". The message "NEXT FORM = 000:" will appear. This means that you haven't set any links yet. If you had, it would read "NEXT FORM = 002:", for example. To set a link to Form 03, type in '003'.

For example, suppose the data entry job consists of three separate forms and we wish to use manual linking as follows:

MANUAL LINKING
Drawing No. 9
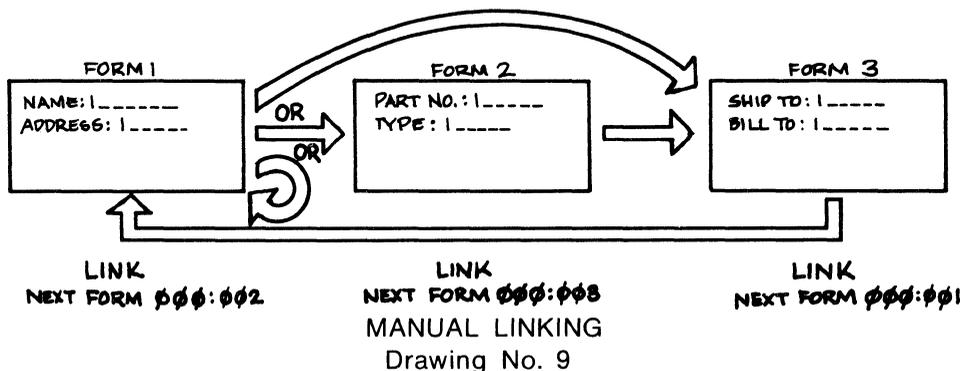
Note that after filling out the first form, the operator has a choice. If the WRITE DATA function keys are pressed, the data will be transferred to diskette and a blank FORM 1 redisplayed. To use FORM 2, the LOAD NEXT FORM function must also be pressed. If LOAD NEXT FORM is activated twice, FORM 3 will come up. After FORM 3, LOAD NEXT FORM will bring up FORM 1 again. If the application requires that a series of forms must be filled out, the chance exists that the operator might neglect to write the data to diskette before loading the next form and beginning to fill it out. The Auto-Link feature solves that problem.

Automatic Linking

Where the application consists of, say, a three part form, we want to link to the next part of that form right away. Automatic linking begins fetching the next form as soon as the WRITE DATA key is pressed. The LOAD NEXT FORM requirement is made unnecessary. This eliminates the possibility of forgetting to WRITE and LOAD, as pressing the WRITE DATA function key will do both.
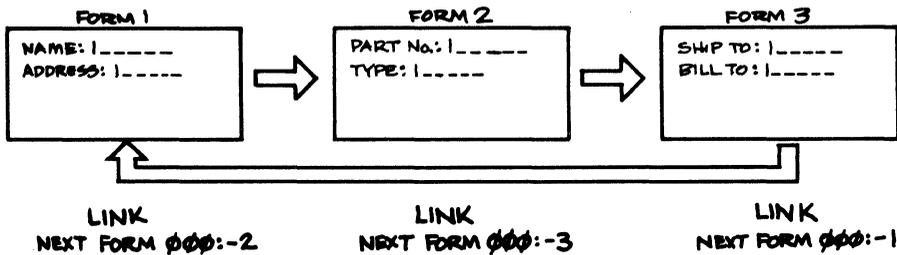
AUTOMATIC LINKING
Drawing No. 10

Automatic Linking is accomplished by the same LINK command, but a minus sign (-) must prefix the link numeral. For example, filling in the message space with NEXT FORM = 000:-2 specifies that the form assigned number two will be linked automatically when the WRITE DATA function key is pressed.

For a working environment where the same series of forms will be filled in, the Automatic Link will be most useful. In fact, you probably should make most of your applications run under Auto-Link to save time, data errors and keystrokes.

REVIEW - Form Generator Concepts

We've covered the seven steps that make up the features of the Level I (Form Generator System). It's not necessary to use all of these features. In fact, once the form is created, all the steps can be bypassed eliminating any data editing, optional programs or linking.

To help clarify what the various steps accomplish, you can imagine each step creating an overlay to the original form. For example, if you could imagine all the steps recorded somehow on clear sheets you could look through all at once and see the net result of the steps or 'passes'.
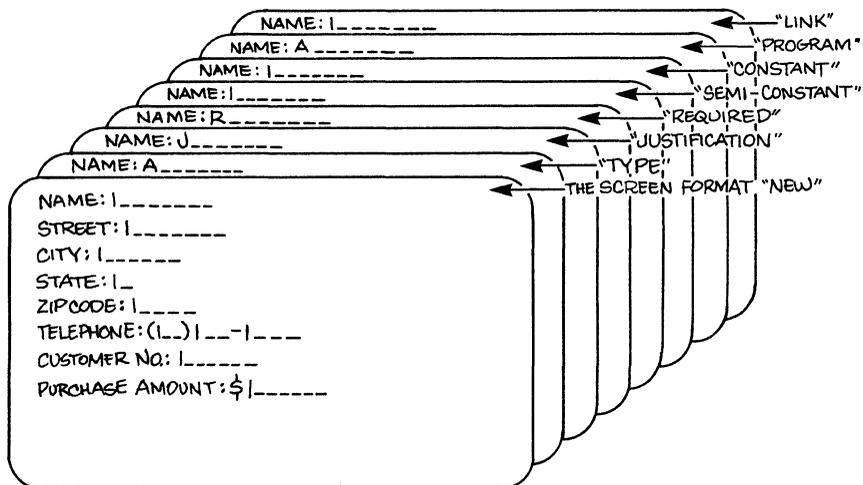
A COMPOSITE OF THE FORM GENERATOR STEPS
Drawing No. 11

PART III
Building a Complete Data Entry System

Now that we've covered all the entry editing features of DATAFORM and dazzled you with its capability (or perhaps you've jumped directly to this point) let's actually run through all the steps and put together a sample system. The object will be to create a form, assign some field-checking and editing criteria to it and finally use this new form to write data onto the diskette. The end result will be a working data entry system.

To accomplish this, you'll need the following items:

1. A Datapoint Diskette 1100 System with at least one drive and preferably two.

2. A pre-recorded diskette containing the DATAFORM form generator and language. This diskette is item DF11SYS 1.1 Order Model Code 20174.

3. Several blank diskettes.

4. If your Datapoint system has a printer attached, great. DATAFORM will work with any of the Datapoint printers.

Before we begin warming up the Diskette 1100 system for our data entry operation, take a few moments to consider the diskette you're holding. It's a remarkable device capable of storing about a quarter of a million characters and locating those stored characters very quickly. To assure a long lifetime for your diskettes observe a few simple rules in their handling

Do not lay the diskette down anywhere without its envelope on it, as it picks up dust and lint which cause errors.
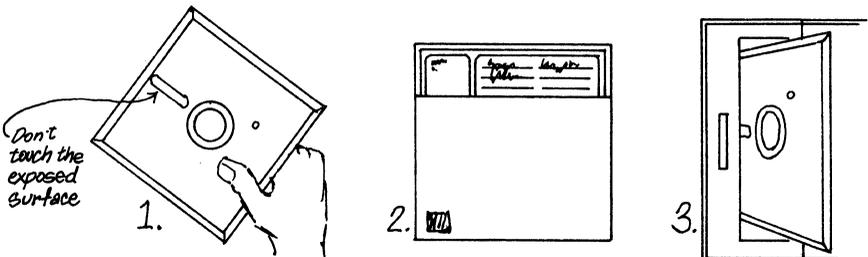
THE ONLY THREE PLACES FOR A DISKETTE
Drawing No. 12

There are normally only three places for a diskette -- in your hand, in the drive or in the envelope.

Do not write on the diskette with a ballpoint pen because it makes an impression on the recording surface. Also, keep the diskette away from magnetic fields such as those generated by heavy electrical equipment (and magnets!). While the diskette is often referred to as a 'flexible' disk, bending can injure it and the name shouldn't be taken literally.

You'll notice several holes or cutouts in the paper holder (not the removable jacket) - these are where the drive mechanism touches the surface. Be careful not to touch the surface or you might find the drive trying to decode your fingerprint!
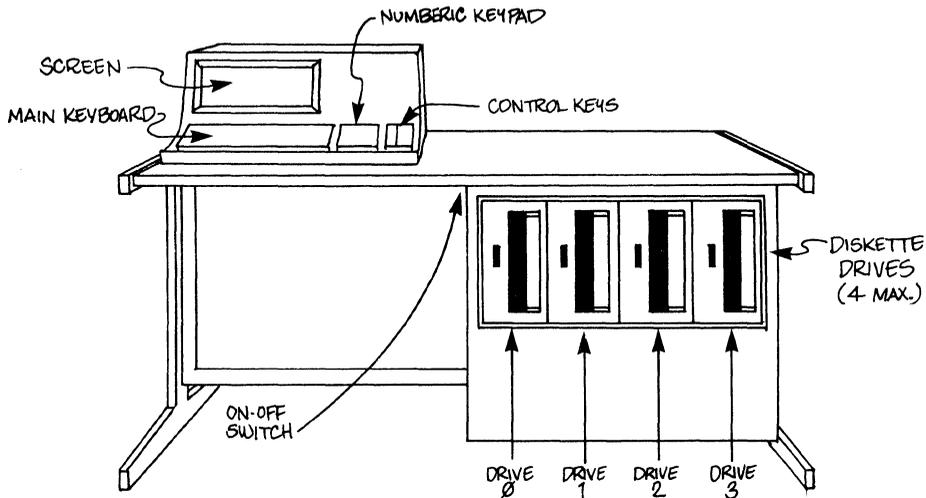
THE DISKETTE 1100
Drawing No. 13

Operating the DISKETTE 1100

Initially, make sure the Diskette 1100 System is running by plugging it in and switching it ON. The Processor ON-OFF switch should be left in the ON (rear) position, and the Diskette System should be switched ON using the console ON-OFF switch by your right knee as you sit at the console. You can tell that the 1100 is running by the little lamps located under the RUN and STOP keys -- the STOP key will be lit initially.
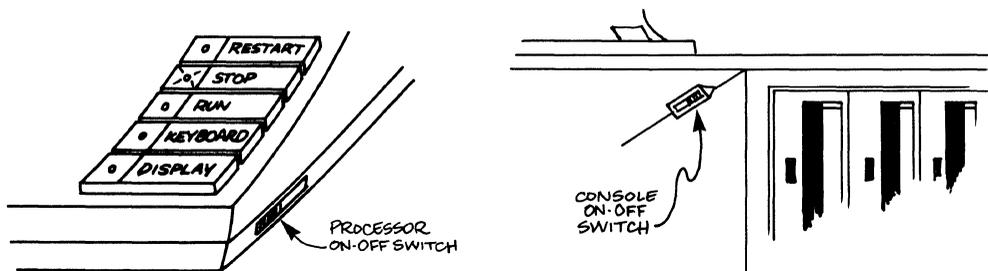
CONSOLE & PROCESSOR POWER SWITCHES
Drawing No. 14

For a complete pictorial guide to operating the equipment, see a copy of "Guide to Operating Datapoint Equipment", Model Code 60252.

Duplicating the DATAFORM Diskette

It's a good idea to create a second copy of the DATAFORM diskette. Chances are very good that nothing will happen to the diskette, but if you lay your Cuban cigar on your only copy then it's tough to go on unless you happen to have an extra.

Incidentally, all data processing media, such as diskettes, cassettes and disks, should be regularly duplicated. If you manage to harm or lose one, the volume of information lost can be staggering. So staggering as to make the cost and time to duplicate the media immeasurable.

See Appendix E for the duplicating procedure and copying your diskettes. Cassette duplicating is also covered.

Section 1. Generating the Screen Form

Step 1. Running the Operating System

The diskette marked DF11SYS goes in the drive nearest the processor - Drive 0. Normally, the first item is numbered 1 but in the computer trade, numbering starts at zero. Naturally, the second drive is called Drive 1.

To insert the diskette, slide the door open on Drive 0 and slide the diskette in until you hear a faint click and the diskette is securely locked in. Be sure the diskette faces the correct way (see the drawing). Then slide the black door over the slot opening. The diskette is now loaded and ready for use.
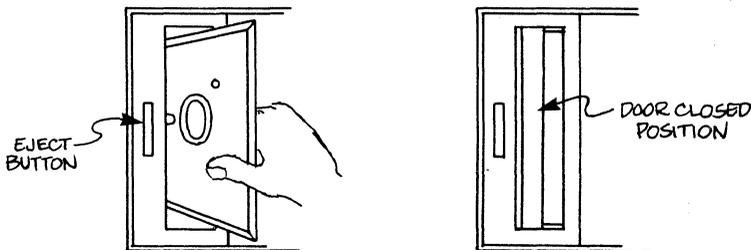
INSERT THE DISKETTE IN THE CORRECT POSITION
Drawing No. 15

To begin running the Diskette Operating System, press the RESTART key, which is located on the far right side of the keyboard. This begins the 'DOS' program which enables you to run other programs. After pressing the RESTART key the screen will look like this:
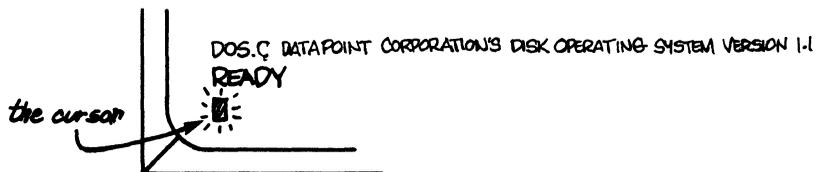
THE PROCESSOR SCREEN CURSOR
Drawing No. 16

You'll notice a blinking rectangle of light appear under the message. This marker is known as the cursor and shows you where a letter would appear if you typed a character from the keyboard.

Step 2. The Form Generator

You need to ask DOS (which is now running) to load the Form Generator program (which is on the diskette in drive 0) and to assign the yet-to-be-created form a name and a number. We'll use SHOP01 as an identifier for our first form for the Green Earth Garden Shoppe. The name that you give to your form must contain no more than six alphabetic characters; the number must be between 1 and 99.

A few notes about the typographic conventions we use in this book:

1. The ⤸ mark means to press the ENTER key, which is located where the carriage return key would be on a typewriter. The mark actually never appears on the terminal.
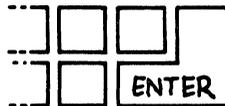
DATAPOINT PROCESSOR ENTER KEY
Drawing No. 17

2. To correct a typing mistake, use the BACKSPACE or CANCEL key to backwards erase one character at a time. Characters can be corrected this way until ENTER is pressed, at which time the typed in information is processed. Here is how our terminal looked when we entered the name of the program to be run and the name of the first form:
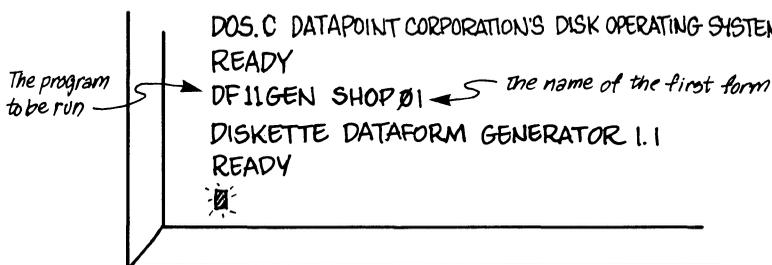
RUNNING THE FORM GENERATOR PROGRAM
Drawing No. 18

Now you must type in OLD or NEW. If you're revising a form previously keyed in, type "OLD⤸" to fetch it for revision. If you are starting work on a new form, type "NEW⤸".

Step 3. Creating the Form

After typing NEW, the screen will blank out and the cursor will appear in the upper left hand corner.

You now are ready to go through pass 1 of form generation.

The cursor will appear on the screen. To move the cursor around the screen, press a key from the numeric pad while holding down the DISPLAY key. Each numeric key when used this way, performs a different function. The following drawing shows the many functions possible.

	Cursor (8) Up ↑		
	(5) Duplicate Character		
Character Insert	7	8	9	Erase Frame
Cursor Left ←	4	5	6	Cursor Right →
Word Remove	1	2	3	Form Expand
	Ø	•	Erase Line
	Character Remove (Ø)	Cursor Down ↓ (2)	

DATAFORM Form Generator Functions - Use Display Key
Keyboard Key Causes Repeat Function

this overlay is actual size. You can copy the page and cut out the template.

o	KEYBOARD

o	DISPLAY
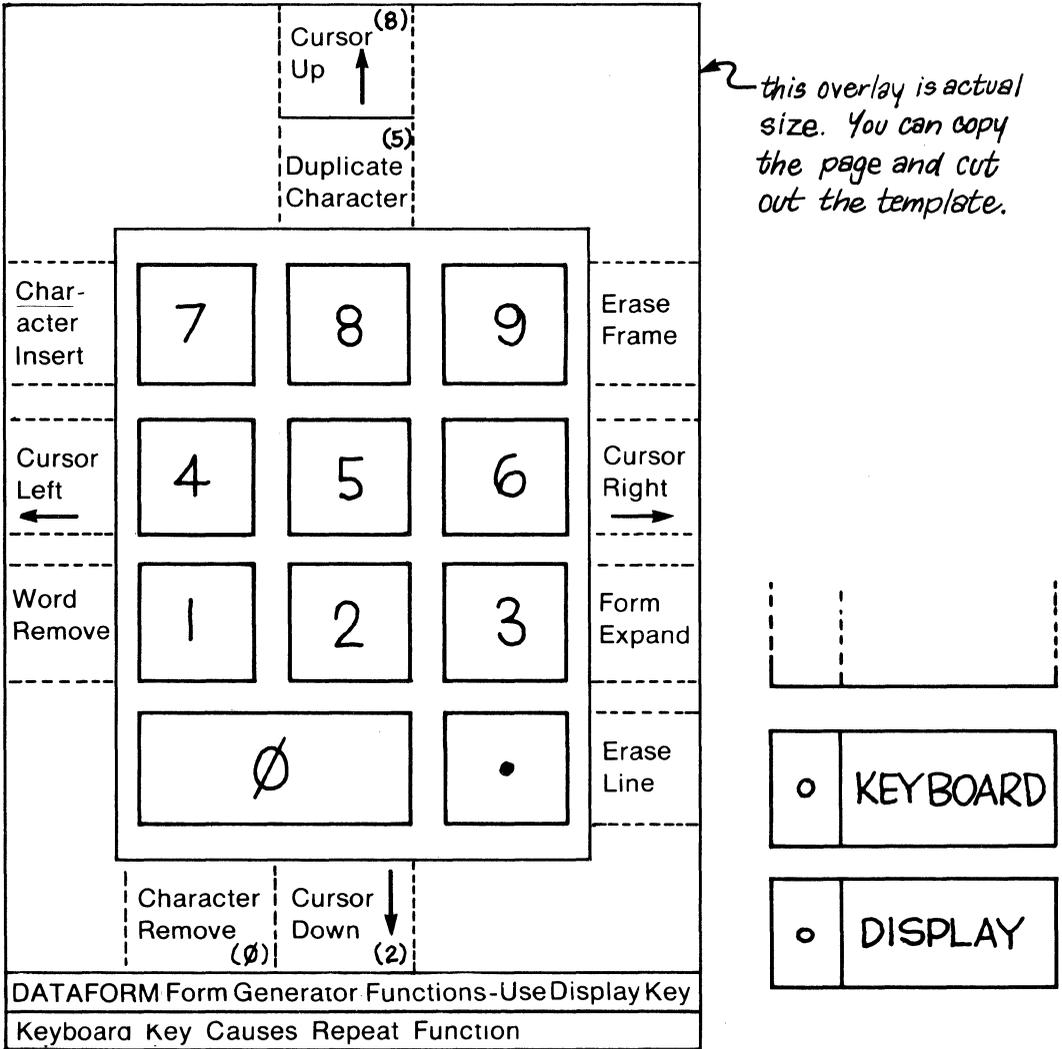
FORM GENERATOR COMMANDS VIA NUMERIC KEYBOARD
Drawing No. 19

Using these keys, the cursor can be moved around the screen and a variety of other functions performed. Holding the KEYBOARD key down, along with another key or key combination causes that character or function to be repeated. (Remember that these special functions are only used for generating the screen form -- these same keys have different meaning in data entry operations).

Probably the best way to become familiar with the screen format generation process is to sit down and type characters on the screen

and then try out all the buttons to see what happens. For purposes of simplicity, you can create any form by using the cursor movement keys and the BACKSPACE and ENTER keys.

As a first project you can type in the following Garden Shoppe's form. This form is shown in detail so if you're jumping into this project mid-book, you can cookbook together this form. We'll be using this form throughout most of the book.

THE COMPLETE GARDEN SHOPPE SCREEN FORMAT
Drawing No. 20

After you've typed the form to correspond to our final format above, hold the DISPLAY key down and press the CANCEL key. While this sounds strange, this combination of keys tells the Form Generator program that you've finished typing or arranging your form and you wish to proceed to the next step. Pressing the DISPLAY and CANCEL keys together places your screen format in memory, and tells the Form Generator program that no more changes will be forthcoming. The display "nnnn DATA" and "nnnn BYTES LEFT" will then appear (nnnn means some numbers will be represented here).

Step 4. Assigning Edit Types
Remember the types of restrictions we could place on each field? Here's a refresher:

A Uppercase Alpha (A to Z) and Space Bar; left justified and blank filled

D Numeric Digits (0-9); left justified and blank filled

N Numeric Field -- Digits (0-9), one decimal point and a leading minus sign; left justified and blank filled

M Mixed -- Combines Alpha (A) and Numeric (N)

O Minus-Overpunch -- Typing a minus sign in this
Numeric field causes the rightmost non-space character
to be overpunched (i.e. a '1' becomes a 'J')

To assign these restrictions to fields, type 'TYPE2' and your form will
reappear with the cursor waiting for your first entry. Type in the
appropriate code and hit the ENTER key. The vertical bar will be
replaced by the letter which you entered. You can elect to skip over
the field by hitting the ENTER key and assigning no restrictions. If
you make a mistake, keep going. After the last field has been
assigned, the cursor will jump back to the first field giving you a
chance for second thoughts or error corrections. The CANCEL key
may be used at this time to clear unwanted letters. The finished
screen now looks like this:

```
NAME: A_____
STREET: M_____
CITY: A_____
STATE: A_
ZIP CODE: D____
TELEPHONE: (D__)D__-D____
CUSTOMER NO.: D_____
PURCHASE AMOUNT: $N_____
```

ASSIGNING TYPE'S TO FIELDS
Drawing No. 21

Once you've done that, press the DISPLAY and CANCEL keys again --
READY will reappear.

Step 5. Assigning Field Justification / Fill

JUSTIFY provides left or right justification and blank or zero fill
to fields whose default condition does not satisfy your requirements.
The following is a nutshell review of the JUSTIFY codes:

J Right justification and leading blanks

Z Left justification and zero filled

R Right justification and zero filled

Type "JUS2" and enter a 'J' at the beginning of the Purchase Amount
field.

```
NAME: I_____
STREET: I_____
CITY: I_____
STATE: I_
ZIP CODE: I_____
TELEPHONE: ( L_) L_- I____
CUSTOMER No.: I_____
PURCHASE AMOUNT: $ J_____
```

ASSIGNING JUSTIFICATION & FILL TO FIELDS
Drawing No. 22

Once filled in, type DISPLAY/CANCEL.

Step 6. Assigning REQUIRED Fields
     REQUIRE forces the operator to fill in certain fields according to the following codes:

     R Required Field (at least one character)

     F Fill controlled Field

     B Required and Fill Controlled

     P Program Reserved (no keyin permitted)

     S Required and Program Reserved

     K Keyin-continuous (automatically entered with the last character)

     X Required and Keyin-continuous

Type "REQ⟩" and fill the following when the form reappears:

```
NAME: R_____
STREET: R_____
CITY: R_____
STATE: R_
ZIP CODE: F_____
TELEPHONE: (F__)F__-F___
CUSTOMER No.: I_____
PURCHASE AMOUNT: $R_____
```

ASSIGNING REQUIRED FIELDS
Drawing No. 23

Once filled in, type the DISPLAY/CANCEL combination.

Step 7. Assigning SEMI-CONSTANT Field Data
Our example has two fields which will usually remain the same. Type "SEM⟩" and now type in the needed text. Use the ENTER key to skip over fields which will have no SEMI-CONSTANT text. Ours will look like this:

```
NAME: I_____
STREET: I_____
CITY: I_____
STATE: MN
ZIP CODE: I____
TELEPHONE:(612) I___-I___
CUSTOMER NO.:I_____
PURCHASE AMOUNT:$I_____
```

ASSIGNING SEMI-CONSTANT FIELD DATA
Drawing No. 24

Once filled in, type DISPLAY/CANCEL again.

Step 8. Assigning CONSTANT Field Data
This function works exactly like the SEMI-CONSTANT feature except the operator cannot change the contents of the field. Type "CON⟩" and proceed as in the previous step. If you need CONSTANT spaces, enter the caret (Λ) for each space desired. Positions not set to some character are not valid. Our example uses no CONSTANT fields.

Step 9. Assigning Programs to Fields
In many cases you might want to have a DATAFORM program operate on a field. The programs can be named A through Z and you can assign them by typing "PRO⟩" and then typing in the appropriate letter in the field. The chapter on Level II covers this in detail. It's handled, however, in the same manner as other steps. DISPLAY/CANCEL gets you out of this step also. Our example uses no programs, so skip this step entirely.

Step 10. LINKing to Other Forms
By typing the command "LINK⟩" the message "NEXT FORM = 000:" will appear. The 000 means you haven't set any links yet (unless you're revising an old form and then the number will be the previously assigned link) and, if you like, you can provide for manual linking to another form by completing the message as NEXT FORM = 000:002, or more simply NEXT FORM = 000:2 if you're aghast at typing too many zeros.

Remember that the forms were numbered when we used the Forms Generator (SHOP01). If we wanted this form to link up with

another form, such as SHOP02, then the LINK would be instructed to link to form 002 and this number would be typed in now. Our example used only one form, so this entire step is omitted.

Step 11. Recording the Finished Form onto the Diskette

If you've been following the example, you should have completed all the necessary steps and are left with "READY" on the screen. Type "OUT↲", and when the completed form is written to diskette, the messages appear on the screen, as shown in the following illustration:

OUT
PROGRAM BASE ADDRESS: nnnnn ← n's are numbers
nnnn BYTES LEFT ← this is how much form space is left
DONE - LOAD NEXT SYSTEM
DOS.C DATAPOINT CORPORATION'S DISK OPERATING SYSTEM VERSION 1.1
READY
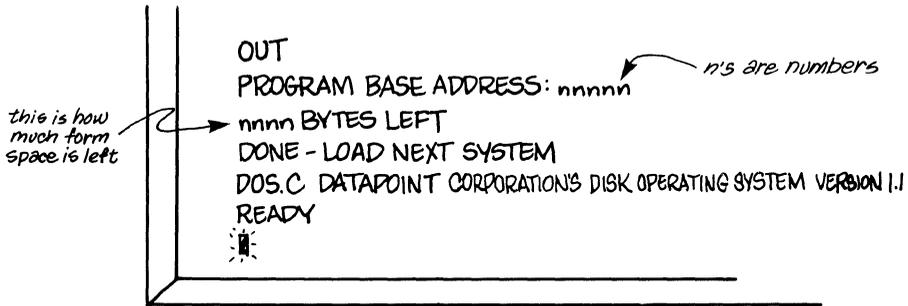
SCREEN MESSAGES AFTER LOADING A FINISHED FORM
Drawing No. 25

At this time any mistakes you made will be listed. If your form contains too many fields or requires too much data area (both are very rare events) then a message "nnn BYTES OVER" will appear and you'll have to condense the form (information in Appendix D will help you with this). One byte is equal to one character.

You can now type "OLD↲ "or "NEW↲ " in again and revise SHOP01 or type in another form if you wish, such as SHOP02.

Note: Had a program letter been assigned during the program assignment pass, the message "PROGRAM MISSING" would also be displayed, should you not have a program written for this form on diskette. Level II DATAFORM explains program generation.

Step 12. Modifying a Previously Generated Form

If you've recorded a form on diskette, it maybe easier to change it rather than to start fresh. Begin at Step 1 but type "OLD↲" rather than "NEW↲". If only edit criteria is to be changed, do not type in "REVISE↲", but instead enter the edit (i.e., TYPE, JUS, etc.) which is to be changed. REVISE is used only when the format of the form is to be changed. The old form will be displayed when you type "REV↲" or when new edit restrictions such as "TYPE↲" are entered. If REVISE is used, all the edit criteria must be re-entered. DISPLAY/CANCEL, then "OUT↲" will then write the revised form out to diskette.

PART IV
The Finished Data Entry Package

This is the end result of our labors. During the Data Entry phase, the operator sits before the Datapoint, follows the cursor on the screen form and loads data onto the diskette. Before we begin to type in the data, let's look at what's been created so far, and where we're headed.

We have, out on diskette, a file called SHOP01/DFF. The /DFF was added automatically by DATAFORM to note that it's a "DATAFORM FILE".

Another file, named SHOP01/DFD will automatically be created when we begin to keyin the data using our form. The /DFD stands for "DATAFORM Data", logically enough.

If you are skeptical about this, type in CAT₂ (short for CATALOG) and the Operating System will show you all the files on the diskette. Try CAT after you've finished doing the example - the files that weren't there before will now be part of your working system.

By the way, the diskette files you will be working with can be automatically expanded, ended or modified without regard for the complexities that accompany conventional complex systems.

To begin the actual data entry operation type in the command as follows:

```
READY
DFII SHOP01
```

THE START OF THE DATA ENTRY OPERATION
Drawing No. 26

Two lines then appear on the screen - "DISKETTE DATAFORM INTERPRETER 1.1" and "SELECT DATA MODE". To start a new file, type "Start₂" or "S₂".

The DATAFORM interpreter responds with "START FILE?" as a safeguard to insure that you know that you are starting a new file (therefore, if one exists, it will be written over). Type "Y₂" to start a new file or "N₂" if you now realize that you typed "S₂" by mistake.

So, if you have actually followed our example to this point, the Garden Shoppe form would be glowing on the screen and now you're ready for the data to be entered!

```
NAME: ADAM SMITH
ADDRESS: 123 WALL ST.
CITY: N.Y.
STATE: NY
ZIP CODE: _ _ _ _ _
CUSTOMER NO.: _ _ _ _ _
PURCHASE AMOUNT: $_ _ _ _ _ _
```

YOU NOW HAVE A WORKING DATA ENTRY SYSTEM
Drawing No. 27

Data Entry Operator Controls

Note that the cursor appears at the beginning of the first non-Constant field to be filled in. You can try filling in a few fields to get the hang of it. Note that a click is heard each time a character is typed and accepted and that a beep is emitted if you try to type alphabetic characters in a numeric-only field, or if any other error occurs.

During the data entry process (DATAFORM Interpreter) two programs might be considered to be running at the same time -- a type of Monitor and the Data Entry (fill-in-the-form) program. The first message to appear was written by the Monitor. Typing "START⟩" brought the form up on the screen and began the data entry process. There are many reasons you might want to get back to this Monitor and we'll be discussing these in the section called "Data Entry Operation Modes".

To make the data entry operation human-oriented and easy, the numeric pad is again used for special functions. The illustration below shows a template with the data entry functions indicated. As before, with the Form Generator, the DISPLAY key must be pressed in conjunction with the Numeric Pad keys, to make use of the special functions.

Backspace
Record

Rewind
Data
File

Read
Record

Monitor

Erase
Form
Data

Load
Next
Form

Back
Field
Tab

Write
Data
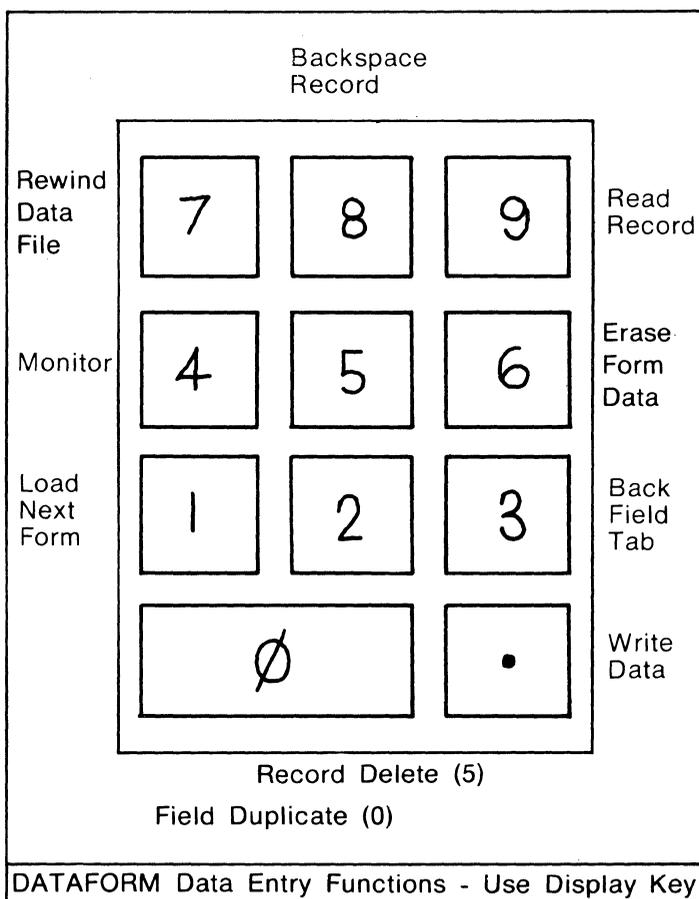
This overlay is actual size. You can copy the page and cut out the template - or, more durable plastic ones are available.

Record Delete (5)

Field Duplicate (0)
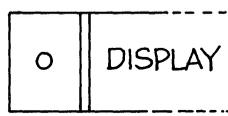
o ‖ DISPLAY

DATAFORM Data Entry Functions - Use Display Key

DATA ENTRY COMMANDS VIA NUMERIC KEYBOARD
Drawing No. 28

Besides these, the ENTER, BACKSPACE, and CANCEL keys are also used during data entry. You'll want to try out each function, so an explanation is listed here.

DATA ENTRY COMMANDS

Tab Forward and Backward.

The ENTER key is used as a forward tab (jump to the next field) and the Backspace Field function (DISPLAY/3) permits backward tabbing. Forward tabbing past required fields is not permitted as the operator must enter at least one character in these. Note that Alpha or Numeric editing occurs while the field is being keyed into. When the field is complete, further editing is performed on numeric and right justified fields to insure compliance with format restrictions. Also if you had assigned a Level II program to operate on this field, the program would not be run until all other editing had been performed successfully.

When errors are detected in a field, instead of moving to the next field, the cursor will be placed at the beginning of the field just entered and a Beep will be sounded. The illegal data will not be erased automatically but will be written over when the correct data is typed in. If the operator decides to tab past the field, the last accepted data (blanks if none has been entered) will be displayed.

Field Erase - (CANCEL key).

The CANCEL key causes the current field to be erased and the cursor repositioned to the start of the field.

Form Data Erase - (DISPLAY and 6).

Simultaneously pressing the DISPLAY and 6 keys clears the data without writing it to diskette and redisplays the cleaned form, plus Constants and Semi-Constants.

Field Duplication - (DISPLAY and 0).

Once a form has been completed, the data can be written to diskette (see Write Data). The data entered during this time is stored in memory such that it is available to the operator for field duplications through use of the DISPLAY and 0 keys. This causes the field data, from the same field of the previous form which was written to diskette, to be written into the present field. Using this feature, you can easily enter data where much of it is the same. The differences between forms can be edited in, rather than retyping all the data fields.

Write Data - (DISPLAY and Period keys).

The DISPLAY and period keys cause the form data to be written to diskette. If only a single form is used, all data fields will be cleared (except for the Constants and Semi-Constants) and the form will be redisplayed with the cursor waiting in the first non-Constant field. If Auto-Link is specfied for the form, the linked form is automatically loaded and displayed. NOTE: unless Auto-Link is used with the automatic write feature, it is possible to Load Next Form without having written the data to diskette; beware of this hazard.

Load Next Form - (DISPLAY and 1).

The next form specified by the LINK criteria entered on the form will be brought into memory when the DISPLAY and 1 keys are pressed. If no link has been specified, the same form will be cleared of data and redisplayed, or possibly the words "BAD FORM" may be displayed and you may be returned to Monitor. If the latter occurs, make sure you close the data file.

Delete Record - (DISPLAY and 5).

The DISPLAY and 5 keys cause the current diskette data record (the data currently entered on the screen) to be deleted from the data file.

Return to Monitor - (DISPLAY and 4).

The DISPLAY and 4 keys cause the Data Entry Interpreter to return to the Monitor. Only in the Monitor mode can the various data entry functions be utilized. See "Data Entry Operating Modes" which follows.

Data Entry Operating Modes

During use of the DATAFORM level I package a file of records is created on diskette. This diskette file is the end result of the intelligent data entry operation. Many times an operator will find it necessary to change previously entered data or examine it or add more data to an existing file.

DATAFORM, in addition to easy form generation, provides a powerful facility for modifying (editing) data files. The operator reviews entered data. If a print-out of the outward data is required, there's a facility to do that, too.

The Data Entry Mode permits data to be keyed in — the data file is open

The Monitor Mode permits the operator to examine or modify entered data — File must be closed
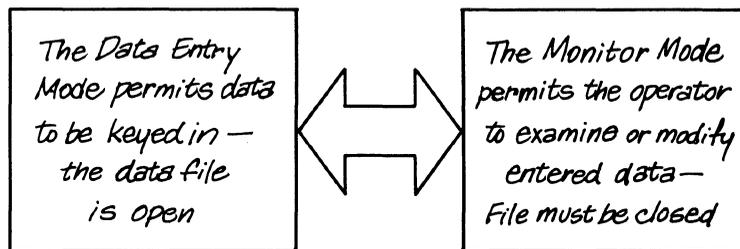
THE TWO MODES OF DATAFORM
Drawing No. 29

When we first keyed in "DF11 SHOP01ʔ", we told the Operating System to get the Monitor running and also look in the file named SHOP01 to find the first form. After that, we were in the Monitor Mode and selected our first task to be START. These are often useful things that the Monitor mode can do.

The one thing to keep in mind is the file you're working with. If you're in Data Entry Mode and want to use one of the Monitor functions, be sure to close the file with the END Monitor command. This puts a marker on your file and assures data integrity. You can always re-open the file for later operations.

Writing a Beginning of Data Mark (START)

As we mentioned earlier, begin every initial data entry operation with START. If you use START with an existing data file, however, you'll wipe out all the old data. You can type "START 001⟩" (or 'S˙ 1') or any valid form number to begin with a specific form. If you keyin simply "START⟩", DATAFORM will select Form 1. Since we want to open a new data file, answer YES to the question "START FILE?". This question is asked to make sure you don't destroy a needed data file.

Writing the END of Data Mark (END).

Whether you're finishing up with a data entry project or closing up shop for the day you must write an END mark on the data file to preserve the good data. After doing your last Write Data, press DISPLAY/4 and when "DF-READY" appears, you'll be in Monitor mode. Then you may type "END⟩ ". The question "END FILE?" should be answered "YES⟩", unless you know that you are positioned somewhere in the middle of your data file. The data file will be 'Closed' -- up to that point the file is considered 'Open'. "END" should be needed only when entering data under ADD or START conditions.

Adding to an Existing Data File (ADD)

Suppose you have ended your data entry job for the day and someone comes breathlessly running in with a last minute record to be added. No big problem. If you're in Monitor mode, type "ADD⟩" and specify a valid form number. The Monitor will fetch the form number you indicated. While you are typing the new record, DATAFORM will find the End of Data mark and position the diskette to write the next record. You may now key in any number of records. (Be sure you type "END⟩" when finished.)

Modifying an Existing Data File (MOD)

The same person who handed you the additional data might well have informed you that an earlier entered record needs to be changed. By typing "MOD 01⟩" or any valid form number in the Monitor mode, the diskette finds and displays the first data record to allow you to search forward for the record you need to correct. Pressing Read Record (DISPLAY and 9) advances the file forward and displays each record (only data records created by the selected form are displayed). Pressing Backspace (DISPLAY and 8) permits you to read each record in the reverse direction. Additionally, you can position to the beginning of the data file by using Rewind (DISPLAY and 7).

When the data in a record is displayed, you have two options: leave the data unchanged, or retype any field on the screen and press Write Data (DISPLAY and Period) to overwrite the original record.

Switching to Data Entry Mode (DATA)

If you return to the monitor and find that you need to correct your latest data entry and you haven't typed END yet, type "DATA⟩" to return to Data Entry mode.

Review the Data File (REW)

    This function takes you back to the beginning of the data file and displays the first record. You must be in Monitor mode when you type "REW⟩" (or "R⟩"). This function is particularly useful when used with the MOD or FIND functions. Use the numeric keys to advance through the records.

Locating and Modifying a Specific Record (FIND)

    If you need to find a certain record, and you know that it contains some unique field entry such as "125 SHELL ST." in the Street field, that record can be quickly located using FIND. The specified form will be displayed so that the operator can type into any field he wishes to use as a key in searching the data file. It is also permissible to key into more than one field at a time for each search. Type all the information into the fields you wish searched. The Monitor will compare the records with the data entered for a match. After the appropriate field(s) have been filled, press the Read Record function (DISPLAY and 9) to initiate the search. If a match is not found, the message, "END OF DATA" is displayed and control is returned to the Monitor. Be careful not to do an END after a FIND. To halt a record search, press both the KEYBOARD and DISPLAY keys simultaneously. The search will halt and control will return to the Monitor; the file will be intact.

Operating the Finished Data Entry System

    You've created a working, sophisticated intelligent data entry system! Now, if you're in a production environment, show the operator how to enter the commands and use the numeric pad. Then, all that needs to be done is to have the operator practice a little with the system that you created. He or she will get the hang of it very quickly. No technical education is needed to produce accurate and rapid data entry. Remember, that "START" will start your data file anew, and erase all your operators practice data. Working (or "live") data may now be entered by the operator. Note that when a form is completely filled out and written (DISPLAY and Period) to diskette, the same form will pop up on the screen automatically with all fields blank. The cursor will await your keyin in the first non-Constant field. If you have no more data to enter, just go to Monitor (DISPLAY and 4) mode and type "END⟩". Remember to answer the question "END FILE?".

    Once your good data has been written to diskette and ENDed, the data may be processed on the Datapoint 1100 by other programs or transferred on telephone lines using DATAPOLL or a number of the other communications programs (see the DATAPOLL Simplified User's Guide).

    If you are done for the time being, and you've ENDed all data files, you can leave the Datapoint 1100 by opening the black door in Drive 0, pressing the EJECT button, and taking out the DATAFORM diskette (putting it in its protective wrapper, of course). If no one else will be using the Datapoint 1100, you will want to turn the console ON-OFF switch (right by your knee) to OFF.

The Diskette 1100 and DATAFORM are economical and fast. This strongpoint allows you to quickly generate a form and turn it around for Data Entry use -- very impressive if you're trying to convince someone of the virtues of intelligent data entry; for example a sales or corporate management demonstration. You could actually create a form while they watched, type in the editing features and then enter data.

Where to Go from Here

Now that you're competent in Level I, you have a choice: get busy and put the Datapoint to work with Level I or keep reading to see if the Level II programming power might be useful.

PART V
A SUMMARY OF DATA ENTRY COMMANDS
(Monitor Mode)

Function	Key	Operation
START n	S n	START-initialize a data file
ADD n	A n	ADD to the end of a data file
LOAD n	L n	LOAD (display) specified form
DATA	D	Switch to DATA Entry mode
MOD n	M n	MODIFY data records
FIND n	F n	FIND matching record
END	E	Write END of file on data file
REW	R	REWIND data file to first record

n means the number of the form

Data Entry Function Keys
(Data Entry Mode)

ALL DATA ENTRY:

Forward Tab	ENTER
Backspace Field	DISPLAY/3
Field Erase	CANCEL
Form Data Erase	DISPLAY/6
Field Duplication	DISPLAY/0
Write Data to Diskette	DISPLAY/Period
Load Next Form	DISPLAY/1
Delete Record	DISPLAY/5
Return to Monitor	DISPLAY/4

MODIFY & FIND:

Read Next Record	DISPLAY/9
Backspace Record	DISPLAY/8
Display Record 1	DISPLAY/7
Abort Record Search	DISPLAY/KEYBOARD

REWIND 7	BACKSPACE RECORD 8	READ RECORD 9
RETURN TO MONITOR 4	DELETE RECORD 5	ERASE FORM 6
LOAD NEXT FORM 1	2	BACKSPACE FIELD 3
FIELD DUPLICATION Ø		WRITE DATA RECORD ●

DATA ENTRY FUNCTIONS
Drawing No. 30

CHAPTER THREE

LEVEL II DISKETTE DATAFORM

Introduction

As you learned from the first two chapters, Level I DATAFORM provides a quick and straightforward means of building screen forms with some editing capability. In many cases the entire data entry task can be accomplished using only Level I capability.

Instances will arise, however, where complex error-checking or computation is required. For those cases, the powerful DATAFORM programming language may be used. This language can be used to generate short or long programs that are integrated with the forms themselves.

How to Use This Chapter

This chapter approaches the programming language in four parts:

1. A discussion of the general concepts and instructions.
2. The step-by-step operation to get an actual program running on the Datapoint Processor.
3. A thorough discussion of the DATAFORM language.
4. Program examples.

If you've never programmed before, this chapter will offer new insights into the ways and techniques of handling business information. Since space will not permit a comprehensive treatment of the fundamentals of programming, brief explanations of these concepts accompany the discussions of each topic. To assist in the learning process, many examples are used.

Necessary Equipment:

1. A Datapoint 1100 system with at least two Diskette drives.
2. The DF11SYS diskette you used for Level 1 form generation.
3. A Diskette that contains the full version of DOS.C (model code 20175). This diskette contains the Editor, which is a program that allows you to create and modify your programs.

Data Flow in DATAFORM

What is a Field Program?

The programming language contained in DATAFORM is somewhat unique in that it's field oriented. That is, all the information available to the program is contained in the fields which are filled by the person operating the data entry system. The operation of a field program may include access of all of the data, an error message with audio or visual indications or other operations such as linking to another form or writing the data to diskette.

The field program assigned to operate on a certain field performs a set of manipulations or tests which may include checking for valid data in the field, or performing a calculation to relieve the operator of that task. For example, in the Green Earth form (which we used in Level I) you might want to check that a state abbreviation is valid, (i.e., that it is one of the 50 states). Or you might want to accumulate the purchase amount for subsequent use in a batch total form.
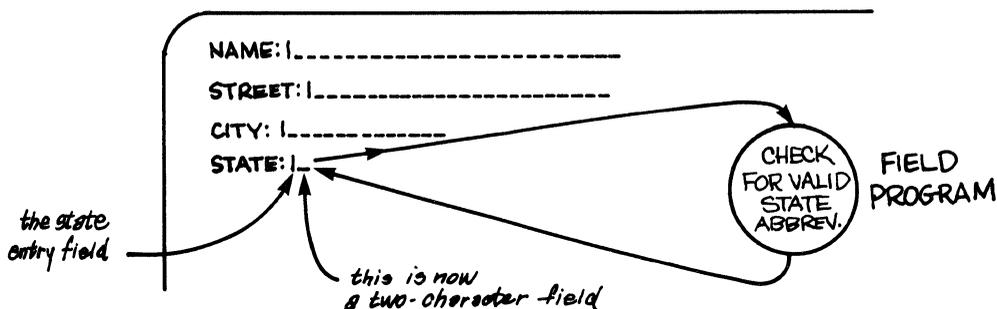
THE FIELD PROGRAM OPERATES ON DATA IN THAT FIELD
Drawing No. 31

To simplify programming, two names are used to designate data buffers (temporary data holding areas):

1.   The data keyed into the current field (INPUT)

2.   The portion of the total data record (all the information entered on the screen) assigned to that field (OUTPUT).

Before we start to build that program, let's see where the data moves without a field program.
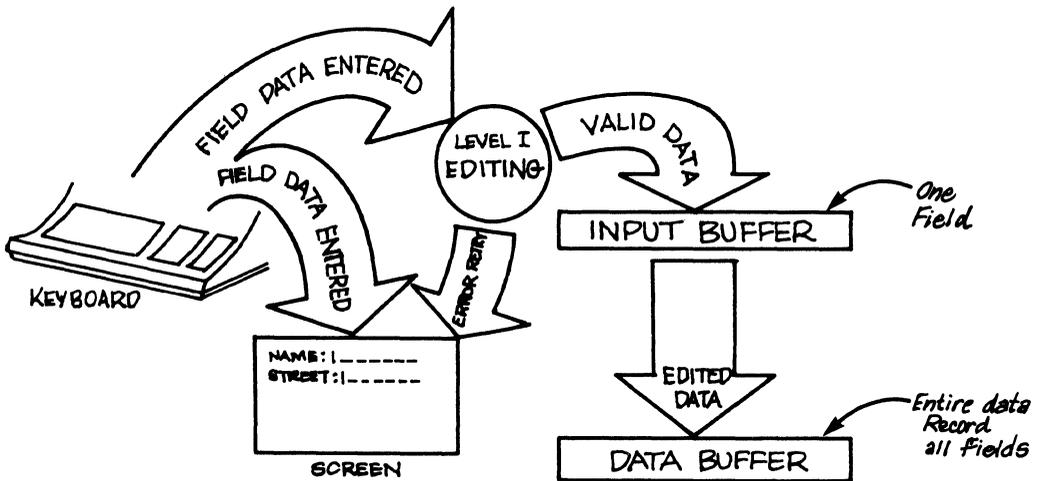
LEVEL I DATA FLOW (NO FIELD PROGRAM)
Drawing No. 32

The Data Buffers

Field Data is entered via the Datapoint Processor keyboard and, upon completion of the field, passes through the series of Level I editing criteria established during the Form Generator operations. This field data is then in the temporary "Input Buffer". (A buffer is a temporary resting place for data.) Thus, at any one time, the Input Buffer reflects all data that the operator has keyed in, while the Output Buffer contains the data to be written to diskette.
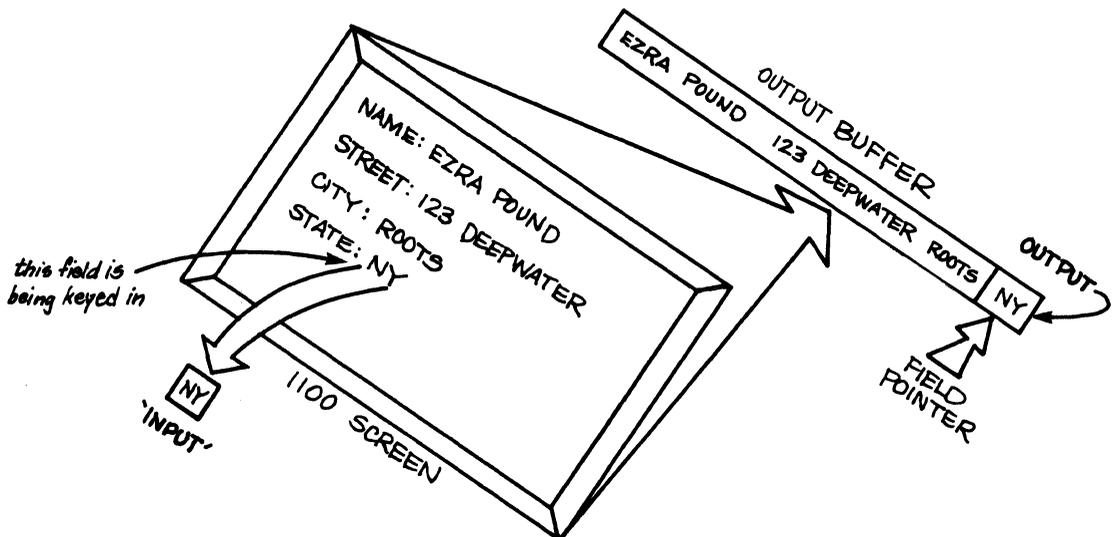
INPUT BUFFER REFLECTS DATA ON SCREEN
Drawing No. 33

KEYBOARD

DATA ENTERED

VALID DATA

LEVEL I EDITING

RETRY

INPUT BUFFER | "INPUT"

DATA

the program works on the contents of these buffers and can display on the screen if needed

DATA OR RETRY MESSAGES TO SCREEN

FIELD PROGRAM LEVEL II

DATA

SCREEN

DATA BUFFER | "OUTPUT"
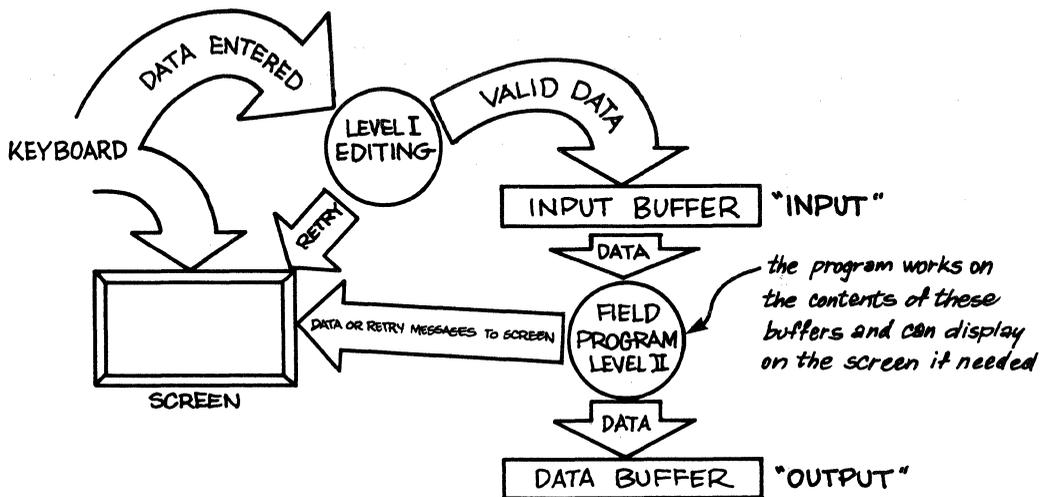
DATA FLOW USING A LEVEL II FIELD PROGRAM
Drawing No. 34

If a field program is assigned, the data that has just been entered and edited by Level I DATAFORM may now be edited by the field program. At this time, the assigned field program (the thing you're going to write) begins to operate upon the data in the Input Buffer. The data in the Input Buffer is known as INPUT and the Data Buffer is known as OUTPUT. It becomes the responsibility of the field program to move the Input data to the Output Buffer, so don't forget or you'll mistakenly write blank records on your diskette!

Since you are now well aware that DATAFORM is field-oriented, let's talk about how these fields can be accessed. As far as a field program is concerned, the data of the current field is always in INPUT and, after the program has completed it's operation on that field, it must be transferred to OUTPUT.

However, there will arise situations where you might want to compare the contents of one field previously entered against another. For example, if the shipping address state was different from the billing address state, the program could convey a warning to the operator.

To use and access fields other than the current one (INPUT), each field is assigned a number. You won't find these numbers appearing on the screen or in a print-out, but they're there. The fields are numbered left to right, top to bottom.

NAME: ①
STREET: ②
CITY: ③
STATE: ④
ZIP CODE: ⑤
TELEPHONE: ⑥⑦⑧
CUSTOMER NO: ⑨
PURCHASE AMOUNT: ⑩

FIELD IDENTIFICATION BY NUMBER
Drawing No. 35

Our Green Earth form has a total of ten fields and if for some reason we wanted to validate the State (field 4) by checking it against some other code in the sixth field we can do it in a number of ways. The simplest way is to assign a name to the state field (e.g., STATE FIELD 4) and then we may refer to that name in our validation program.

In review, you know that the Input Buffer contains the immediate contents of the field, the Output Buffer will eventually reflect all that the operator can key in on that screen form and that each field has a number.

Concepts of Programming

Although the DATAFORM programming language can accomplish a wide variety of tasks, as can any other programming language, one of the most frequent tasks will be the validation of field data. Let's look at what happens when a program begins operation on a field. We'll use a flow chart (a block diagram of logical steps) to save space.
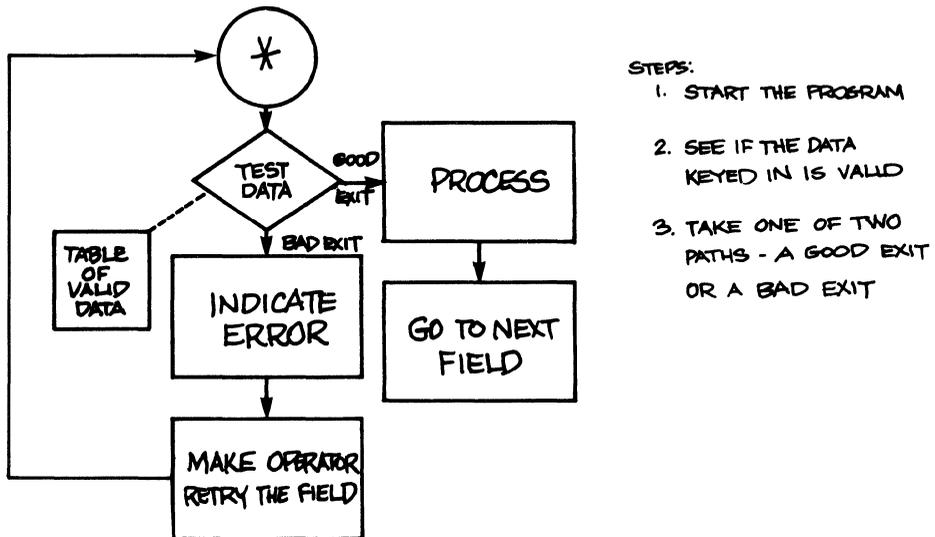
FLOW CHART OF VALIDATION PROGRAM
Drawing No. 36

The process outlined in the flow chart works exactly like a cook sorting potatoes. The program starts each time the cook picks up a potato. The "picture" of what good potatoes look like is in his mind and each comparison allows him to toss the potato in the garbage (bad exit) or in the peeling bin (good exit) for later use. This program, running on the cook's mind, allows him to pare the bad spot off the bad potato, however, and compare it again to see if it's now acceptable.

The DATAFORM program can run in exactly the same manner. Let's examine the elements of this sequence and see how we should use them.

Starting the Program

Remember that during the Form Generator section (Level I) there was a pass for PROGRAMS to be assigned. You could assign up to 26 programs (the number of letters in the alphabet) by assigning a letter to that field. Since we're going to be operating on the State field of the Green Earth form, we should have assigned a letter to that field.

CITY : L_ _ _ _ _ _ _

STATE (A)

ZIP : I_ _ _ _

the program letter is assigned during the PROGRAM pass in Level I
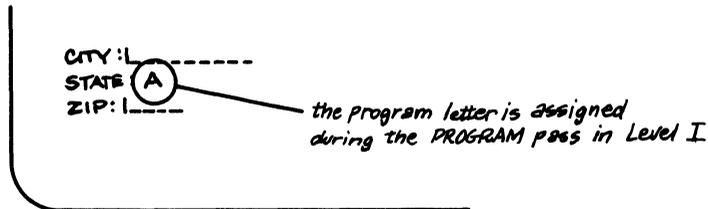
PROGRAM LETTER ASSIGNED DURING LEVEL I PROGRAM PASS
Drawing No. 37

When a letter is assigned, the field program begins running its tasks after all of the Level I validation has been finished. By the way, you can only assign one program to operate on any one field. You can't request programs A, B, and C to operate when a certain field is entered; so keep that in mind when writing your program.

Program Structure

Field programs are essentially divided into two parts. First there is the part containing all data descriptions, messages and tables. (This is what the data is compared to -- the cook's vision of a perfect potato!) The statements which set up this part of the program are called specification statements, or non-executable instructions. Their purpose is simply to set up data for the program's use; they don't actually do work.

The second part is the executable statements. The program name (such as A*) appears as a label on the very first executable statement, and it is this statement that begins the sequence of data entry process events whenever a field program is assigned. The executable statements do the work and make the exit, be it good or bad.

Use of Labels

Labels (the names that we give to statements) make it possible to reference things without being concerned about exactly where they are located in the maze of computer memory. Labels start in column one, are alphanumeric, and should not exceed eight characters.

We have seen that data just keyed in is called INPUT, that we can talk about the portion of the data buffer corresponding to the current field as OUTPUT (thus the actual location of OUTPUT changes with each new field). We can also assign names to portions of the data buffer corresponding to specific fields in the form, e.g., STATE FIELD 4. We can create additional buffers with names and contents of our own choosing as in the following illustration.

reference name or label → STATE FIELD 4 ← *the field we need to access*

FIELD ACCESS VIA LABEL & NUMERIC POSITION OF FIELD
Drawing No. 38

The "STATE FIELD 4" statement allows us to use the name STATE in our program with the result that the contents of Field 4 will be used. This technique of using a handy name to identify a field is called "Labeling" and you'll see much more of it.

The field programs have special one letter names. These are typed into the appropriate field during the PROGRAM Assignment Pass of form generation. To identify a particular label as the beginning of a program, we make the first character a letter and put an asterisk after it; program 'S' starts executing at the instruction labeled S*. Instructions within programs may also have names so that one instruction can direct processing to another point in the program.

Let's suppose that you want to see if the State, which the operator has typed in, is a valid abbreviation. To accomplish this we must build our own "buffer" filled with valid abbreviations of states. The technique of creating your own buffer is necessary if you want to compare the INPUT field to specific lists, maintain cumulative totals separate from the data that is written on diskette, or display messages when errors occur. So once you understand this technique you will have a useful tool for all kinds of applications.

To accomplish the task of seeing if the State which was keyed in is a valid abbreviation, the first step is to create the buffer. For the sake of brevity we'll only use 5 states, although we could have used all 50. Here's what the buffer looks like set up in the DATAFORM language:

STATLIST WORK "MN","MI","WI","IL","IN"
the label → *the statement* → *the data* →

USING THE WORK INSTRUCTION TO SET UP DATA
Drawing No. 39

"STATLIST" is the name that the group of state abbreviations is stored under. WORK is an instruction that informs the computer to store these abbreviations in memory for future use. Note that if there was only one valid State abbreviation the operator is allowed to key into, STATLIST could be rearranged to make sure that abbreviation was keyed in.

STATLIST WORK "MN"

A WORK AREA WITH ONLY ONE ITEM
Drawing No. 40

Executable Statements or Instructions

The tables or lists or non-executable statements (as you will), provide us a basis with which to compare our incoming data. The instructions that do work include the comparison instruction (which we'll work with) plus a wide variety of others. In this section, only the commands necessary to our program are explained. See section 3 for a complete description of the language.

For example, let's take the list of valid abbreviations labeled "STATLIST", which we created, and add the necessary executable instruction to have the field in INPUT compared against the valid abbreviation in the list in STATLIST. A flow chart will show the sequence of events.
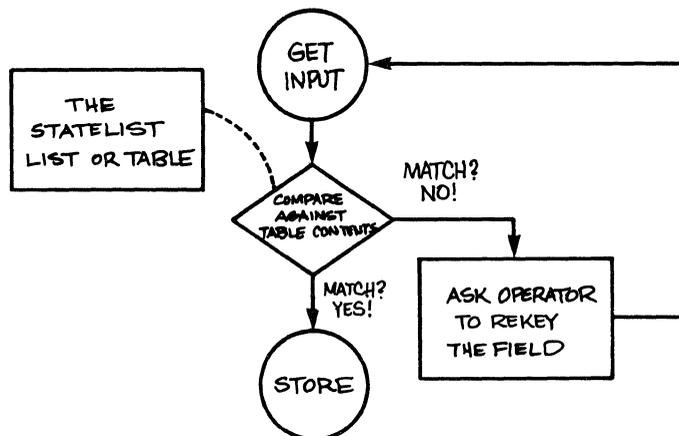
FLOW CHART OF THE COMPARISON (IF-INTABLE) PATHS
Drawing No. 41

The object will be to see if the State abbreviation, which the operator keyed in, matches one of the abbreviations in the STATLIST. This instruction comes out in almost pure English. We will use a comparison instruction known as IF-THEN.

CHECKING INPUT DATA USING IF-INTABLE INSTRUCTION
Drawing No. 42

The IF-THEN instruction is told to consider STATLIST as a table, compare INPUT to see if it's the same as any of the entries INTABLE, then if it finds it in the table, to STORE it. If it does not find the field in the table, the instruction is ignored and the program drops to the next instruction, which (for example) might be "AGAIN". Both STORE and AGAIN are exit instructions.

One of the critical elements of the field program is its termination or "exit path". It is important to understand the "exit" instruction as it affects both data movement into the Data Buffer and transfer of

50

program control to the next field or form. So the exit instruction has a lot to do with what happens next.

Exit Paths

To exit, so that data entry can resume and the Input Buffer can be written to the Output Buffer, the statement "STORE" can be used. To cause the INPUT data to be rejected and force the operator to re-enter (retype) the field, the statement "AGAIN" can be used. For example, if the State abbreviation in the Green Earth form was wrong, our problem will exit out the error path using "AGAIN". So, the bad data failure path is "AGAIN" while the success path is "STORE".

Entry Point

So we've added the executable instruction and the concept of the exit paths. Now all we need is an entry point and this program will actually run. Remember that program A was assigned to the State field during Level I. To identify the program and note the starting point we use the same A with an asterisk after it in the very first space. Remember, the first few spaces are reserved for labels. Here's the complete program ready to go.
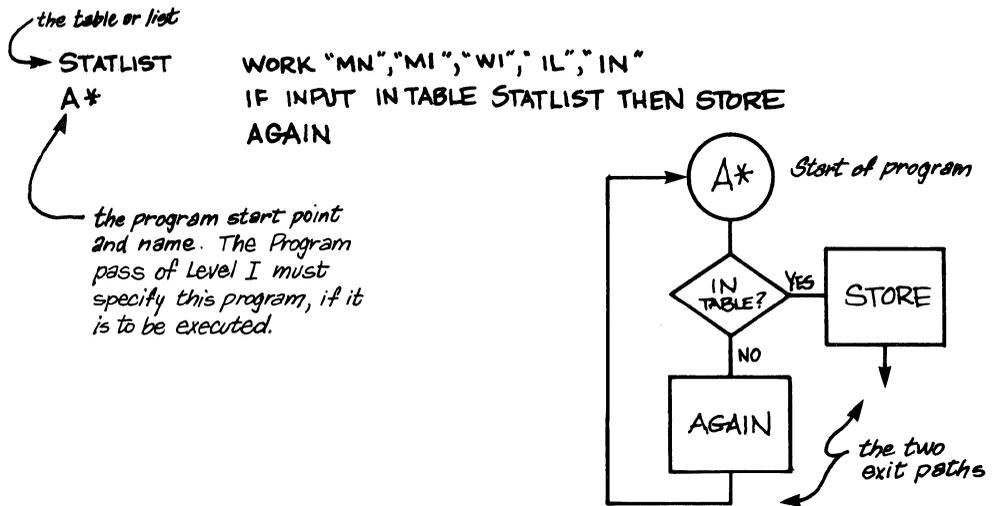
COMPLETE PROGRAM WITH FLOWCHART
Drawing No. 43

Program Results and Some Thoughts

The important thing to remember is that if the program didn't find a match, it would go to the next instruction (which is AGAIN). That's how simple the process of generating a program is. Now let's look at the program elements in detail.

In this example there is no processing done if the field is good and no special error message if the field is bad.

To expand the program let's investigate how to display an error message. The first thing to do is to set up a buffer containing the error message.

BADMSG WORK "BAD STATE"
Drawing No. 44

The new program uses the MESSAGE instruction to display the error message on the bottom line of the screen. (You recall the screen is 12 lines deep by 80 columns wide.)
Complete the program as follows:

```
BADMSG          WORK "BAD STATE"
STATLIST        WORK "MN","MI","WI","IL","IN"
A*              IF INPUT INTABLE STATLIST THEN STORE
                MESSAGE BADMSG
                AGAIN
```

Drawing No. 45

If you don't like the message on the bottom line, an alternative instruction, SHOW may be used. This instruction puts the message in the current field position. The message should be the same length as the field, however. If it is longer, only the number of positions allowed for that field will be displayed, and the rest will be truncated (or cut off) as follows:

```
BADMSG          WORK "BAD STATE"
STATLIST        WORK "MN","MI","WI","IL","IN"
A*              IF INPUT INTABLE STATLIST THEN STORE
                SHOW BADMSG
                AGAIN
```

after an error occurs using the show instruction, this is how the screen looks. Only two characters will show.

CITY: SAN ANTONIO
STATE: BA ← *this is where the message appears using SHOW*
ZIP CODE:
TELEPHONE:
BAD STATE ← *using MESSAGE, the message appears on the bottom line*
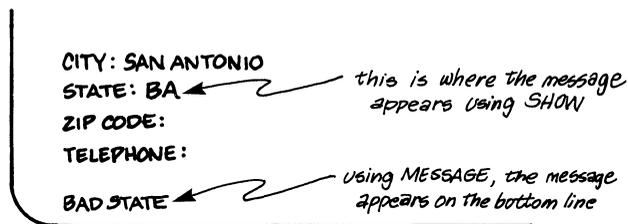
DISPLAYING ERROR MESSAGE UNDER PROGRAM CONTROL
Drawing No. 46

MESSAGE and SHOW statements may also be used to display computation or conversion results.

Generating A Code Number
Let's expand our program again so that it has a processing step. We will convert the State abbreviation to a number and then store that number and show it to the operator.

```
BADMSG        WORK "BAD STATE"
STATLIST      WORK "MN","MI","WI","IL","IN"
STATENO       WORK "01","02","03","04","05"
A*            IF INPUT INTABLE STATLIST THEN GOOD
              MESSAGE BADMSG
              AGAIN
GOOD          CONVERT INPUT BY STATLIST AND STATENO
                      GIVING OUTPUT
              SHOW
              NEXT
```

VALIDATION PROGRAM WITH PROCESSING ADDED
Drawing No. 47

This program has several new features. One of the most important involves labeling an executable instruction and, by use of that, jumping over two instructions. GOOD is the label and it is placed out against the left hand margin like STATLIST and A* were. The program has several things happening, so a flow chart might make things clearer.
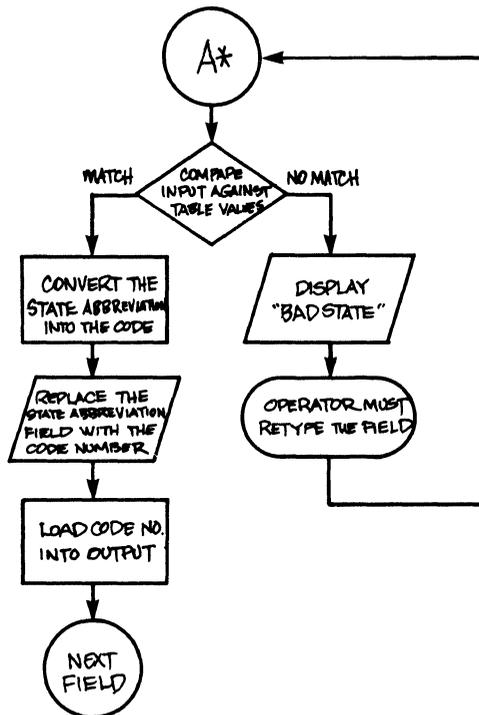
EXPANDED PROGRAM WITH CODE NUMBER CONVERSION
Drawing No. 48

Ignoring the CONVERT instruction for the moment, there are two new instructions in this program:

1.   SHOW with no name following it means SHOW OUTPUT; otherwise SHOW the contents of the field whose

name follows

2.　　NEXT just moves you to the next field and awaits your keyin

If we want to have both the state abbreviation and the State number in the data record we would have to make changes both to the Green Earth form and to our program.

First, the form needs another field to hold the state code we're going to generate.

NAME: |_____

STREET: |_____

CITY: |_____

STATE: L CODE: |_ ←———————— *the new code will be automatically displayed in this field*

ZIP CODE: |____

TELEPHONE: (|__)|__-|___

CUSTOMER NO.:_____

PURCHASE AMOUNT: $_____

ADDING A FIELD TO HOLD A STATE CODE
Drawing No. 49

The new field should be assigned a code of 'P', Program Reserved, during the REQUIRE pass of form generation.

The "GOOD" part of our program (which is assigned to the State field) will look like this:

```
NXTFLD      FIELD+1
GOOD        CONVERT INPUT BY STATLIST AND STATENO GIVING NXTFLD
            STORE
```

Drawing No. 50

The FIELD instruction is now being used in conjunction with the CONVERT instruction to access a "relative" field instead of an absolute field. Here we want the field that is +1 from the current field. We no longer have to SHOW the results of the conversion since it will be displayed automatically in the Code field when the State field is entered. This is because the Code field is defined as Program Reserved, causing the contents to be displayed and entered automatically.

Notice that referencing NXTFLD didn't actually change the normal flow of data entry. The STORE was still dealing with the INPUT and OUTPUT for the State abbreviation field -- not the State Code field.

NAME: I----------------------- *this is field -2*
STREET: I----------------------- *this is field -1*
CITY: I---------------
STATE: I✓ CODE: I---- *this is the current field*
ZIP CODE: I---- *this is field +1*
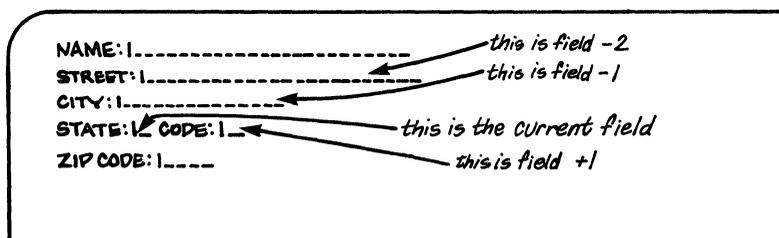
FIELDS CAN BE ACCESSED RELATIVE TO CURRENT LOCATION
Drawing No. 51

Range Checking

Data entry often involves seeing if numbers lie within a certain range. The following example checks to see if the entered date is from 1 through 31.

F*

YES   CHECK IF DATE IN 1-31   NO

MOVE DATA TO OUTPUT BUFFER

RETYPE FIELD

NEXT FIELD
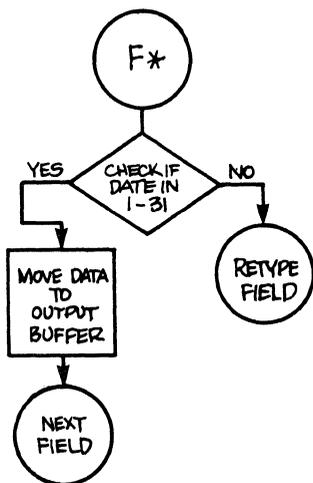
DAYTABLE   WORK "Ø1","31"
F*   IF INPUT NOTINRANGE DAYTABLE THEN AGAIN
            STORE

RANGE-CHECKING A DATE
Drawing No. 52

The IF-NOTINRANGE refers to the DAYTABLE label which, by use of a WORK statement, defines the acceptable range of the data. By placing quotes around the '01' and '31' and separating them by a comma, the WORK statement defines a range from 1 through 31. A similar program to check month range could be easily written:

MONTABLE   WORK "Ø1","12"
G*   IF INPUT NOTINRANGE MONTABLE THEN AGAIN
            STORE

Drawing No. 53

A data entry program might use three short programs like these to check the reasonableness of a date entered by the operator. A more complex version can be written to check for the correct number of days in months with 28, 30 or 31 days.

Any business computer worth its salt must be able to handle addition, subtraction, division and multiplication. A data entry operation might multiply price times quantity and then compare its result with what the operator keys in. Or it can do more extensive computation on numbers that have been keyed in, and take the burden off a central (home office) computer. In any case, arithmetic is easy when using the DATAFORM programming language.

As an example, let's take the contents of the current field and add it to the contents of the first field and display the total in the very last field. The following example is a form that does just that.

```
NBR RED PENCILS ORDERED:  |__
NBR BLUE PENCILS ORDERED:  |__
TOTAL PENCILS ORDERED:  |___
```

Drawing No. 54

To display the total, we'll use program Q to store the input from field 1 (red pencils) and program R to do the computation (adding input from fields 1 and 2), as follows:

```
TOTAL       WORK "0000"
Q*          ADD INPUT TO TOTAL
            STORE
R*          ADD INPUT TO TOTAL
            MOVE INPUT TO OUTPUT
            CHANGE +1
            SHOW TOTAL
            NEXT
```

SIMPLE COMPUTATION PROGRAM
Drawing No. 55

PART II
Generating a DATAFORM Level II Program

Several steps (naturally) are involved in writing a program and loading it onto your data entry system diskette. Here's a summary of those steps that are necessary to create a finished program:

1. Designing the needed program
2. Typing in the program
3. Compiling the program (converting it to computer readable form)
4. Combining form and program
5. Testing the complete data entry system

NOTE: You will need an additional diskette that contains the full version of DOS.C to create Level II DATAFORM programs. This diskette (model code number 20175) contains the Editor, the program that lets you create your programs.

Step 1. Designing the Program

For purposes of illustration, we will write a program that will operate on the Customer Number field of the Green Earth Garden Shoppe form. While this program could easily be made more specific, it will serve as a good illustration.

When the Green Earth Garden Shoppe customer form was generated, the Customer Number field required a seven digit, numeric, right justified zero-filled field. During the Program Assignment pass (PRO), this field did not have a program assigned. However, a program will now be assigned to examine the customer number and reject it if it is greater than 1,999,999. If it is accepted, the cursor will proceed to the next field. If it is rejected, an error message will be displayed on line 12 of the screen and the cursor will be repositioned at the beginning of the keyin field for you to try again.

Let's examine a DATAFORM Program that will do this for us.

```
BADNO    WORK "IMPROPER NUMBER"
MAXNO    WORK "1999999"
A*       IF INPUT GREATER MAXNO THEN BAD
         STORE
BAD      MESSAGE BADNO
         AGAIN
```

LEVEL II PROGRAM TO CHECK CUSTOMER NUMBER
Drawing No. 56

Step 2. Typing in the Program

You need two diskettes to help you begin the creation of your field program. Along with DF11SYS, the diskette you used for Level I DATAFORM work, you need a diskette that contains the full version of DOS.C (model code number 20175). The DF11SYS diskette contains a very limited version of the operating system (DOS.C), and we'll need the extensive facilities of the Editor, which is contained on the DOS.C diskette.

The Editor is a program that allows you to create and modify your program. You'll find just how easy it is to type in your program and make any corrections using this Editor.

Turn the Diskette 1100 system ON if it isn't already. Put the DF11SYS Diskette in Drive 0 and the DOS.C Diskette in Drive 1. Press RESTART.

Once the READY message appears, we're going to use the Editor program. To create the DATAFORM field program for our form, named SHOP01, type "EDIT SHOP01;D".

The ";D" tells the Editor that we want to use the preset tabbing facility. To make the job easier, the Editor program has preset tab stops for labels and instructions. Tab 1 is located at the left edge, where the labels are typed. Tab 2 is located about 9 spaces from the left edge, where instructions begin. If you tap the space bar while you're in the first tab area (columns 1-8), you'll automatically be skipped over to the ninth column, where you can begin typing your instructions (see the illustration below).

The drawing below shows what your program will look like when it is typed in. Be sure to use only upper case (CAPS) letters for the program instructions(messages and labels may be both upper and lower case). And remember to tap the ENTER key after each line is finished.

area for labels — area for instructions

the arrowhead

BADNO	WORK "IMPROPER NUMBER"
MAXNO	WORK "1999999"
A*	IF INPUT GREATER MAXNO THEN BAD
	STORE
BAD	MESSAGE BADNO
	AGAIN

Tab 1 (left edge)   Tab 2
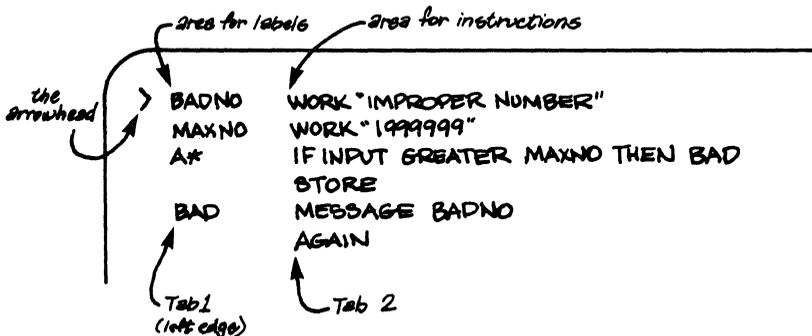
SCREEN PROGRAM TYPED IN UNDER GENERAL EDITOR
Drawing No. 57

What if you make a mistake? If you make it while typing, the BACKSPACE key will rub out the offending characters and the CANCEL key will obliterate the entire line.

However, let's suppose that you've finished typing in the program, and several errors mysteriously escaped your attention. This will give us a chance to show some of the power of the Editor program.

Notice the small arrowhead (>) in the left-most column. Press the KEYBOARD and DISPLAY keys one at a time and watch the arrowhead (line pointer) move up and down. By fiddling with these two keys, you can point out any line with an error.

Now, with the cursor in the left-most position at the bottom of the screen, you can type in the Editor commands to get at the error. All Editor commands are prefaced with a colon. Unless you begin each command with a colon, the Editor will assume it's just another line of the program. Remember that the arrowhead must be pointing (each time) to the line which is to be changed. Here are three of the most versatile Editor commands:

:DEL   (Delete) Blots out the entire indicated line and lets you try again.

:INS   (Insert) Opens up a space between two lines so that another line may be squeezed in.

:MOD   (Modify) This command lets you change individual characters or a group of characters. Suppose you accidentally typed DSPLAY and didn't notice it. Point the arrowhead at the line and type ":MOD DSPLAY<DISPLAY". The Less Than (<) Symbol is always placed between the old and new characters. This line will now contain the proper word.

DSPLAY<DISPLAY
↑   ↑   ↑
(OLD) | (NEW)
(REPLACES)

Other standard Editor commands may also be used (see Appendix F). The commands may be shortened to one letter, if desired, i.e. :MOD can be :M.

All this will be somewhat hazy until you have some experience at hammering away at the keyboard. One last step - when you've got everything so it looks good, stop and contemplate your handiwork. Then type the last and most important Editor command.

:END   (End of Program) Indicates to the Editor that the programmer is done writing text and to write a complete, perfect copy of his text (source program) to a diskette file by the name of SHOP01/TXT.

If you don't type ":END↲", and you skip to the next step, the processor will toss your program to the electronic winds, and you'll have to painstakingly type in the program again.

Step 3. Compiling the Program

After you have typed :END, the Compiler should be run. The Compiler will process the text you typed in under the Editor (EDIT), and create an object file (computer-readable language).

The Compiler searches for a printer and if it finds one connected to your Datapoint 1100 Diskette system, it asks if a listing (printout of the program) is to be produced.

If you answer "YES↲", the Compiler asks "LIST CODE TOO?" The Compiler is asking if you want the octal equivalent (machine translatable code that represents your DATAFORM program in the computer's memory) printed out in addition to the program. You probably will never need to see this code, so answer "NO↲".

Whether you request a printout or not, the Compiler displays an identifying message, "DISKETTE DATAFORM COMPILER 1.1".

Also, if you requested a printout, the Compiler asks for a heading for the printout. We'll use "SHOP01 PROGRAM" for our heading.

So, if you request a printout, your screen will look like this:

```
DOS.C DATAPOINT CORPORATION'S DISK OPERATING SYSTEM VERSION 1.1
READY
DF11CMP SHOP01
LIST ON PRINTER? YES
LIST CODE TOO? NO
DISKETTE DATAFORM COMPILER 1.1
HEADING: SHOP01 PROGRAM
```

Drawing No. 58

If you do not have a printer attached to your Diskette 1100 System, your screen should look like this:

```
DOS.C DATAPOINT CORPORATION'S DISK OPERATING SYSTEM VERSION 1.1
READY
DF11CMP SHOP01
DISKETTE DATAFORM COMPILER 1.1
```

Drawing No. 59

When the compiler is done with your program, the screen will look like this:

```
DISKETTE DATAFORM COMPILER 1.1
STORAGE USED IN DECIMAL: 42 RELOCATABLE 0 COMMON
FIELD PROGRAMS:
            A        00030
END OF COMPILATION: NO ERRORS
```

Drawing No. 60

Then the Operating System will display "READY".

A file has now been created by the name SHOP01/DFP. The printout of the program should look as follows:

```
PAGE  1      SHOP01/TXT    RECEIPTS

LINE  LABEL   OPERANDS
   1.
   2.  BADNO   WORK   "IMPROPER NUMBER"

   3.  MAXNO   WORK   "1999999"
   4.  A*      IF INPUT GT MAXNO THEN BAD
   5.          STORE
   6.  BAD     MESSAGE BDNO
   7.          AGAIN




STORAGE USED IN DECIMAL:    42 RELOCATABLE:    0 COMMON.

FIELD PROGRAMS:
              A          00030
END OF COMPILATION:  NO ERRORS.
```

If there are any errors, the offending statement will be flagged with an asterisk, and an error type shown. These messages will be inserted between the lines of the printout, but the screen display will only show the lines of code which are in error. The "END OF COMPILATION" message shows the total number of errors. To correct any errors, simply re-edit and recompile the program.

Step 4. Combining Form and Program.

To add the Field Program to the form, it is necessary to return to the Form Generation procedure. If you did assign the correct program letter during generation, these three steps are all that is required:

1. To access the DATAFORM generator, type "DF11GEN SHOP01)".

2. Since your form already exists, type "OLD)".

3. To include the compiled Field Program with the form and to return to the DOS program, type "OUT)".

Since we did not assign a program letter during form creation (Level I), we'll have to go back and do it now. Follow these easy steps:

1. To access the DATAFORM generator, type "DF11GEN SHOP01)".

2. Since the form already exists, type "OLD)".

3. To change the PROGRAM assignment, type "PRO)"

(Remember, don't do a REVISE unless you want to go through all the FORM generation passes again.)

4. Press the ENTER key to skip through each field until you are in the Customer Number field. Type an "A" and press DISPLAY/CANCEL.

5. Now that you're done, type "OUT⟩". NOTE: If these instructions are confusing, skim through Chapter 2, Level I again.

If your program was not out there, or was not yet compiled, the display would show a " PROGRAM MISSING " message. In any event, the next displays are "nnnn BYTES LEFT" (nnnn is some number), "DONE-LOAD NEXT SYSTEM", and then the Diskette Operating System message appears.

Step 5. Operating the Complete Data Entry System.

Data entry for Level II programs is similar to data entry for Level I programs (discussed in Part IV of Chapter II). The only difference is that your Field program will check certain data entries.

To enter data into our Green Earth Garden Shoppe form, we'll type "DF11 SHOP01;S⟩". The ";S⟩" tells the system that we want to start data entry and delete any file that may contain data for the SHOP01 form. This way we don't have to answer the question "SELECT DATA MODE".

Similarly, later we can add data to the end of our data file by typing "DF11 SHOP01;A⟩". Start and Add are the only two DATAFORM commands that can be added to the initial command in this way.

The form will appear on the screen. Enter data in the same manner as you did in Level I. Notice, however, that if you try to type a number larger than 1,999,999 in the Customer Number Field, your program will display the message "IMPROPER NUMBER" and reject it by beeping and returning the cursor to the beginning of the field.

CUSTOMER NO.: 2000000   ← this number will be
PURCHASE AMOUNT: rejected
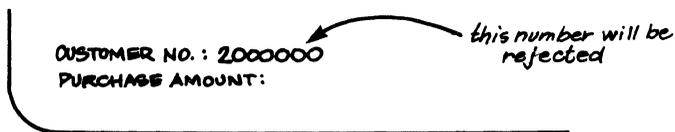
THE PROGRAM REJECTS WRONG NUMBERS
Drawing No. 61

After testing, give this form to a data entry operator to begin the data entry. You now have a completely functioning data entry system with expanded (program) editing.

When you (or the data entry operator) are done with the data entry, ";END⟩" the last entry and type "OUT⟩" to get out the DATAFORM data entry program. Remove the diskettes and turn off the terminal and you're free to do something else.

PART III
Summary of the Language Components

The preceding sections showed just how useful field programs can be. To offer ourselves wider horizons, however, a thorough discussion of the language is necessary. This section outlines the many capabilities of the language.

Fields and Labels

First let's take a closer look at fields and labels. A field is any physical location in either the program or the form where data is stored. Each field has an associated length. A label identifies a location in memory, i.e. the location of a field or instruction. Finally, to throw you a bit of a curve, four pre-defined labels represent fields. They are:

INPUT   This addresses the field which contains the data just keyed in.

OUTPUT   This addresses the field which contains the data which is waiting to be written to the data file.

NULL   This defines a null or no-data field.

RETRY   This field contains a flag.

NOTE: A flag is usually a one character "clue" which is checked during the program to see if a certain thing has or has not occured yet.

Fields can also be defined by additional specification statements as follows:

label   DATA n,m   or STUFF1 DATA 1,20

Data statements point to the data record (output buffer) where n and m refer to the first and last column of the data record to be referenced by the label (STUFF1).

WORK statements pre-define characters:

label   WORK "PREDEFINED CHARACTERS"

WORK is also used in defining working storage for octal constants (for more advanced applications). Finally, the WORK statement can be used to reserve space:

label   WORK 5

where 5 indicates the size of the non pre-defined working storage for temporary storage of data; i.e., A place in- memory will be reserved for five characters.

The COMMON statement can be used the same as WORK statement. Also, non pre-defined COMMON statements can be used to pass data between forms.

label   COMMON "CONSTANTS" or label   COMMON 5

The REDEFINE statement defines a field which is a subfield of another defined field.

label2   REDEFINE label1, n, m

Now, label2 defines a field that starts with column n of field label1 and is m columns in length.

The FIELD Statement can define either the absolute or relative field in the form output.

label   FIELD n

This defines the nth field in the form.

label   FIELD /sign/ m

(sign is a + or -)

This defines the field number relative to the present field, i.e. m fields (+ or -) away from the field that you're presently working in.

Tables

A table is a number of fields, grouped successively, with a label pointing to the first field in the table.

label   WORK "TB1", "TB2", "TB3"

This will define a table with three entries, TB1, TB2, and TB3. Each entry has a length of three and the table has a length of nine plus the END OF TABLE Byte (the space taken up by one character or 8 computer-readable bits) for a total of 10 characters (Bytes).

Exit Paths

AGAIN	Re-requests the current field, after an error is detected
CLOSE	Passes control back to the DATAFORM Interpreter after an End Of File Mark (:END) is written. That is, an End of File mark is written on the data file and the message "READY" is displayed, leaving the operator in Monitor mode

END	Passes control back to the DATAFORM Interpreter (for subsequent LINK, NEXT, etc.) after writing OUTPUT to diskette
NEXT	The current field number is incremented and the cursor is moved to the next sequential field (no data is stored)
STORE	Moves INPUT to OUTPUT and positions to the next field in the form. In other words, it's the same as MOVE INPUT TO OUTPUT and NEXT
CHAIN	Load the specified form as if the operator had typed "LOAD n" and start execution at the first non-Constant field of the new form. In the following example, form 5 would be loaded:

```
LABEL       WORK       "05"
A*          CHAIN      LABEL
```

Executable Statements

Executable Statements are instructions to the DATAFORM Interpreter to "Do Something". Most of the Executable Statements work with fields and with tables to perform advanced editing and data manipulation based on programmed logic.

Executable Statements fall into six groups:

1. Instructions which transfer information (MOVE, CONVERT, etc.)

2. Instructions that perform arithmetic (ADD, SUB, MUL, DIV)

3. Instructions which transfer control (GOTO , CALL, AGAIN, STORE, etc.)

4. Instructions which compare then transfer control (IF condition THEN)

5. Instructions which provide output control, i.e. write on the screen or the diskette (SHOW, WRITE, etc.)

6. Instructions which affect the field index (CHANGE, RESET)

STATEMENTS THAT TRANSFER INFORMATION

ALIGN   This statement transfers numerical data from one field to another field, aligning the decimal point in the first field with the decimal point on the second field. If no decimal point exists, it is assumed to be at the right-most edge of the destination field. Truncation occurs if the receiving field is smaller.

EXAMPLE:
```
    FLD1   WORK "123.74"
    FLD2   WORK "0000.0"
           ALIGN FLD1 TO FLD2
FLD2 will become "0123.7"
```

CONVERT   This statement looks for an entry in Table # 1 equal to Field #1 and if found, moves the corresponding entry from Table #2 to Field #2.

EXAMPLE:
```
    TBL1   WORK "01","02","03","04"
    TBL2   WORK "JAN","FEB","MAR","APR"
    FLD1   WORK "03"
    FLD2   WORK "XXX"
           CONVERT FLD1 BY TBL1 AND TBL2 GIVING FLD2
FLD2 will now be "MAR"
```

LOOKUP   This statement uses one field as a numeric (numbers should be used to be certain of results) index to a table, and moves the indexed table entry to a second field.

EXAMPLE:
```
    FLD1   WORK "02"
    TBL1   WORK "MON","TUE","WED"
    FLD2   WORK "XXX"
           LOOKUP FLD1 IN TBL1 GIVING FLD2
FLD2 will now be "TUE"
```

MOVE   This statement literally "moves" one field to another field location

EXAMPLE:
```
    FLD1   WORK "ABCD"
    FLD2   WORK "WXY"
    FLD3   WORK "123456"
           MOVE FLD1 TO FLD2
```

FLD2 will now be "ABC"

            MOVE FLD1 TO FLD3
FLD3 will be "ABCD56"

Characters are left justified with truncation at the right, if the receiving field is smaller, and no filler if the receiving field is larger.

SET   This statement causes the first character
                of one field to be moved to all positions
                of a second field

                EXAMPLE:
                    FLD1   WORK "0"
                    FLD2   WORK "123456789"
                            SET FLD2 TO FLD1
                When done, FLD2 will be "000000000"

STATEMENTS TO PERFORM ARITHMETIC OPERATIONS

ADD   This statement adds two numeric fields
                and puts the result either in the second
                field or in a designated third field.

                EXAMPLE:
                    FLD1   WORK "123.4"
                    FLD2   WORK "101.2"
                    FLD3   WORK "000.0"
                            ADD FLD1 TO FLD2
                This will make FLD2 equal to "224.6"
                            ADD FLD1 TO FLD2 GIVING FLD3
                FLD2 will remain "101.2" and FLD3 will be "224.6"

SUBTRACT   This statement subtracts one numeric field
or SUB   from a second field, and puts the result in
                either the second field or a third field.

                EXAMPLE:
                    FLD1   WORK "124.4"
                    FLD2   WORK "243.6"
                    FLD3   WORK "000.0"
                            SUB FLD1 FROM FLD2
                FLD2 will now be "119.2"
                            SUB FLD1 FROM FLD2 GIVING FLD3
                FLD2 will be unchanged and FLD3 will be "119.2"

MULTIPLY   This statement multiplies one field by
or MPY   another field, placing the result either
or MUL   in the second field or in a third field.

EXAMPLE:
```
     FLD1   WORK "10.4"
     FLD2   WORK "04.0"
     FLD3   WORK "00.0"
            MPY FLD1 BY FLD2
```
FLD2 will now equal "41.6"
```
            MPY FLD1 BY FLD2 GIVING FLD3
```
FLD2 will remain unchanged and FLD3 will be "41.6"

DIVIDE
or DIV

This statement causes one numeric field to be divided into a second numeric field, with the result going into the second field or into a third designated field.

EXAMPLE:
```
     FLD1   WORK "04.0"
     FLD2   WORK "16.4"
     FLD3   WORK "00.0"
            DIV FLD1 INTO FLD2
```
FLD2 will now equal "04.1"
```
            DIV FLD1 INTO FLD2 GIVING FLD3
```
FLD2 will be unchanged and FLD3 will be "04.1"

STATEMENTS THAT PERFORM COMPARISONS

The general formats for comparison statements are:

```
     label   IF field1 RELATION field2 THEN label1
     label   IF field1 RELATION table1 THEN label1
```

If the relation is true, the program is directed to the statement designated by label1 or to the exit path designated by pre-designated table1. Relations are defined in the following sections.

ASCII Comparisons

Fields can be compared on a character basis from left to right. The length of field1 determines the length of the comparison. If field1 is longer than field2, the results are indeterminate. The relation can have the following values in ASCII Comparison.

EQUAL or EQU or EQ
GE or GEQ (Greater or Equal)
GREATER or GT or GTR
LE or LEQ (Less than or Equal)
LESSTHAN or LESS or LT
NOTEQUAL or NE or NEQ

EXAMPLE:
```
FLD1    WORK "0123"
FLD2    WORK "0122"
        IF FLD1 EQ FLD2 THEN LBL1
        NEXT
LBL1    something
```

Since they are not equal, processing will fall through to the NEXT statement.

Table Lookup

A table can be checked to see if a matching field is in the table. The relation can have the following values in Table Lookup:

INRANGE or INR
NOTINRANGE or NIR
INTABLE or INT
NOTINTABLE or NIT

EXAMPLE:
```
TBL1    WORK "1","2","3","4"
FLD1    WORK "2"
        IF FLD1 INT TBL1 THEN LBL1
        NEXT
LBL1    something
```

Since FLD1 is in the table, control will be passed to the statement at LBL1., rather than "NEXT". The length of FLD1 is used in the Lookup.

Check Digit Verification

Numeric fields that contain a check digit can be verified with CK10 (Mod 10 Check) or CK11 (Mod 11 Check). The format is:

```
label   IF field1 CK10 field2 THEN label1
```

Field1 contains the numeric field with the check digit in the least significant position. Field2 contains the weighing factor and is assumed to be one character shorter than Field1.

EXAMPLE:
```
FLD1    WORK "864213574"
FLD2    WORK "12121212"
        IF FLD1 CK10 FLD2 THEN LBL1
        AGAIN
LBL1    SOMETHING
```

Since the check is TRUE, control will pass to LBL1.

STATEMENTS THAT PROVIDE FOR OUTPUT CONTROL

There are several executable statements that give the field program control over data output to the display screen or to the data diskette.

Write to Data File
The WRITE statement will write the form output buffer to the data file on the diskette. Control is returned to the next program statement. The data area is not cleared. The format is:

label   WRITE

Display A Message On the Screen
The MESSAGE statement will write the specified message on the bottom line of the screen (line 12). It causes the bottom line of the form (if the form has anything on line 12) to be erased, the message to be displayed and the INPUT field to be destroyed.

EXAMPLE:
MSG   WORK "CHECK ERROR"
MESSAGE MSG

Display a Message in the Current Field
The SHOW statement can display either a designated message or the current field value in the current field area on the screen. If the contents of the input buffer is to be saved, it should be done before the SHOW. SHOW destroys the current field portion of the input buffer! The format for displaying a message is:

label1 SHOW   label2

EXAMPLE:
MSG   WORK "BAD INPUT"
SHOW MSG

The format for displaying the current value of the data record is:

label   SHOW

EXAMPLE:
Q*   MOVE INPUT TO OUTPUT
SHOW

Issue the BEEP Sound
The BEEP statement causes the machine to BEEP.

EXAMPLE: BEEP

Remember that the exit "AGAIN" does an automatic "BEEP".

Load a New Form

The CHAIN statement causes a different form to be loaded. The current data record is not written. The specified form is loaded and control is passed to the Interpreter at the first field of the new form.

FORMAT: label2   CHAIN Label1

EXAMPLE: LABEL1   WORK "05"

                            CHAIN LABEL1

Display the Current Form

The FORMSHOW statement will cause the current form to be redisplayed on the screen with all data fields cleared. The output record is not cleared. The INPUT buffer is destroyed.

FORMAT: label   FORMSHOW

Remember, this is not an exit.

STATEMENTS THAT CAUSE TRANSFER OF CONTROL

Unconditional Transfer

The GOTO statement causes an unconditional transfer of control.

FORMAT: label   GOTO label1

EXAMPLE: GOTO LBL1

Subroutines

A subroutine is a group of commonly used executable statements that can be called up for use again and again. The subroutine is called up by the CALL statement. The RETURN statement, within the subroutine, causes control to be transferred back to the main program to the statement following the CALL statement.

FORMAT: label1   CALL label2

                 label2   executable statement
                               executable statement
Subroutine   executable statement
                               executable statement
                               RETURN

EXAMPLE: This subroutine displays the error message
                    "BAD INPUT" and returns to the statement
                    following the CALL

```
MSG        WORK "BAD INPUT"
           CALL BIMSG


BIMSG      MESSAGE MSG
           RETURN
```

STATEMENTS THAT AFFECT FIELD POINTER

Move INPUT Field Pointer

When a field program is called after the operator completes the field entry, there exists a pointer to the current field number which is used by certain executable statements and exit paths. This pointer can be changed by the CHANGE statement and reset to its original value by the RESET statement.

The CHANGE statement can refer to the field number in either a relative or an absolute manner.

An example of relative field pointer positioning is:

CHANGE + 1
or CHANGE - 3

where the number defines the relative pointer movement from your present field.

The absolute positioning is shown by the example:

CHANGE 6

where the pointer is directed toward the sixth field in the form.

The pointer can be returned to its original position with the RESET command. In the following example, notice how the pointer position is CHANGE and RESET.

RESET

```
ONE: I__
TWO: I__
THREE: I__  ◄──────  the program is assigned
FOUR: I__                  to this field
```
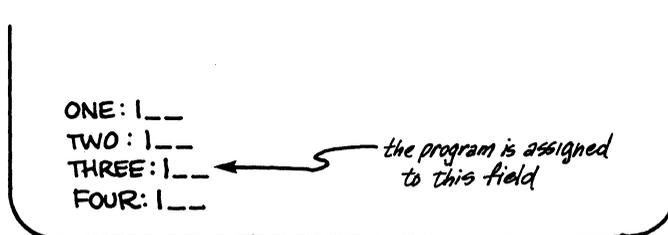
```
T*   CHANGE  -I  ◄──────────── now the pointer is to field "TWO"
     CHANGE   I  ◄──────────── now to field "ONE"
     CHANGE  +I  ◄──────────── now to field "TWO"
     CHANGE  +2  ◄──────────── now to field "FOUR"
     RESET       ◄──────────── now back to the original field "THREE"
     STORE
```

USING CHANGE TO MOVE FIELD POINTER
Drawing No. 62

A NOTE ON NUMERIC FIELDS

Numeric fields must contain only numeric digits (0 to 9), an optional decimal point, and an optional minus sign. There can be no more than one decimal point and if a minus sign is present, it must be in the leftmost significant position.

Also, there can be no more than sixteen significant digits to the left of the decimal point and no more than eight to the right of the decimal point.

A NOTE ON CHECK DIGITS

A check digit is a single digit which is appended to a number that is generally used for identification purposes (such as an account number, credit card number, etc.) in order to validate that number. The check digit is generated through the use of a mathematic algorithm (procedure) using a *fixed* weighing factor. When the identification number which already includes the check digit is entered into a data entry system, it can be checked for a correct check digit and rejected if not correct. This can greatly reduce errors generated by incorrectly entering the identification number.

Mod 10 check digits are generated and checked as follows:

```
Keyin ID          2345676
ID Number         234567
Weight factor     121212
```

```
Sum of Digits of Product ──────────────────────────────┐
Products ──────────────────────────┐                    │
Effective Weights ────────────┐     │                    │
Basic ID ─────────────┐       │     │                    │
                      ▼       ▼     ▼                    ▼
                      2 × 1 = Ø2 ──────────▶  Ø + 2 =  2
                      3 × 2 = Ø6 ──────────▶  Ø + 6 =  6
                      4 × 1 = Ø4 ──────────▶  Ø + 4 =  4
                      5 × 2 = 1Ø ──────────▶  1 + Ø =  1
                      6 × 1 = Ø6 ──────────▶  Ø + 6 =  6
                      7 × 2 = 14 ──────────▶  1 + 4 =  5
                                                 TOTAL = 24
```
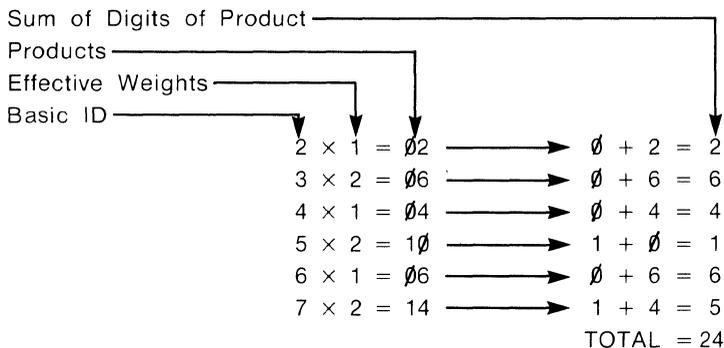
To determine the Mod 10 check digit, subtract the total (24) from the next highest multiple of the base 10 (30) to yield 6.   The total ID number will now be 2345676.

Mod 11 check digits are generated and checked as follows:

```
Keyin ID        2345676
ID Number       234567
Weight Factor   765432
```

```
Products ─────────────────────┐
Effective Weights ─────────┐   │
Basic ID ──────────┐       │   │
                   ▼       ▼   ▼
                   2   ×   7 = 14
                   3   ×   6 = 18
                   4   ×   5 = 2Ø
                   5   ×   4 = 2Ø
                   6   ×   3 = 18
                   7   ×   2 = 14
                   TOTAL   = 1Ø4
```
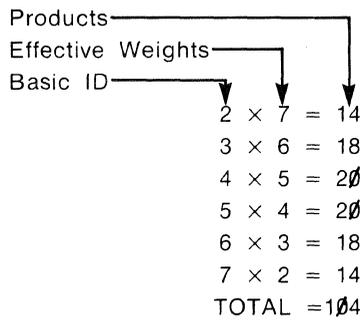
To determine the Mod 11 check digit, subtract the total (104) from the next highest multiple of 11 (110) to yield 6. The total ID number will now be 2345676.

If a check digit of 10 is yielded, it will be made zero, therefore only 2345670 will compare favorably.

PART IV
Program Examples

Dealer File

The purpose of this form is to create a dealer file. Each record in the file will contain the date of the entry, the dealer's name and address and the shipping address. Since several entries will be made on the same date, the date field should only have to be entered once. Also, the operator should be able to indicate that the shipping address is the same as the mailing address.

Form Generation

The City and State Fields are Semi-Constant fields and are entered in the form as underline (__) fields. During the SEM Pass, the city and state are entered. Note that when the city is entered, the field is space-filled with carets. If this was not done, the field would be improperly filled on the data file.

Program Generation

There are three field programs assigned to this form: A*, B*, and C*.

Program A* is assigned to the Date field. The program forces the operator to enter the date the first time the program is run after loading. After the date field has been initialized, a null input (ENTER key only) allows the stored date to be used.

Program B* is assigned to the "Ship To" keyin field. If the keyin is an "S", the Address, City, State, and Zip fields are moved from the preceding fields and the form record is written to diskette.

Program C* is assigned to the Ship To Zip field. It will be called at the end of the form if an "S" was not entered in the "Ship To" field. It causes the complete form record to be written to diskette.

```
                       LARGELOT FURNITURE COMPANY
                          San Antonio, Texas

DATE |
DEALER |
ADDRESS |
CITY SAN ANTONIO                       STATE TX   ZIP |

           SHIP TO <
ADDRESS |
CITY SAN ANTONIO                       STATE TX   ZIP |
```

```
                              LARGELOT FURNITURE COMPANY
                                 San Antonio, Texas

                 DATE |
SIZE                6
TYPE                D
REQUIRED            F
PROGRAM             A

                 DEALER |
SIZE                34
TYPE                M
REQUIRED            R
PROGRAM
```

```
                ADDRESS |
SIZE                     33
TYPE                     M
REQUIRED                 R
PROGRAM

                CITY SAN ANTONIO                    STATE TX   ZIP |
SIZE                     27                              2          5
TYPE                     A                               A          D
REQUIRED                 R                               B          B
PROGRAM

                    SHIP TO <
SIZE                         1
TYPE                         A
REQUIRED                     B
PROGRAM                      B

                ADDRESS |
SIZE                     33
TYPE                     M
REQUIRED                 R
PROGRAM

                CITY SAN ANTONIO                    STATE TX   ZIP |
SIZE                     27                              2          5
TYPE                     A                               A          D
REQUIRED                 R                               B          B
PROGRAM                                                             C
```

```
LABEL   OPERANDS
DATE        WORK "       "
SP          WORK "       "

A*          IF    NULL NE INPUT THEN GO
            IF    SP EQ DATE THEN AGAIN
GOT         MOVE  DATE TO OUTPUT
            SHOW
            NEXT
GO          MOVE  INPUT TO DATE
            GOTO  GOT
.
.
.
SS          WORK  "S"
ADDR1       FIELD 3
CITY1       FIELD 4
STATE1      FIELD 5
ZIP1        FIELD 6
ADDR2       FIELD 8
CITY2       FIELD 9
STATE2      FIELD 10
ZIP2        FIELD 11

B*          IF    INPUT NE SS THEN NEXT
            MOVE  ADDR1 TO ADDR2
            CALL  CHG
            MOVE  CITY1 TO CITY2
            CALL  CHG
            MOVE  STATE1 TO STATE2
            CALL  CHG
            MOVE  ZIP1 TO ZIP2
            CALL  CHG
            END
```

76

```
CHG          CHANGE+1
             SHOW
             RETURN
  .
  .
  .
C*           MOVE  INPUT TO OUTPUT
             END
```

Validate Date

This program checks the date input for probable errors. It verifies that the month is in the range of 01 through 12, that the date is within the maximum for the month, and that the year is 75.

```
             ENTER DATE:  | /| /|
SIZE                      2  2  2
TYPE                      D  D  D
REQUIRED                  B  B  B
PROGRAM                   H  I  J
```

```
LABEL   OPERANDS
MNTH         WORK  "01","12"
DYINM        WORK  "31","28","31","30","31","30":

                   "31","31","30","31","30","31"

MXDY         WORK  "00"
YR           WORK  "74"
  .
  .
H*           IF    INPUT NIR MNTH THEN AGAIN
             LOOKUP INPUT IN DYINM GIVING MXDY

             STORE
  .
  .
I*           IF    INPUT GT MXDY THEN AGAIN
             STORE
  .
  .
J*           IF    INPUT NE YR THEN AGAIN
             STORE
```

Order Entry

This is a two-part form in which the heading information is entered on the first form and the item information is entered on the second form. The second form is written for each line number.

For each Heading Record there can be any number of Item Records. The Heading Form will be Form 6 and the Item Form, Form 7.

Form 6 -- The first three fields are for the date (month, day, year) and are just keyin fields, i.e., they do not appear in the Output Record. They call programs A*, B*, and C* which create the date code in Field 5. The format of the five digit date code is (year)/(day in year). Checks are made to insure that the date is reasonable.

The fourth field is a Constant field, which is the first Output Field. This identified the Heading Record in the data.

Fields 6 through 10 are for the Name and Address.

Field 11 is the tax rate, and calls program D*. This program checks to insure that the tax rate is in an appropriate range and stores it in a COMMON location for further use by Form 7.

The operator can go back at any time if his form entry is found to be incorrect. Then the operator presses DISPLAY and Period, and the form is written to the diskette and Form 7 is called.

```
DATE: <_/<_/<_           FORM ID: 006          DATE CODE: |____
NAME: |_____
ADDRESS: |_____
CITY: |_____  STATE: |_   ZIP: |____
              TAX RATE: 0.0 %
```

WHEN ALL FIELDS HAVE BEEN ENTERED TO YOUR SATISFACTION, PLEASE PRESS THE DISPLAY

AND PERIOD KEYS SIMULTANEOUSLY IN ORDER TO WRITE THE DATA TO THE FILE AND LOAD

THE NEXT FORM FOR ENTRY OF THE LINE ITEMS........

FORM: 06, LINK: -07, DATA RECORD: 120 BYTES, FREE SPACE: 4279 BYTES

```
          DATE: <_/<_/<_          FORM ID: 006          DATE CODE: |____
SIZE            2  2  2                3                       5
TYPE            D  D  D                D                       D
JUSTIFY
REQUIRED        B  B  B                B                       S
PROGRAM         A  B  C

          NAME: |_____
SIZE            35
TYPE            M
JUSTIFY
REQUIRED        R
PROGRAM

          ADDRESS: |_____
SIZE            32
TYPE            M
JUSTIFY
REQUIRED        R
PROGRAM

          CITY: |_____  STATE: |_   ZIP: |____
SIZE            35                                2           5
TYPE            A                                 A           D
JUSTIFY
REQUIRED        R                                 B           B
PROGRAM

                        TAX RATE: 0.0 %
SIZE                          3
TYPE                          N
JUSTIFY
REQUIRED                      R
PROGRAM                       D
```

WHEN ALL FIELDS HAVE BEEN ENTERED TO YOUR SATISFACTION, PLEASE PRESS THE DISPLAY

AND PERIOD KEYS SIMULTANEOUSLY IN ORDER TO WRITE THE DATA TO THE FILE AND LOAD
THE NEXT FORM FOR ENTRY OF THE LINE ITEMS........

```
LABEL   OPERANDS
.
. CONSTANT, VARIABLE & TABLE DATA
.
TAX      COMMON "0.0"
HOLDMO   WORK "00"
MORNG    WORK "01","12"
TOTDAY   WORK "000","031","059","090":
              "120","151","181","212":
              "243","273","304","334","365"
```

```
YR        DATA 4,5
DAY       DATA 6,8
DAMO      WORK "000"
ONE       WORK "01"
XHLD      WORK "000"
YRRNG     WORK "75","76"
TXRNG     WORK "0.0","6.0"
CKMO      WORK "00"
.
.
. PROGRAM A - INPUT & CHECK MONTH
.
A*        IF INPUT NIR MORNG THEN AGAIN
          MOVE INPUT TO HOLDMO
          NEXT
.
.
. PROGRAM B - INPUT DAY & CALCULATE JULIAN DATE
.
B*        IF INPUT LT ONE THEN AGAIN
          LOOKUP HOLDMO IN TOTDAY GIVING DAMO
          ADD INPUT TO DAMO
          ADD ONE TO HOLDMO GIVING CKMO
          LOOKUP CKMO IN TOTDAY GIVING XHLD
          IF DAMO GT XHLD THEN AGAIN
          MOVE DAMO TO DAY
          NEXT
.
.
. PROGRAM C - INPUT & CHECK YEAR
.
C*        IF INPUT NIR YRRNG THEN AGAIN
          MOVE INPUT TO YR
          CHANGE +2
          SHOW YR
          NEXT
.
.
. PROGRAM D - INPUT & CHECK TAX BASE
.
D*        ALIGN INPUT TO TAX
          IF TAX NIR TXRNG THEN AGAIN
          MOVE TAX TO OUTPUT
          SHOW TAX
          WRITE
          NEXT
```

Part Number Entry Program

Form 7 -- Part number entry program. Field 1 is a Constant Identification field. Fields 2 through 4 allow entry at part number, quantity and cost. A Null input at Field 2 Chains back to Form 6.

Field 4 calls program M which calculates the total tax and total amount for the line item.

The operator can go back at any time to change the form if an error is found. DISPLAY/Period directs writing to the diskette and brings up subsequent Line Item Forms.

```
                    FORM ID: 007


PART NUMBER      QTY       AMOUNT          TAX           TOTAL
|_____         |__      $|_____        $|_____    $|_____
                           ENTER DECIMAL, TOO
```

WHEN ALL FIELDS HAVE BEEN ENTERED TO YOUR SATISFACTION, PLEASE PRESS THE DISPLAY

AND PERIOD KEYS SIMULTANEOUSLY IN ORDER TO WRITE THE DATA TO THE FILE AND LOAD
ANOTHER (EMPTY) LINE ITEM FORM. THEN YOU MAY EITHER PRESS THE ENTER KEY WHILE
IN THE PART NUMBER FIELD, IF YOU HAVE NO MORE LINE ITEMS THIS ORDER, OR ENTER
ANOTHER LINE ITEM. PRESSING ENTER WILL GIVE YOU THE ORDER HEADER FORM AGAIN....

FORM: 07, LINK: 06, DATA RECORD: 043 BYTES, FREE SPACE: 4364 BYTES

FORM ID: 007

SIZE	3
TYPE	D
JUSTIFY	
REQUIRED	B
PROGRAM	

	PART NUMBER	QTY	AMOUNT	TAX	TOTAL
	|_____	|__	$|_____	$|_____	$|_____
SIZE	8	3	7	11	11
TYPE	D	D	N	N	N
JUSTIFY	J	J			
REQUIRED	F	R	R	S	S
PROGRAM	L		M		

ENTER DECIMAL, TOO
WHEN ALL FIELDS HAVE BEEN ENTERED TO YOUR SATISFACTION, PLEASE PRESS THE DISPLAY

AND PERIOD KEYS SIMULTANEOUSLY IN ORDER TO WRITE THE DATA TO THE FILE AND LOAD
ANOTHER (EMPTY) LINE ITEM FORM. THEN YOU MAY EITHER PRESS THE ENTER KEY WHILE
IN THE PART NUMBER FIELD, IF YOU HAVE NO MORE LINE ITEMS THIS ORDER, OR ENTER
ANOTHER LINE ITEM. PRESSING ENTER WILL GIVE YOU THE ORDER HEADER FORM AGAIN....

```
LABEL   OPERANDS
.
. CONSTANT, VARIABLE & TABLE DATA
.
TAX       COMMON 4
TOTAL     DATA 33,43
TTAX      DATA 22,32
C         WORK "100"
WKAMT     WORK "0000.00"
QTY       DATA 12,14
WTX       WORK "0.000"
SIX       WORK "6"
.
. PROGRAM L - IF INPUT NULL, CHAIN TO FORM 6
.
L*        IF NULL EQ INPUT THEN GO
          STORE
.
GO        CHAIN SIX
.
. PROGRAM M - CALCULATE TOTAL PRICE, INCLUDING TAX
.
M*        ALIGN INPUT TO WKAMT
          MOVE WKAMT TO OUTPUT
          SHOW
          DIV C INTO TAX GIVING WTX
          MPY QTY BY WKAMT
          MPY WTX BY WKAMT GIVING TTAX
          ADD WKAMT TO TTAX GIVING TOTAL
          CHANGE +1
          SHOW
          CHANGE +1
          SHOW
          NEXT
```

```
00675142HORATIO HORNBLOWER                113 CAPTAIN STREET             SCHOO
NER                          NY120516.0

006751 2HORATIO HORNBLOWER                113 CAPTAIN STREET             SCHOO
NER                          NY120516.0

00711111111  20040.000000000000400000000084

00722222222  40020.000000000000400000000084

00733333333 100100.000000000006000000001060

00675142JOHNNY APPLESEED                  110 ORCHARD STREET             APPLE
TREE                         IL618205.0
```

```
00675ι42JOHNNY APPLESEED              110 ORCHARD STREET              APPLE
TREE                    IL618205.0

00744444444  21000.00000000001000000002100

00755555555  80500.00000000002000000004200

0Ω675142MARK TWAIN                    18 STERNWHEELER LANE           RIVER
  CITY                  MS492014.5

00675142MARK TWAIN                    18 STERNWHEELER LANE           RIVER
  CITY                  MS492014.5

Ω0766666666 500250.0000000000112000000002612

0Ω675142DON DIEGO                     1 SAN PEDRO ROAD               BAHA
                        CA951295.5

00675142DON DIEGO                     1 SAN PEDRO ROAD               BAHA
                        CA951295.5

00777777777  24500.0000000000049500000009495

00788888888 100042.00000000002300000000443
```

PART V

The Print Utility

You can use the Print Utility program to print your data files or your DATAFORM forms. To print the SHOP01 form, type "DF11PRT SHOP01)" To print the SHOP01 data, type "DF11PRT SHOP01;D)". To print both the SHOP01 form and data, type "DF11PRT SHOP01;A)".

For more details about the Print Utility, see the DISKETTE DATAFORM User's Guide.

CHAPTER FOUR

CASSETTE DATAFORM

PART I
Overview

Introduction

The previous chapters discussed the design, generation, and testing of a form, as well as the writing, compiling, and use of an accompanying program. Although the cassette is a different media altogether, the concepts of form creation, form generation, programming and data entry are very similar in the manner in which they apply to all three types of DATAFORM.

If, after reading this chapter, more specific information concerning CASSETTE DATAFORM is needed, it can be found in the CASSETTE DATAFORM User's Guide. This chapter assumes a basic knowledge of DATAFORM concepts from the preceding chapters and familiarity with Datapoint equipment.

Equipment Needed:

1. Any Datapoint Processor with cassette capability (1100, 2200, 5500) and at least 8K of memory. Forms and programs generated on one Datapoint Processor (say, the 1100) can be used interchangeably with any other Datapoint Processor (say, the 2200 or 5500).

2. At least five cassette tapes, three of which will be program tapes and two blank or "scratch" tapes; see Programs Required section, below, for a description of the program tapes.

3. A printer is very useful for printing out data files, forms, programs, and doing a multitude of tasks. Any of the printers which are presently offered, such as the Servo, Belt, Matrix, 300 or 600 LPM Printers, can be used for these duties.

Programs Required:

NOTE: Any programming system will undergo a series of improvements and changes during usage, so the user should request the latest version of the cassette programs. Since this book covers only the fundamental elements of DATAFORM, no problems should be encountered using a later version of the program.

The complete CASSETTE DATAFORM programming system resides on three cassettes. Note that these three cassettes are only used during actual preparation, and that the final system used by the operator for data entry operation is on one cassette. The cassettes consists of:

1. DF2FGS   DATAFORM Form Generator (Form Generator and some additional functions). Model Code 20078.

2. DF2SYS   DATAFORM Systems (Interpreter, Configurator and Utilities -- final form and program catalog). Model Code 20080.

3. DF2PGS   DATAFORM Program Generator (Program Editor, Compiler and Print Utility). Model Code 20079.

Additionally, other system utilities, such as DATAPOLL, the automatic communications polling and answering program, are available in cassette form.

Drawing No. 63, on the next page, represents all the phases in the development of an operational Level I CASSETTE DATAFORM data entry system. The ensuing parts of this chapter represent abbreviated step-by-step guides for Levels I and II, expanded to include any factors peculiar to cassette operation.

STEPS:

① DEFINE *the* SCREEN FORMAT.
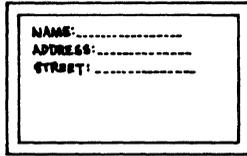
② GENERATE THE FORM WITH DF2FGS CASSETTE AND CREATE FORM CASSETTE.
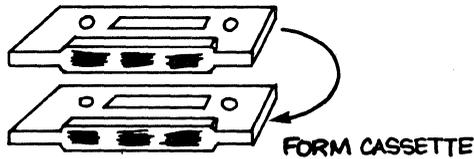
FORM CASSETTE

③ LOAD FINISHED FORM INTO DF2SYS CASSETTE

DF2SYS
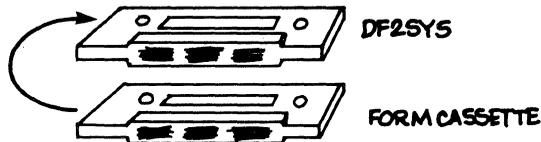
FORM CASSETTE

④ ENTER DATA TO CASSETTE USING FINISHED SCREEN FORM

YOUR FORM

DF2SYS WITH FORMS
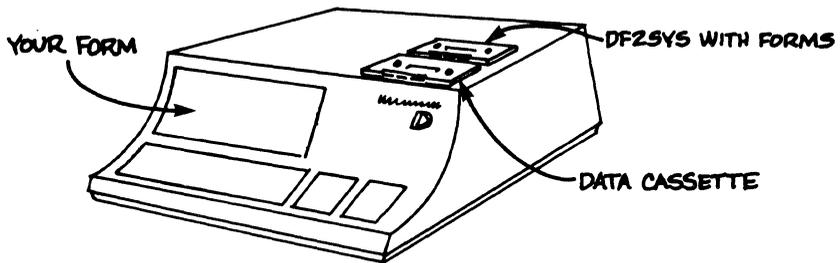
DATA CASSETTE

CASSETTE DATAFORM LEVEL I
SYSTEM DEVELOPMENT PHASES
Drawing No. 63

PART II
Level I Step-by-Step Guide

Before we begin creating a Level I data entry system, a word about cassettes. Do not place your cassettes on top of the Datapoint Processor while it is in operation; and don't place cassettes near any magnetic substance. Stray magnetic fields can destroy the data on the tape. Keep tapes in a relatively dustfree environment.

Here's an operating tip -- when mounting a cassette, make sure that the clear leader is not exposed. Use a pencil to wind the sprockets around such that only brown tape is showing, as in the following illustration.

make sure the tape appearing is all brown not clear
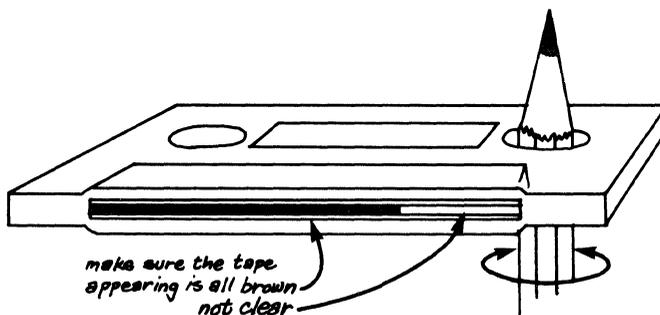
A CASSETTE OPERATION HINT
Drawing No. 64

This assures us that the tape deck won't be confused in thinking it's reached the end of a cassette, and refuse to rewind it in the correct direction.

Now, let's create a working Level I CASSETTE DATAFORM data entry system.

STEP 1. Load the Form Generator Tape

Insert the DF2FGS tape into the rear cassette deck, label side up. Note that the cassette is actually mounted into the rails on the door of the deck. Press the door down, flush with the case, and push the black tab handle forward. You'll hear the tab handle assembly slide in with a solid clunk.

Press the RESTART key. Then put one of the blank tapes (we'll label ours FORM01) in the front deck. After "READY" appears, type "NEW⟩" ("⟩" means press the ENTER key).

STEP 2. Creating the Form

Type the desired form on the screen in the exact same manner as described in the Diskette DATAFORM chapters (see chapter 2). We'll use the following form for our examples. Remember to use the numeric pad to help create the form.

NUMERIC PAD

Display key must be pressed to effect cursor position.

Datapoint Screen

NAME: |_____

NUMBER: |_____

ORIGIN: |_____

DESTINATION: |_____

the operator will fill in the underlined parts

vertical bars
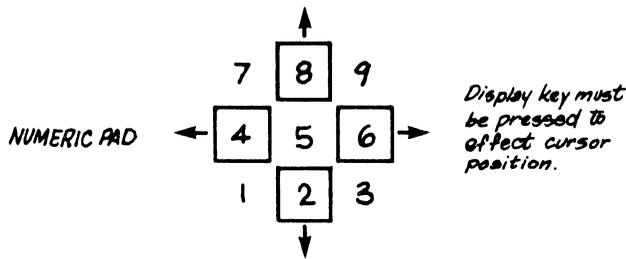
CREATING THE FORM
Drawing No. 65

STEP 3. Load Finished Form into Memory

After you have typed the form in exactly the format you like, hold down the DISPLAY key and then press the CANCEL key.

STEP 4. Assigning Field TYPEs

After the "BYTES LEFT" message has appeared, keyin "TYPE2". Enter desired edits in each field.

TYPEs allowed:

N Numeric; allows one decimal point with a leading minus sign; left justified and trailing blanks

A Alphabetic only

D Numeric only; left justified with trailing blanks

M Mixed; Alphabetic and Numeric

L Same as N, but with trailing zeros

R Same as N, but right justified and left zero filled

B Same as N, but right justified and left blank filled

No Entry -- No field restrictions

NOTE: These TYPEs are different than the TYPEs allowed in Diskette DATAFORM.

After all the fields are defined, press DISPLAY/CANCEL.

Datapoint Screen

NAME : A_____
NUMBER : N_____
ORIGIN: A_____
DESTINATION : A_____

During the TYPE assignment the form reappears and the letters A and N (or others) are typed in

FIELD TYPES
Drawing No. 66

STEP 5. Assigning REQUIREd fields
After "READY" has appeared on the screen, type "REQ⅃".

Conditions allowed:

R Operator must enter at least one character

F Operator must fill all positions

B Combines both Required and Fill; field may not be skipped and must be completely filled in

P The program enters the data

S Must execute program before record may be written

No Entry -- Operator may skip this field

NOTE: These conditions are different than the conditions allowed in Diskette DATAFORM.

Now press DISPLAY and CANCEL keys simultaneously.

Datapoint Screen

NAME: R_____

NUMBER: _____→

ORIGIN: _____→

DESTINATION: R_____

the operator is allowed to skip these fields, but not the others

REQUIRED TYPES
Drawing No. 67

STEP 6. Assigning SEMI-CONSTANT/CONSTANT Fields

After "READY" appears on screen, type "SEM♪" or "CON♪". Any text typed in the fields becomes automatically displayed each time the form is called up. If a Semi-Constant field is set up, the operator may overwrite the field during Data Entry mode. Then press DISPLAY/CANCEL.

Datapoint Screen

NAME: _____

NUMBER: _____

ORIGIN: NEW YORK____

DESTINATION: _____

*This will be automatically typed in during this operation as most items will be shipped from N.Y., and the operator will be saved the trouble*

SEMI-CONSTANT/CONSTANT FIELDS
Drawing No. 68

STEP 7. Assigning Programs to Fields

As mentioned earlier, Level II of DATAFORM allows programs to be written in a high level language that performs a wide variety of tasks. This step or pass of the Form Generator permits the system designer to type in the letters A through Z which corresponds to the maximum 26 programs which may be assigned to one form.

We can assume, for the sake of an example, that we've written a short program that checks for the correct range in a customer number.

NAME: _____

NUMBER: A_____ ←

ORIGIN: NEW YORK ____

DESTINATION: _____

Program A checks the range of the Number field
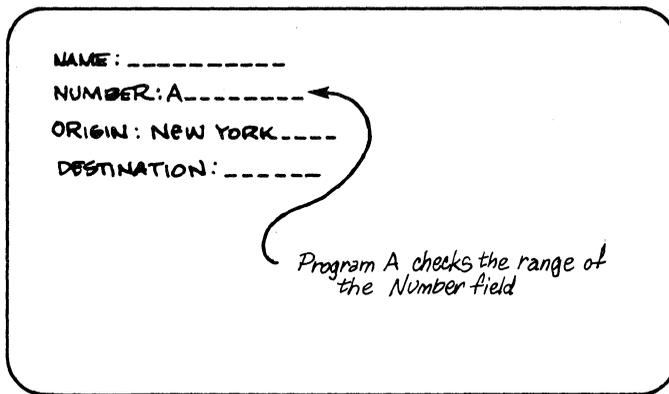
PROGRAM ASSIGNMENT
Drawing No. 69

Remember that the use of these programs is optional. If you feel the Form Generator features will meet system requirements, this step or pass may be skipped. If you call for a program in this step, and then elect not to write it, don't worry. If the Interpreter can't find the referenced program, it keeps going.

Program writing is explained in the Level II Cassette DATAFORM section.

STEP 8. LINKING to Other Forms

Should you desire to link to another form, you would do it at this point by typing "LINK↲". The message "NEXT FORM = 000:" will appear. This means you haven't set any links yet. If you had, it would read "NEXT FORM = 002:", for example. To set a link to Form 03, type in "003↲" (or "3↲"). DISPLAY/CANCEL is not needed here. Since Cassette LINKing works the same as Diskette LINKing, see Part III of Chapter 2 for complete LINKing details.

STEP 9. Write Form with Edits to Front Tape

After "READY" has reappeared, type "OUT↲". Had a program been assigned to this form, the message "DO YOU HAVE A PROGRAM TAPE?" would be displayed. An answer of "NO↲" would allow you to proceed without the program.

Any error messages which occur during form generation are explained in Appendix B.

STEP 10. Catalog the Form

After the "DONE-LOAD NEXT SYSTEM" message appears, remove the DF2FGS tape from the rear deck and replace it with the system tape, DF2SYS. Immediately after pressing the RESTART key, press the DISPLAY and KEYBOARD keys down and hold them down until the "DF2 CONFIGURATOR" message appears. Holding these keys down forces the program to be in Configurator mode. The Configurator allows us to enter our form in the system catalog (a catalog is a record of where everything is, so once we enter our form in the catalog, the system will know that it exists).

Type "CAT⟩" to see if there already are any entries in the catalog. To delete any unwanted entries, type "CHOP01⟩". To enter the form in the catalog, type "IN∅1⟩". When the "READY" message appears, TAKE YOUR FORM TAPE OUT of the front deck.

Handy Configurator Features

These Configurator commands will be very useful to you as you use the CASSETTE DATAFORM system.

INT	Loads the Interpreter
CAT	Display the contents of the System tape (Catalog)
IN n	Add form n to the System Catalog
DEL n	Delete form number n.
CHOP n	Delete all forms from n to the end of the catalog
REP n	Replace form n.
DUP	Duplicate the System tape without forms into the blank cassette in the front deck.
DUP-ALL	Same as DUP but also copies all cataloged forms.
OUT n	Copy file n to a scratch cassette in the front deck. This allows you to modify a form and then recatalog it.
COPY	Copy the data entered via your form to the cassette in the front deck.
FPRINT n	Print catalogued form n.
FPRINT ALL	Print all catalogued forms
DPRINT	Print the data tape contents

These commands are explained in detail in the CASSETTE DATAFORM User's Guide.

STEP 11. Data Entry Operating Modes

After the message "READY" is displayed, type in "INT⟩" to load the Interpreter. When the message "DF 2 INTERPRETER" is displayed, place your remaining blank tape in the front deck and type "START 001⟩" (or "S 1⟩"). You will have to answer YES to the question "ARE YOU SURE?".

There are other modes which may be used in data entry, and they are as follows:

Function	Key	Operation
START n	S n	START - initialize a data tape
ADD n	A n	ADD to the end of a data tape
CONTINUE	C	CONTINUE data entry after modification
LOAD n	L n	LOAD the specified form
DATA	D	Switch to DATA entry mode

MOD n	M n	MODIFY data records
FIND n	F n	FIND or search for matching data record
END	E	Write END of file on data tape
REW	R	REWIND data tape to first record

n means the number of the form

STEP 12. Entering Data

Type the data into the form (as an operator would) and observe field checking operations.   Correct errors by using the CANCEL and BACKSPACE keys. The following keys are also used:

Function	Key
Write data to tape	DISPLAY and Period
Forward tab	ENTER
Backward tab	DISPLAY and 3
Duplicate field	DISPLAY and Zero

NOTE: The Data Entry Numeric Pad Template from the Diskette chapter can be used here, with the exception of the Numeric 5 key (Delete Record).

NAME: ART BUCHWALD___
NUMBER: 444_____
ORIGIN: NEW YORK____
DESTINATION: WASHINGTON D.C.

The operator then presses DISPLAY and PERIOD to write this information to tape.

THE FINISHED PROGRAM IN OPERATION
Drawing No. 70

```
NAME: GEORGE ORWELL _ _ _ _
NUMBER: 1984 _ _ _ _ _ _ _
ORIGIN: LONDON _ _ _ _
DESTINATION: LOS ANGELES _ _ _

        Note here that the operator elected to write
          over the semi-constant field.
```

SAME FORM FILLED IN DIFFERENTLY
Drawing No. 71

After all the necessary data is entered in the fields for each form press DISPLAY and Period to write the data to tape. The form will automatically be displayed again with blank fields, awaiting keyin at the first non-Constant field.

STEP 13. Closing the Cassette Data File

After all the forms are entered, press the DISPLAY and 4 keys. After "READY" has appeared, type "END⟩" (or "E⟩") to close the data tape (front tape). The "ARE YOU SURE" question asks if you are at the end of the file. Since you are at the end of the file, answer "YES".

STEP 14. Reviewing and Modifying the Data

If necessary, type "MOD⟩" (or "M⟩") to rewind the tape and begin the manual search by displaying the first record. To examine subsequent records, press DISPLAY and 9. This advances the tape one record at a time and displays its contents. The tape search can also be reversed by using DISPLAY and 8.

After the new information is typed into each field, press ENTER. Then, to rewrite the modified record to cassette, press DISPLAY and Period. Because of the exact positioning required, you can only reposition and remodify a record four times.

You may now modify another record or press DISPLAY/4 then employ any of the modes listed in Step 11.

SUMMARY

Although this example is greatly simplified, it is, nonetheless, a working data entry system. It demonstrates the use of the DATAFORM Form Generator and System Interpreter (Monitor and Configurator). Additional sophisticated error checking and data manipulation power can be added by using field programs written in the DATAFORM programming language. These will be explained in the Level II part of this chapter, which follows.

Level II Step-by-Step Guide

Here's a summary of the steps that are necessary to create a finished field program:

1. Design the needed program
2. Type in the field program; it will be put on the blank tape
3. Compile the field program (convert it to computer readable language)
4. Combine the compiled program and the forms tape
5. Test the finished data entry system

STEP 1. Designing the Program

For purposes of illustration, we will use the same form that we used in Level I, and add a field program to operate on the Number field.

When our form was generated, the Number field required an eight digit numeric right justified blank-filled field. During the Program Assignment pass (PRO), this field did not have a program assigned. However, now we'll assign a program that will examine the number and reject it if it is greater than 1,999,999. If it is accepted, the cursor will proceed to the next field. If it is rejected, an error message will be displayed on line 12 (bottom line) of the screen, and the cursor will be repositioned at the beginning of the keyin field for re-entry.

Now, let's keyin a DATAFORM program that will do this for us.

STEP 2. Typing in the Program

Mount the DF2PGS tape in the rear deck, a blank tape in the front deck and press RESTART. When the tape has finished loading, the screen will look like this:

```
DF2 EDIT
CMP, OLD, NEW, DUP; PARAMETERS
?
```

Drawing No. 72

Type "NEW⟩" (or "N⟩"). This tells the Program Generator System tape that we're going to make a new program and also calls for the Editor program (GEDIT). If you had wanted to modify an old program, "OLD⟩" (or "O⟩") would be keyed in at this point.

The screen will blank out and the cursor will appear in the lower left hand corner. Use the same basic Editor commands as those explained in the Diskette section (Part II of Chapter 3) to type in the following Field program.

area for labels · area for instructions

the arrowhead

```
>  BADNO    WORK "IMPROPER NUMBER"
   MAXNO    WORK "1999999"
   A*       IF INPUT GREATER MAXNO THEN BAD
            STORE
   BAD      MESSAGE BADNO
            AGAIN
   Tab 1
   (left edge)          Tab 2
```
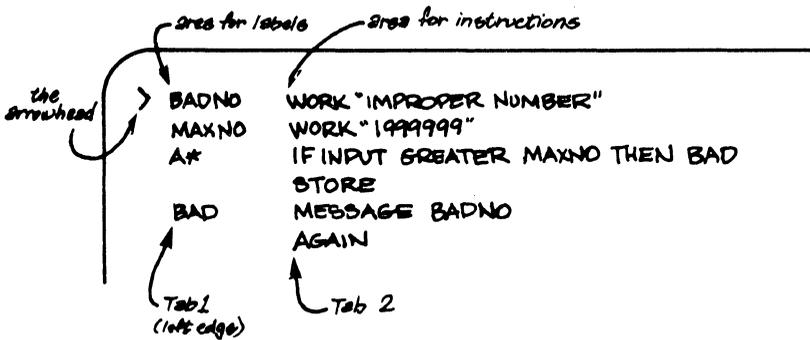
SCREEN WITH PROGRAM TYPED IN UNDER EDITOR OF DF2PGS
Drawing No. 73

STEP 3. Compiling the Program

After you have typed ":END⟩" (or ":E"), the DF2PGS program will automatically compile (make computer-readable) the text you typed in under the Editor (GEDIT).

When the Compiler has finished loading, it will display the "DF2 COMPILER" message.

If there is a printer attached to your Datapoint Processor, you will be asked "LIST ON LOCAL PRINTER?" If you want a copy of your program, type "YES⟩". It will then ask "LIST CODE TOO?" Since you probably won't ever need to see the actual machine code for your program, type "NO⟩".

When the compilation and printing are complete, the following display will appear:

```
STORAGE USED IN DECIMAL: 42 RELOCATABLE; 0 COMMON
FIELD PROGRAMS:
            A    00030
END OF COMPILATION: NO ERRORS
```
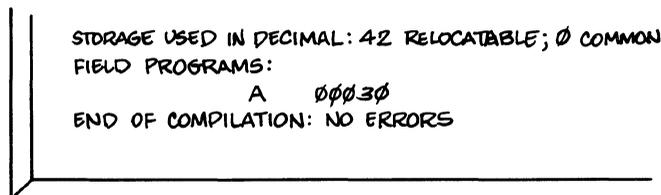
Drawing No. 74

STEP 4. Combining Form and Program

To add the Field Program to the form, it is necessary to repeat the Form Generation Phase.

A. Place the previously created Form tape (the one marked Form 1) in the front deck and the Form Generator Program (DF2FGS) Tape in the rear deck. Press RESTART.

B. When the Form Generator Program is loaded, it will display "DF2 GENERATOR" and await your keyin with "READY" and a blinking cursor. Since we want to re-edit our old tape, the first command to type is "OLD ".

C. After the "READY" message appears, the various passes (TYPE, REQ, LINK, SEM, CON, PRO) can be

redone. Since we have to assign our new program to the Number field, type "PRO⟩" to enter the Program Assignment Pass. To make the assignment, press the ENTER key until the cursor is in the first column of the Number field. Type 'A' to make the program assignment, then press the DISPLAY/CANCEL keys to return to the Monitor.

D. Now type "OUT⟩" to dump the form to the front tape. Since a program is now specified, the following will be displayed:

PROGRAM BASE ADDRESS nnnn DO YOU HAVE A PROGRAM TAPE?

Answer YES and you will be instructed to "PLACE TAPE IN REAR DECK AND PRESS ENTER". Remove the DF2FGS tape from the rear deck and replace it with the program tape (the one labeled "Program A"). Then, press ENTER and wait while the tapes move and the display "REPLACE DF2FGS TAPE, PRESS ENTER" comes up. Your program is now added to the form tape. Put the DF2FGS tape back into the back deck and press ENTER. Your program will now be added to the Form Generator tape, too. A message similar to the following will appear:

```
1134 BYTES LEFT
DONE-LOAD NEXT SYSTEM
```

Drawing No. 75

You can now remove both tapes.

STEP 5. Using the Complete Data Entry System
Load the DF2SYS tape in the rear deck and press RESTART. When the "DF2 INTERPRETER" and "READY" messages appear, put a blank tape in the front deck and type "S⟩". Answer "YES⟩" to the "ARE YOU SURE?" question because you really do want to start over.
After you see the "READY" message, remove the DF2SYS tape and put the tape labeled Form 1 in the rear deck. Now type "NEW⟩" and you're ready to use your form and field program for data entry.
If you try to type a number larger than 1,999,999 in the NUMBER field, the program will display the message "IMPROPER NUMBER" and reject it by beeping and returning the cursor to the field.

```
NAME : ART BUCHWALD___
NUMBER : 2000000__  ◄
ORIGIN : NEW YORK_____
DESTINATION :_____   └ this number will be rejected
```

WRONG NUMBERS ARE REJECTED BY THE PROGRAM
Drawing No. 76

CHAPTER FIVE

CARTRIDGE DISK DATAFORM

PART I
Overview

Introduction

DISK DATAFORM was originally conceived as a CASSETTE DATAFORM supportive system. However, it has grown until today it is an integral DATAFORM system of its own. It still supports CASSETTE DATAFORM, however, and greatly speeds up the generation of forms and the editing and compilation of programs for cassettes.

If, after reading this chapter, more specific information concerning DISK DATAFORM is needed, it can be found in the DISK DATAFORM User's Guide. This chapter, however, assumes a certain degree of knowledge of DATAFORM concepts (from the previous chapters) and familiarity with Datapoint equipment.

Equipment Needed:

1. A Datapoint 2200 or 5500 with 16K or more of memory. Forms and programs generated on one Datapoint Processor can be used interchangeably with any other Datapoint Processor

2. At least one Cartridge Disk Drive. A second Disk Drive is handy for duplicating your Cartridge Disk Pack.

3. At least one Cartridge Disk Pack with System Files on it; the DISK DATAFORM Files can be put on the pack with the use of two DISK DATAFORM cassettes (see Programs Required section).

4. A printer is extremely useful for printing out data files, forms programs, and doing a multitude of tasks. Any of the printers which are presently offered, such as the Servo, Belt, Matrix, 300 or 600 LPM Printers, can be used for these duties.

Programs Required:

NOTE: Any programming system will undergo a series of improvements and changes during usage, so the user should request the latest version of the DISK DATAFORM programs. Since this book covers only the fundamental elements of DATAFORM, no problems should be encountered using a later version of the programs.

The DISK DATAFORM program is on two cassettes (Model Code 20081). The contents of each cassette should be added to your cartridge disk pack by placing the cassette in the front deck and typing "MIN; A0 ".

Once you've added the contents of both cassettes to your

cartridge disk pack, the following files will have been added to the cartridge disk:

DFEDIT/CMD	DOSDF2 Program Editor
DFCMP/CMD	DOSDF2 Program Compiler
DFGEN/CMD	DOSDF2 Form Generator
DFINT/CMD	DOSDF2 Interpreter
DFCON/CMD	DOSDF2 Configurator
CDF2CON/ABS	DF2 Configurator
CDF2INT/ABS	DF2 Interpreter

Also, 15 files labeled DDFXTND/OVA through DDFXTND/OVO, which are the DOSDF2 Extended Interpreter Overlays, and 15 files labeled CDFXTND/OVA through CDFXTND/OVO, which are the cassette DF2 Extended Interpreter Overlays, are included. Thus, 37 new files will have been added to your Cartridge Disk.

NOTE: The Overlay file names must not be changed.

Additionally, other system utilities such as DATAPOLL, the automatic communications polling and answering program, are available in cassette form.

The ensuing parts of this chapter represent abbreviated step-by-step guides for Levels I and II, expanded to include any features peculiar to DISK DATAFORM operation.

PART II
Level I Step by-Step Guide

Here's a brief outline of how to create a working DATAFORM Data Entry program using the Cartridge Disk System.

STEP 1. The Form Generator
With the Disk System on and your Cartridge Disk Pack in Drive 0, press the RESTART key.

NOTE: If a Datapoint 5500 is used, both the RESTART and RUN keys must be pressed simultaneously.

Your form name must follow the same rules as the names of Diskette and Cassette forms. That is, the name is six characters or less and a number from 1 to 99 follows it. We'll use SEND01 for the name of our form.
After the Disk Operating System (DOS) displays "READY", type "DFGEN SEND01⟩".
The "DOS   DF2 GENERATOR" and "READY" messages will appear, with a blinking cursor underneath. Since we're creating a new form, type "NEW⟩" ('⟩' means press the ENTER key). A cursor should appear in the extreme upper left position of the screen, awaiting creation of your form.

STEP 2. Creating the Form
Type the desired form on the screen using the numeric pad features. See Part II of Chapter 2 for the form creation explanation. Type "DISPLAY/CANCEL" when you're finished.

STEP 3. Form Creation Passes
Use the instructions from the previous chapter (CASSETTE DATAFORM) to go through these form creation passes.

1.   Assigning Field TYPEs.
2.   Assigning REQUIRED Fields.
3.   Assigning SEMI-CONSTANT/CONSTANT Fields.
4.   Assigning PROGRAMs to Fields.
5.   LINKing to other Forms.

STEP 4. Recording the Form onto the Disk
By now, you should have completed all the necessary steps and are left with "READY" on the screen. Type "OUT⟩" and the displays "PROGRAM BASE ADDRESS: nnnn", "nnnn BYTES LEFT" and "DONE-LOAD NEXT SYSTEM" will appear (nnnn is some number). If an error is detected, it will be indicated at this time.

STEP 5. Operating the Data Entry System
Type "DFINT SEND01⟩" to start data entry. When the displays "DOS DF2 INTERPRETER" and "SELECT DATA MODE" are shown on

the screen, type "START⏎" (or "S⏎"). After you answer "YES⏎" to the "ARE YOU SURE?" question, the form will appear waiting for data entry in the first non-constant field.

Type the data (as an operator would) and observe field checking operations. Correct errors by use of the CANCEL and BACKSPACE keys. The following keys are also used:

Function	Keys
Write Data to Disk	DISPLAY/Period
Forward Tab	ENTER
Backward Tab	DISPLAY/3
Duplicate Field	DISPLAY/0
Erase Form Data	DISPLAY/6

STEP 6. Closing the Disk Data File

After all the necessary data is entered in the fields, press DISPLAY and Period to write the data to disk. The form will automatically be displayed again with blank fields, awaiting keyin at the first non-Constant field. If you have entered all of your data, press DISPLAY and 4 to go back to Monitor. Then type in "END⏎" (or "E⏎") and be sure to answer the question "ARE YOU SURE?". This question reminds you that you should be positioned at the end of your data file before saying "END" as it writes an End Of File (EOF) mark wherever you are. END does one additional thing in DISK DATAFORM; it returns you to the Operating System.

STEP 7. Adding to and Modifying the Data

Type "ADD⏎" (or "A⏎") to position to the End Of File and begin adding additional records. Type "MOD⏎" (or "M⏎") to position to the beginning of the data file. Then use DISPLAY and 9 to advance through the data file (displaying each time) a record at a time. Note that DISPLAY and 8 takes you back a record at a time.

Records displayed can be changed according to information re-typed (be sure to press ENTER), and then rewritten to the disk when DISPLAY and Period are pressed. The following modes may also be used:

Function	Key	Operation
CONTINUE	C	CONTINUE data entry after modification
LOAD n	L n	LOAD the specified form
DATA	D	Switch to DATA entry mode
FIND n	F n	FIND or search for matching data record
REW	R	REWIND data file to first record
		n means the number of the form

The Configurator

To generate a cassette system tape, you'll have to use the cassette Configurator. See the Level 1 CASSETTE DATAFORM section for a description of how to use the Configurator. The forms to be written to tape must first be cataloged, using the "IN" command.

Then, type "DFCON SEND".

NOTE: This is the only time the form number must not be included in the name.

A file named SEND/CAT is generated.

Use the "DUP ALL" or "LGO" command to write a system tape (DF2SYS) with all the forms specified in the catalog (contained in File SEND/CAT).

Additional information concerning the Configurator can be found in the DISK DATAFORM User's Guide.

SUMMARY

Although this example is greatly simplified, it is, nonetheless, a working data entry system. Additional sophisticated error checking and data manipulation power can be added by using field programs written in the DATAFORM language. These will be explained in Part III, which follows.

PART III
Level II Step-by-Step Guide

Here's the summary of the steps that are necessary to create a finished field program:

1. Design the needed program
2. Type in the program
3. Compile the program (convert it to computer readable language)
4. Combine the compiled program and the forms
5. Test the finished data entry system

STEP 1. Designing the Program

For purposes of illustration, we will use the same form that we used in Level I of Cassette DATAFORM, and add a program to operate on the Number field.

When our form was generated, the Number field required an eight digit numeric right justified blank-filled field. During the Program Assignment pass (PRO), this field did not have a program assigned. However, a program will now be assigned to examine the number and reject it if it is greater than 1,999,999. If it is accepted, the cursor will proceed to the next field. If it is rejected, an error message will be displayed on line 12 (bottom line) of the screen, and the cursor will be repositioned at the beginning of the keyin field for re-entry.

Now, let's keyin a DATAFORM program that will do this for us.

STEP 2. Typing in the Program

The Disk System should be turned ON, the Cartridge Disk Pack installed in Drive 0 and the drive set to RUN. Press RESTART and when "DISK OPERATING SYSTEM" and "READY" appear, call up the General Editor (GEDIT) by typing "DFEDIT SEND01 ;Dג".

The ";D" allows you to use DATAFORM tabbing (explained in part III of Chapter 3).

Type in the following field program using the Editor functions described in Part III of Chapter 3.

STEP 3. Compiling the Program

After you have typed ":ENDג" (or ":E"), the Operating System will come up READY. To compile (make computer-readable) the text you typed in under the Editor (GEDIT), type "DFCMP SEND01ג".

You will have to answer "LIST ON LOCAL PRINTER?" if there's a printer attached to your system.

If there are any errors, the offending statement will be flagged with an asterisk, and an error type shown. These diagnostics will be inserted between the lines of the printout, but the screen display will only show the lines of code which are in error. The End of Compilation will then show the total number of errors. To correct any errors, simply re-edit and re-compile the program.

STEP 4. Combining Form and Program

To add the Field Program to the form, it is necessary to repeat the Form Generation Phase. However, if you did indeed assign the correct program letter during generation, a short procedure is all that is required. It's done in three steps as follows:

1. "DFGEN SEND01⟩"

2. "OLD⟩"

3. "OUT⟩"

Since we did not assign a program letter during generation (Level I), we should do so as follows:

1. "DFGEN SEND01⟩"

2. "OLD⟩"

3. "PRO⟩ "

4. Press the ENTER key until you are in the Number
   Field, type an 'A' and then press DISPLAY/CANCEL

5. "OUT⟩"

STEP 5. Operating the Complete Data Entry System.

Begin data entry by typing "DFINT SEND01⟩ ". Then enter "START" or simply "S⟩" to begin a new file. Enter data in the same manner as you did in Level I. Notice, however, that if you try to type a number larger than 1,999,999 in the Number Field, your program will display the message "IMPROPER NUMBER" and reject it by Beeping and returning the cursor to the beginning of the field.

APPENDIX A

DISKETTE DATAFORM

DISKETTE DATAFORM SPECIFICATION STATEMENTS

DATA   Output data record location

WORK   Define work area

COMMON   Define common area

REDEFINE   Redefine labels

FIELD   Define fields

DISKETTE DATAFORM EXECUTABLE STATEMENTS

ALIGN   Move and align decimal point

BACKSPACE   Position data file back n records

DELETE   Delete current record from the data file

ENTRYMODE   Change to ADD/START Data Entry mode

FIELDNO   Put current field number in label

MODIFYMODE   Change to Modify mode

PEOF   Position to end of data file

READ   Read next record into data area

WEOF   Write an End Of File (EOF) mark on data file

CONVERT   Field conversion using two tables

LOOKUP   Indexed table lookup

MOVE   Data move

SET   Fill entire field with a single character

ADD   Addition

SUBTRACT   Subtraction

MULTIPLY	Multiplication
DIVIDE	Division
IF	Comparison

EQUAL	Equal to
GREATER	Greater than
LEQ	Less than or equal to
GEQ	Greater than or equal to
LESSTHAN	Less than
NOTEQUAL	Not equal to
INRANGE	In range of table
INTABLE	In table
NOTINRANGE	Not in range of table
NOTINTABLE	Not in table
CK10	Mod 10 check digit routine
CK11	Mod 11 check digit routine

WRITE	Write output data to diskette
MESSAGE	Display a message on line 12
SHOW	Display current field data
BEEP	Beep sound
CHAIN	Load next form
FORMSHOW	Display form
GOTO	Transfer control
CALL	Enter Subroutine
RETURN	Exit Subroutine
CHANGE	Move field pointer
RESET	Reset field pointer

DISKETTE DATAFORM EXIT PATHS

AGAIN	Repeat field
CLOSE	END the file and return to the Operating System
END	Write data and do next form
NEXT	Keyin next field

STORE	Move input to output and go to next field

DISKETTE DATAFORM DATA BUFFERS

INPUT	Keyed in data buffer -- one field
OUTPUT	Form output data buffer -- one field
NULL	Location of Binary zero in data buffer
RETRY	Data entry mode flag 0 = Data Entry mode 1 = Modify mode

DISKETTE DATAFORM
COMPILER ERROR MESSAGES

NAME REQUIRED

The name of the program source file must be typed in the initial command line.

BAD LABEL INITIATOR

A character that was neither a decimal point nor a space nor alphanumeric appeared in column 1 of the input line.

INVALID OCTAL

The character string pointed to by the asterisk contains a character which is not in the set 0-7.

ILLEGAL OPERATOR

Something other than those operators appearing in this Appendix was the first nonblank symbol after column 1 (or after the label, if one exists).

NUMBER FROM 1-249 EXPECTED

The indicated symbol is non-numeric, or if numeric, not in the specified range.

COMMA EXPECTED

The symbol after the first number in a DATA statement was not a comma.

FIELD2 IS LESS THAN FIELD1

In a DATA statement, the second field is less than the first.

LABEL REQUIRED

The DATA, REDEFINE and WORK statements all require a label.

DOUBLE QUOTE ASSUMED

A pre-defined constant either in WORK or COMMON statements should be terminated by a double quotation mark. If it is not there, it is assumed.

ILLEGAL LITERAL

In a table, every item enclosed in double quotation marks must be of equal length. Those that are of different length than the first item are flagged in error.

IMPROPER CONTINUATION

If a COMMON or WORK table is continued from a line, the following line must have a blank in column one, and the first symbol on the line must be a double quotation mark. If either of these is not the case, the continuation is an improper one.

UNDEFINED LABEL

A label is referenced which is neither one of the eight pre-defined labels, nor defined elsewhere in the program.

MISSPELLED WORD

A specific Reserved Word -- for example, the TO in an ADD statement -- has been misspelled. The misspelled word is assumed to be the one expected, and the next symbol is expected to be a legal label.

ILLEGAL CONDITION

The connective in an IF statement is not one of those listed in this Appendix. Nothing about the connective is assumed.

DUPLICATE LABEL

The label beginning the line listed is duplicated previously in the program (or it is one of the eight pre-defined labels). The second and any subsequent definitions of the label are ignored.

MAXIMUM LABELS REACHED

The maximum number of labels allowed by the compiler is fixed at 246, excluding the pre-defined labels. All labels after this maximum is reached are ignored.

COMMON LIMIT EXCEEDED

The COMMON block may not exceed 100 bytes or reserved amount. Anything defined as COMMON after this length will not be accepted.

PROGRAM COUNTER ERROR

The program counter, at the end of pass two does not equal the program counter at the end of pass one. This is an internal compiler error message.

COMMON PRECEDES RESERVE

A RESERVE statement was encountered after a COMMON statement. Since reserve changes the starting address of common, this is an illegal situation.

DISKETTE FORM GENERATOR ERROR MESSAGES

BAD FORM NAME

The form name specified in the command line did not end with a two digit number.

BAD FORM

There is no form in memory or the form in memory cannot be written out, or have any pass except REVISE executed, because of some error condition.

NO FIELDS DEFINED

Forms may not be created without at least one field (this may be a keyin only field).

MORE THAN 126 FIELDS

During image generation more than 126 data fields were defined. The form must be revised before it may be written out.

MORE THAN 249 DATA

During image generation more than 249 data characters were defined. The form must be revised before it may be written out.

XXX DATA
YYY BYTES LEFT

The messages appear immediately after the image generation phase of form generation. They are for information only.

YYY BYTES OVER

If this message appears after image generation, the form image, data area and edit table have combined to overflow the user space. Something must be reduced.

PROGRAM BASE ADDRESS XXXXX

This is the address (in decimal) of the first location in the user space available for program code.

PROGRAM MISSING

A program specified in the program pass is not contained in the program file (or there is no program file at all).

DISKETTE DATAFORM INTERPRETER ERROR MESSAGES

CONTINUOUS BEEPING DURING DATA ENTRY

An illegal constant has been defined at form generation time. The constant must be reset to conform with the edit criteria before proceeding.

CONTINUOUS CLICKING DURING DATA ENTRY

An all-Constant form with no keyin field has been loaded. The form must be corrected before data entry may proceed.

SELECT DATA MODE

No START, ADD, MOD or FIND command has been executed.

END OF DATA

End of file has been reached on the data file.

DATA FILE OPEN

An "open" type operation was attempted before ending the current data file.

DATA FILE CLOSED

A close type instruction was attempted before opening the current data file.

NO FIELDS

Somehow you got a form with no fields defined.

NO LINK SET

The operator attempted to load the linked form and no link was set.

FILE SPACE FULL

The data file has reached the limit of the disk space. All records posted to the write routine have been written and an end-of-file marker is written. The user should type "OS" to return to the operating system. You can, however, modify the file if desired.

DISKETTE DATAFORM COMMON SYSTEM ERROR MESSAGES

FILE MISSING or FORM MISSING

The form number specified is not present as SYSNAMnn/DFF.

In the Interpreter, this message may mean that the next form specified in the current form's link is not present, or that your command assumes that there is a form in memory (e.g., DATA) and none is loaded.

NAME REQUIRED

The initial command line did not include the system name or form name required by the program.

ILLEGAL DEVICE SPECIFICATION

The initial command line included a disk drive specification which was improperly formated.

BAD NUMBER

The form number may have been omitted, out of range (I-99) or non-numeric. Or, the form specified is not in the catalog. Note that if the form number is omitted in a command which optionally accepts form numbers (e.g., START n) the command line cannot end with a space.

PRINT UTILITY ERRORS

BAD SYSTEM NAME

Name specified in the command line cannot possibly be a system name since it is greater than 8 characters.

NO PRINTER

A DF11PRT command was attempted with no printer connected or switched on.

APPENDIX B

CASSETTE DATAFORM

CASSETTE DATAFORM SPECIFICATION   STATEMENTS

DATA   Output data record locations

WORK   Define work area

COMMON   Define common area

EQU   Define octal addresses

REDEFINE   Redefine new labels to old labels

FIELD   Define fields

CASSETTE DATAFORM EXECUTABLE STATEMENTS

ALIGN   Move and align decimal point

CONVERT   Field conversion through table lookup

LOOKUP   Indexed table lookup

MOVE   Data move

SET   Fill entire field with a single character

ADD   Addition

SUBTRACT   Subtraction

MULTIPLY   Multiplication

DIVIDE   Division

IF   Comparison

EQUAL	Equal to
GREATER	Greater than
LEQ	Less than or equal to
GEQ	Greater than or equal to
LESSTHAN	Less than
NOTEQUAL	Not equal to
INRANGE	In range of table
INTABLE	In table
NOTINRANGE	Not in range of table

NOTINTABLE	Not in table
CK10	Mod 10 check digit routine
CK11	Mod 11 check digit routine
WRITE	Write output data to tape
MESSAGE	Display a message on line 12
SHOW	Display current field data
BEEP	Beep sound
CHAIN	Load next form
FORMSHOW	Display form
GOTO	Transfer control
CALL	Enter Subroutine
RETURN	Exit Subroutine
CHANGE	Move field pointer
RESET	Reset field pointer

CASSETTE DATAFORM EXIT PATHS

AGAIN	Repeat field
CLOSE	END the file and return to the Operating System
END	Write data and do next form
NEXT	Keyin next field
STORE	Move input to output and go to next field

CASSETTE DATAFORM DATA BUFFERS

INPUT	Keyed in data buffer -- one field
OUTPUT	Form output data buffer -- one field
NULL	Location of Binary zero in data buffer
RETRY	Data entry mode flag 0 = Data Entry mode 1 = Modify mode

CASSETTE DATAFORM COMPILER ERRORS

PARITY ERROR: A/C?

A parity error persisted on a cassette read operation after five retries. A response of 'A' will abort the compilation; a response of 'C' will use the bad block as if nothing were wrong with it, and continue the compilation.

BAD LABEL INITIATOR

A character that was neither a decimal point nor a space nor alphanumeric appeared in column 1 of the input line.

INVALID OCTAL

The character string pointed to by the asterisk contains a character which is not in the set 0-7.

ILLEGAL OPERATOR

Something other than those statement operators appearing in this Appendix was the first nonblank symbol after column 1 (or after the label, if one exists).

NUMBER FROM 1-245 EXPECTED

The indicated symbol is non-numeric, or if numeric, not in the specified range.

COMMA EXPECTED

The symbol after the first number in a DATA statement was not a comma.

FIELD2 IS LESS THAN FIELD1

In a DATA statement, the second field is less than the first.

LABEL REQUIRED

THE DATA, REDEFINE and WORK statements all require a label.

DOUBLE QUOTE ASSUMED

A pre-defined constant either in WORK or COMMON statements should be terminated by a double quotation mark. If it is not there, it is assumed.

ILLEGAL LITERAL

In a table, every item enclosed in double quotation marks must be of equal length. Those that are of different length than the first item are flagged in error.

IMPROPER CONTINUATION

If a COMMON or WORK table is continued from a line, the following line must have a blank in column one, and the first symbol on the line must be a double quotation mark. If either of these is not the case, the continuation is an improper one.

UNDEFINED LABEL

A label is referenced which is neither one of the eight pre-defined labels, nor defined elsewhere in the program.

MISSPELLED WORD

A specific Reserved Word -- for example, the TO in an ADD statement -- has been misspelled. The misspelled word is assumed to be the one expected, and the next symbol is expected to be a legal label.

ILLEGAL CONDITION

The connective in an IF statement is not one of those listed in this Appendix. Nothing about the connective is assumed.

DUPLICATE LABEL

The label beginning the line listed is duplicated previously in the program (or it is one of the eight pre-defined labels). The second and any subsequent definitions of the label are ignored.

MAXIMUM LABELS REACHED

The maximum number of labels allowed by the compiler is fixed at 125, including the pre-defined labels. All labels after this maximum is reached are ignored.

COMMON LIMIT EXCEEDED

The COMMON block may not exceed 100 bytes. Anything defined as COMMON after this length will not be accepted.

PROGRAM COUNTER ERROR

The program counter at the end of pass two does not equal the program counter at the end of pass one. This is an internal compiler error message.

CASSETTE DATAFORM COMMON SYSTEM ERRORS

FILE MISSING

Some form, present in the catalog, is missing on the system tape, or the file marker necessary for positioning the input tape is missing, or a form is short (i.e., it doesn't contain the necessary 6 blocks).

BAD NUMBER

The form number may have been omitted, out of range (1-99) or non-numeric. Or, the form specified is not in the catalog. Note that if the form number is omitted in a command which optionally accepts form numbers (e.g., START n), the command line cannot end with a space.
In the Interpreter, this message may mean that the next form specified in the current form's link is not in the catalog, or that your command assumes that there is a form ˙in memory (e.g., ENTER) and none is loaded.

PARITY ERROR ON DECK n

Indicates a parity error was detected -- where n is the cassette deck number (1 is the rear deck and 2 is the front deck). Before this message is displayed, four attempts are made to read the record.

INTERNAL ERROR x ON DECK n

This message indicates a tape or tape deck failure. The x is replaced by a letter indicating the error condition as follows:

    D Parity error
    E End of tape
    F End Of File (EOF)
    G Unfindable file
    Z Write failure

Generally these errors occur only if something is severely wrong with the cassette. Error Z may occur if the Write Protect tab has been punched on the cassette or if the tape is improperly inserted in the deck. If Error Z occurs often, a hardware failure should be suspected.
The letter n in the message is replaced by the number of the tape deck on which the error occured (deck 1 is the rear deck and deck 2 is the front deck).

CASSETTE DATAFORM CONFIGURATOR ERRORS

END OF FILE MISSING

End of tape reached during COPY -- an End Of File (EOF) marker is automatically written.

AUTO NOT SET

This is given in response to a MANUAL command if the Auto-Load entry is not set.

FORM CATALOG IS UNLOADABLE, DUMMY CATALOG GENERATED

File 1 of the forms catalog is in error and a dummy (full) catalog has been substituted. Steps should be taken to recover the system.

NUMBER IN USE

The form number specified for an IN command was already assigned.

CASSETTE DATAFORM GENERATOR ERRORS

BAD FORM

The form just written is unloadable due to parity errors or missing blocks.

MORE THAN 126 FIELDS

During image generation, more than 126 data fields were defined. The form must be revised before it may be written out.

MORE THAN 245 DATA

During image generation, more than 245 data character slots were defined. The form must be revised before it may be written out.

XXX DATA
YYY BYTES LEFT

The messages appear immediately after the image generation phase of form generation. They are for information only.

YYY BYTES OVER

If this message appears after image generation, the form image, data area and edit table have combined to overflow the user space. Something must be reduced.

CASSETTE DATAFORM INTERPRETER ERRORS

CONTINUOUS BEEPING DURING DATA ENTRY

An illegal Constant has been defined at form generation time. The Constant must be reset to conform with the edit criteria before proceeding.

CONTINUOUS CLICKING DURING DATA ENTRY

An all-Constant form with no keyin field has been loaded. The form must be corrected before data entry may proceed.

TAPE CLOSED

No START, ADD, MOD or FIND command has been executed.

END OF TAPE

End of tape was encountered during data entry or an unrecoverable tape error occured during writing. If the error occurs due to end of tape, the data tape is automatically backspaced twice and an End Of File (EOF) marker is written. If it is a write-parity error, the EOF is written where the record would have been. This means that the last two records since the operator is keying one in are lost. Totals being accumulated by field programs may no longer be valid.

REWRITE LIMIT REACHED, TO VIEW PRESS ENTER

During modification, the record in memory has been rewritten 4 times and cannot be rewritten again; however, it can be interrogated. To reset the Rewrite Counter to zero, use the COPY command of the Configurator.

BAD DATA

There is a parity error in the data.

BAD FORM

There is a parity error in the form or a block is missing.

END OF DATA

End Of File has been reached on the data tape.

TAPE OPEN

An "Open" type operation was attempted before ending the current data tape.

THE CONFIGURATOR

The DATAFORM Configurator is a part of the System Program (DF2SYS) tape. Some of its features include:

INT	Loads the Interpreter
CAT	Display the contents of the System tape (Catalog)
IN	Add new forms to the System Catalog
DEL	Delete forms from the System Catalog
CHOP	Delete multiple forms from the System Catalog
REP	Replace forms on the System Catalog
DUP	Duplicate the System tape without forms
DUP-ALL	Duplicate the System with forms
OUT	Recover a catalogued form
COPY	Copy a data tape
FPRINT	Print a copy of a catalogued form
FPRINT ALL	Print a copy of all catalogued forms
DPRINT	Print a copy of the data tape contents
LGO	Create a Load and Go Interpreter (one which has only the Interpreter, forms and a Boot loader, so that it can be brought up faster for data entry.

An in-depth explanation of the Configurator can be found in the CASSETTE DATAFORM User's Guide.

APPENDIX C

CARTRIDGE DISK DATAFORM

DISK DATAFORM SPECIFICATION STATEMENTS

DATA	Output data record locations
WORK	Define work area
COMMON	Define common area
EQU	Define octal addresses
REDEFINE	Redefine new labels to old labels
FIELD	Define fields

DISK DATAFORM EXECUTABLE STATEMENTS

ALIGN	Move and align decimal point
CONVERT	Field conversion using two tables
LOOKUP	Indexed table lookup
MOVE	Data move
SET	Fill entire field with a single character
ADD	Addition
SUBTRACT	Subtraction
MULTIPLY	Multiplication
DIVIDE	Division
IF	Comparison

EQUAL	Equal to
GREATER	Greater than
LEQ	Less than or equal to
GEQ	Greater than or equal to
LESSTHAN	Less than
NOTEQUAL	Not equal to
INRANGE	In range of table
INTABLE	In table
NOTINRANGE	Not in range of table

NOTINTABLE	Not in table
CK10	Mod 10 check digit routine
CK11	Mod 11 check digit routine

WRITE	Write output data to disk
MESSAGE	Display a message on line 12
SHOW	Display current field data
BEEP	Beep sound
CHAIN	Load next form
FORMSHOW	Display form
GOTO	Transfer control
CALL	Enter Subroutine
RETURN	Exit Subroutine
CHANGE	Move field pointer
RESET	Reset field pointer

DISK DATAFORM EXIT PATHS

AGAIN	Repeat field
CLOSE	END the file and return to the Operating System
END	Write data and do next form
NEXT	Keyin next field
STORE	Move input to output and go to next field

DISK DATAFORM DATA BUFFERS

INPUT	Keyed in data buffer -- one field
OUTPUT	Form output data buffer -- one field
NULL	Location of Binary zero in data buffer
RETRY	Data entry mode flag 0 = Data Entry mode 1 = Modify mode

DISK DATAFORM GENERATOR ERRORS

BAD FORM NAME

The form name specified in the command line did not end with a two digit number.

BAD FORM

The form in memory cannot be written out, or have any pass except REVISE executed, because of some error condition.

NO FIELDS DEFINED

Forms may not be created without at least one field (this may be a keyin only field).

NO ROOM FOR CONSTANTS

All fields in the form were defined using the caret, therefore Constants and Semi-Constants may not be entered.

MORE THAN I26 FIELDS

During image generation more than 126 data fields were defined. The form must be revised before it may be written out.

MORE THAN 245 DATA

During image generation more than 245 data characters were defined. The form must be revised before it may be written out.

XXX DATA
YYY BYTES LEFT

The messages appear immediately after the image generation phase of form generation. They are for information only.

YYY BYTES OVER

If this message appears after image generation, the form image, data area and edit table have combined to overflow the user space. Something must be reduced.

EXTENDED INTERPRETER MISSING

The Extended Interpreters must be catalogued exactly DDFXTND/OVA through /OVO. If the file is not present and an Extended Interpreter is required, this message will appear.

PROGRAM X MISSING

A program specified in the program pass is not contained in the program file (or there is no program file at all).

DISK DATAFORM CONFIGURATOR ERRORS

BAD SYSTEM NAME

Name specified in the command line cannot possibly be a system name since it is greater than 8 characters.

ILLEGAL SYSTEM NAME

During a DUP, DUP ALL OR LGO command one of the DATAFORM 2 cassette program files was not found.

VERIFY FAILURE

During a DUP, DUP ALL or LGO command, while re-reading the tape to verify it, a tape error was encountered.

PARITY ERROR ON DECK 2
COPY, OMIT or END?

A parity error was encountered on a tape being copied. The user has the option to copy the erroneous record (type 'C'), omit it but continue the copy (type 'O'), or terminate the copy at that point ('E') which writes an End Of File (EOF) marker on the new tape.

INTERNAL ERROR x ON DECK n

This message indicates a tape or tape deck failure. The x is replaced by a letter indicating the error condition:

      D parity error
      E end of tape
      F end of file
      G unfindable file
      Z write failure

Generally these errors occur only if something is severely wrong with the cassette. Error Z may occur if the Write Protect tab has been punched on the cassette or if the tape is improperly inserted in the deck. If error Z occurs often, a hardware failure should be suspected.

The letter 'n' in the message is replaced by the number of the tape deck on which the error occured (deck 1 is the rear deck, deck 2 is the front).

128

END OF FILE MISSING

End of tape reached during COPY -- an End Of File (EOF) marker is automatically written.

NUMBER IN USE

The form number specified for an IN command was already assigned.

NO PRINTER

A DPRINT or FPRINT command was attempted with no printer connected or switched on.

DISK DATAFORM INTERPRETER ERRORS

CONTINUOUS BEEPING DURING. DATA ENTRY

An illegal Constant has been defined at form generation time. The Constant must be reset to conform with the edit criteria before proceeding.

CONTINUOUS CLICKING DURING DATA ENTRY

An all-Constant form with no keyin field has been loaded. The form must be corrected before data entry may proceed.

SELECT DATA MODE

No START, ADD, MOD or FIND command has been executed.

END OF DATA

End Of File (EOF) has been reached on the data file.

DATA FILE OPEN

An "Open" type operation was attempted before ending the current data file.

DATA FILE CLOSED

A "Close" type instruction was attempted before opening the current data file.

NO FIELDS

Somehow you got a form with no fields defined.

NO LINK SET

The operator attempted to load the linked form and no link was set.

DISK DATAFORM COMPILER ERRORS

NAME REQUIRED

The name of the program source file must be typed in the initial command line.

BAD LABEL INITIATOR

A character that was neither a decimal point nor a space nor alphanumeric appeared in column 1 of the input line.

INVALID OCTAL

The character string pointed to by the asterisk contains a character which is not in the set 0-7.

ILLEGAL OPERATOR

Something other than those statement operators appearing in this Appendix was the first nonblank symbol after column 1 (or after the label, if one exists).

NUMBER FROM I-245 EXPECTED

The indicated symbol is non-numeric, or if numeric, not in the specified range.

COMMA EXPECTED

The symbol after the first number in a DATA statement was not a comma.

FIELD2 IS LESS THAN FIELD1

In a DATA statement, the second field is less than the first.

LABEL REQUIRED

The DATA, REDEFINE and WORK statements all require a label.

DOUBLE QUOTE ASSUMED

A pre-defined Constant either in WORK or COMMON statements should

be terminated by a double quotation mark. If it is not there, it is assumed.

ILLEGAL LITERAL

In a table, every item enclosed in double quotation marks must be of equal length. Those that are of different length than the first item are flagged in error.

IMPROPER CONTINUATION

If a COMMON or WORK table is continued from a line, the following line must have a blank in column one, and the first symbol on the line must be a double quotation mark. If either of these is not the case, the continuation is an improper one.

UNDEFINED LABEL

A label is referenced which is neither one of the eight pre-defined labels, nor defined elsewhere in the program.

MISSPELLED WORD

A specific reserved word -- for example, the TO in an ADD statement -- has been misspelled. The misspelled word is assumed to be the one expected, and the next symbol is expected to be a legal label.

ILLEGAL CONDITION

The connective in an IF statement is not one of those listed in this Appendix. Nothing about the connective is assumed.

DUPLICATE LABEL

The label beginning on the line being listed, is duplicated previously in the program (or it is one of the eight pre-defined labels). The second and any subsequent definitions of the label are ignored.

MAXIMUM LABELS REACHED

The maximum number of labels allowed by the compiler is fixed at 125, including the pre-defined labels. All labels after this maximum is reached are ignored.

COMMON LIMIT EXCEEDED

The COMMON block may not exceed 100 bytes. Anything defined as COMMON after this length will not be accepted.

PROGRAM COUNTER ERROR

The program counter, at the end of pass two does not equal the program counter at the end of pass one. This is an internal compiler error message.

DISK DATAFORM COMMON SYSTEM ERRORS

FILE MISSING or FORM MISSING

The form number specified is not present as SYSNAM/DFF.
In the Interpreter, this message may mean that the next form specified in the current form's link, is not present, or that your command assumes that there is a form in memory and none is loaded.

NAME REQUIRED

The initial command line did not include the system name or form name required by the program.

ILLEGAL DEVICE SPECIFICATION

The initial command line included a disk drive specification which was improperly formatted.

BAD NUMBER

The form number may have been omitted, out of range (1-99) or non-numeric. Or, the form specified is not in the catalog. Note that if the form number is omitted in a command which optionally accepts form numbers (e.g., START n), the command line cannot end with a space.

APPENDIX D

THE THREE DATAFORMS

Efficient Programming Techniques

You are only given a certain amount of memory space to contain your program. If you've filled this space, or are coming close, use these helpful hints to make your program use less space.

1. Use carets (Λ) rather than underlines (_) for field definitions. The carets compress the unused spaces. See the DATAFORM Reference Manual for the DATAFORM you are using.

2. Place semi-colons at the end of all non-table, non-range variables to suppress the end-of-table character.

3. Use REDEFINE to create Constants or tables which are subsets of other Constants or tables. This technique may also be used for computation or hold areas if the redefined variables are not needed at the same time.

4. Use subroutines to perform repeated operations.

5. Use field displacement referencing to generalize programs used with line-items (i.e., where the same set of fields is entered several times within one form).

6. Use INPUT, OUTPUT and RESET to generalize programs and thus avoid duplication of code.

7. Keep Constants in the form itself by defining them at form generation time instead of using a field program to set them.

8. Combine several fields into one wherever possible (each field requires 6 additional bytes of edit table).

9. Use LOOKUP instead of CONVERT to save one of the tables.

10. Use data areas as work areas whenever possible, thus saving intermediate hold areas.

11. Execute all programs on last field if possible, on the assumption that the operator is usually right, to save NEXT and STORE instructions.

12. Don't use CHANGE/SHOW/CHANGE instructions since NEXT will automatically show and bypass if field is defined as "Program Reserved".

Form Generation Function Keys

There is a set of function keys available in the form generation mode only. When the DISPLAY key is pressed, certain characters become function keys. These function keys can all be found on the number pad. The following functions are available:

7 Character insert
8 Up cursor
9 Erase to end of frame
4 Left cursor
5 Character duplicate
6 Right cursor
1 Word remove
2 Down cursor
3 Form expand (downward)
0 Character remove
   Erase to end of line

The BACKSPACE key and DISPLAY/4 have the same function of non-destructive left cursor movement. Backspacing from column 1 back to column 80 of the previous line is permitted.

The CANCEL key erases the entire line the cursor is on and places the cursor at the beginning of the line.

The KEYBOARD key acts as a REPEAT key for all characters and for most function keys.

The DISPLAY/CANCEL function causes an edit table to be generated with the field position and length set and all edit conditions set to default values.

NUMBER PAD OVERLAY

Char Insert	UP	Erase Frame
LEFT	Dup Char	RIGHT
Word Remove	DOWN	Form Expand
Character Remove		Erase Line

Data Entry Interpreter Function Keys

Mode	Keys	Function
All Data Entry	DISPLAY/4	Return to Monitor
	DISPLAY/.	Write data record or rewrite it
	ENTER	Forward tab
	DISPLAY/3	Backward tab
	DISPLAY/5	Delete record (Diskette only)
	DISPLAY/6	Erase data area
	DISPLAY/1	Load next form
	DISPLAY/0	Duplicate field
MODIFY and FIND	DISPLAY/7	Rewind data file
	DISPLAY/8	Backspace record
	DISPLAY/9	Read record

Rewind Data File	BSP Record	Read Record
Return Monitor	Del Record	Erase Form Data
Load Form		BSP Field
Duplicate Field		Write Record

APPENDIX E

DUPLICATING DISKETTE AND CASSETTE FILES

You should regularly make backup copies of all your diskette files. You can use the COPY program to do this for you.

Load the complete DOS system diskette, just as you did to use the DOS EDITOR (for Level II DATAFORM). Instead of using the Editor, use the COPY program.

For example, to copy file SHOP01/DFF from Drive 0 to Drive 1, type

"COPY SHOP01/DFF:DR0,:DR1ↄ"

Cassette files are just as easy to duplicate, but it is done differently. To make a copy of the system tape with all catalogued forms, place a blank cassette in the front deck and the DF2SYS cassette in the rear deck and type "DUP ALLↄ".

APPENDIX F

EDITOR COMMANDS

The DOS Editor program enables you to create and modify files. All Editor commands are prefixed with a colon (:) to distinguish them from text lines. The pointer must be positioned at the line that needs correcting (use the KEYBOARD and DISPLAY keys to do this).

A full description of all Editor commands is in the DOS User's Guide. Here is an abbreviated list:

:D	Delete entire line
:D text	Delete all characters from the left edge of the line through and including the specified text.
:E	End of file. Write file to disk.
:E*	Display last line of file on screen.
:EO	Display data continuously on screen through last line of file.
:F text	Find line starting with text.
:I	Insert a line.
:L	Type next line in the file.
:L text	Find imbedded text.
:M old< new	Replace old text with new text.
:SC	Erase the lines form top of screen down to and including the pointed line.
:SB	Erase the lines through the bottom of the screen.

DATAPOINT CORPORATION

The Leader in Dispersed Data Processing

9725 Datapoint Drive
San Antonio. Texas 78284
512/690-7151