DynaForm

Release 1.1

A Document Formatting Program
for the
DynaStar Word Processing System

by

Allan G. Jost

DYNASOFT SYSTEMS LIMITED
P.O. Box 51
Windsor Junction, Nova Scotia
Canada B0N 2V0

This document and the DynaForm Program are protected under International Copyright laws and may not be reproduced in any form without the written permission of the copyright owner. Permission is hereby granted to bona fide purchasers to make two copies of the Program, in machine-readable form, for backup and archive purposes only.

Copyright (c) 1982, Dynasoft Systems Limited

TABLE OF CONTENTS

1.0 INTRODUCTION

DynaForm is a document formatting program for use with the DynaStar screen editor for general word processing applications. Its main function is the production of neatly formatted multi-page documents from text files prepared and edited by DynaStar. DynaForm provides the following facilities:

1) Automatic pagination
2) User specified header and footer lines with optional page numbers.
3) User specified unconditional and conditional page breaks.
4) Adjustable top, bottom, and left margins.
5) Adjustable page size and line spacing.
6) Overprint, boldface, double-strike, and underline.
7) Nested file inclusion to permit chaining of files.
8) User-defined macros with conditional command execution.
9) String substitution facilities to support "mail-merge" features for production of customized form letters.
10) String variable assignment from operator input and external data files.
11) Automatic generation of Index and Table of Contents.

DynaForm is <u>not</u> a general purpose text formatter in the traditional sense of the word, and it may not be entirely satisfactory for use with editors other than DynaStar. It is designed to be used in conjunction with DynaStar to provide a complete word processing function in a way that maximizes inter-active operator control of the appearance of the final document. In addressing this goal the DynaStar/DynaForm package deviates in fundamental ways from the design philosophy of traditional editor/text formatter combinations. The central difference lies in the "division of labour" between the "editor" and the "formatter" and the degree of operator interaction inherent in the process. To clearly illustrate this difference, we will start with a discussion of the evolution of computer-assisted document preparation:

Early text editors were written to run on large scale maxi- and mini-computer systems, and they reflected the limitations inherent in the input/output environment they were designed for. This environment was usually either a card reader/ line printer combination (perhaps with no editor at all), or the typical time sharing environment characterized by hundreds of slow speed computer terminals competing for time on a large mainframe computer. The terminals often did not even have lower-case capability (the card readers never did), and the operating speed was limited to 10 or 30 characters per second by the telephone ines which connected them to the central computer. For this eason, the editors used were almost always "line" (or "context") riented, and while some were quite powerful, the editing process as clumsy, and the only way to visualize the appearance of the ocument was to run off a listing. While this process could be sed to prepare presentable documents, it was awkward at best.

Eventually text-formatting programs were written to make it possible to generate neatly formatted documents without relying on the cumbersome editors to try to control the appearance of each paragraph and page. These formatters were very powerful and they could transform a rough looking sequence of words and lines into beautifully paginated, justified, indented paragraphs and pages.

The prevailing philosophy of these formatters was to treat text as a string of words, broken into phrases and sentences by punctuation marks, and into paragraphs and pages by special formatting commands. The words were automatically "filled" into lines and sometimes right-justified in the process. Indenting, "hanging" paragraphs, and special effects were handled by additional formatting commands. Although the input text for these programs was organized into lines for convenience in editing, the lines in the input text usually bore no resemblance to the lines in the final document. These systems worked well, but you never really knew what your text was actually going to look like until you "formatted" and printed it.

Today's microcomputer systems with their high speed video terminals make possible a much better way of producing attractive documents. The most important aspect of this is the power offered by terminals capable of operating at speeds of 1000 characters per second and higher, and displaying the full ASCII character set in both upper and lower case. This makes possible a much higher level of operator interaction while text is entered and edited. This has given rise to a whole new generation of "screen-oriented" editors such as DynaStar. With these editors you are no longer constrained to treating your text as a sequence of words to be edited a line at a time. Instead, your text can be viewed and edited in a form very close to its final appearance on paper: as a series of neatly formatted paragraphs forming a complete document. The video screen serves as a "window" through which you can view a substantial portion of your document at a time, and you can move forward and backward through the document with a couple of keystrokes. You can move your "cursor" in two dimensions through your text, and edit the material around the cursor by deleting, inserting, replacing and moving letters, words, and whole sections of text in a highly interactive manner. Most important, you can see and manipulate your text on the screen in a form which very closely resembles the way it will look on paper.

The DynaStar word processing package was designed to exploit this approach to a much higher degree than was possible only a few years ago. The DynaStar screen editor contains many features which were relegated to the <u>formatters</u> of yesterday: it automatically fills words into lines as you type them, ensuring that your text fits within the margins you have set, and it spreads out your text <u>on the screen</u> if you want a justified right margin. It lets you indent paragraphs as you go and it shows you what the paragraphs will look like. It lets you move the margins, or add words and sentences, and it re-adjusts your text to fit the margins, and then <u>it shows you what it will look like again</u>. There are no more "paragraph" and "indent" commands, and there is <u>no more guesswork</u>. <u>What you see is what you get</u>.

We are now in a position to explain the essential division of labour between the DynaStar editor and the DynaForm formatter. Early formatting programs had to do <u>all</u> the work, including both horizontal formatting (line filling, indentation, and justification) and vertical formatting (paragraph formation and pagination). DynaStar takes care of the horizontal formatting, interactively and on the screen, and it largely takes care of the paragraphs, too. It prepares a text file which is organized as a series of lines, but this time the lines are already formatted as they will print. In a fairly substantial way, about all that the DynaForm formatter has to do is organize your lines into pages and number them for you. For this reason, the largest group of formatting commands in DynaForm is in some way connected with the pagination process. DynaForm contains other facilities, such as the ability to generate customized form letters, but to begin with we will concentrate on pagination.

2.0 PAGE FORMATTING

If, DynaForm is asked to print a text file containing no special formatting instructions, it will make certain assumptions about the desired layout of your pages and produce a neatly paginated document, anyway (provided, of course, that its assumptions were correct!). It will assume, for instance, that you are using 8 1/2 by 11 inch paper and a printer that prints 6 lines per inch, which means that it is possible to fit exactly 66 lines on each page (perforation to perforation). This parameter is known as the page length, and it is measured in lines, not inches. It also assumes that you do not want to print right over the perforations, and it leaves margins at both the top and bottom of each page to avoid this. By default, it will leave 3 blank lines at the top of each page (the "top margin"), and 8 lines at the bottom (the "bottom margin"). The bottom margin will include a "footer" line, 2 lines (the "footer margin") below the last line of text on the page, containing the page number (starting at 1) in approximately the center. It will assume that y[illegible]r text was entered with its left margin in column 1, and it w[illegible]l shift everything on each page to the right 8 columns (the "page offset") since this will be required on most printers to leave an adequate left margin on the paper. All of this, of course, can be changed.

In general, DynaForm assumes that everything contained in a text file should be printed. However, if it finds a line which begins with a period, it will not print the line, interpreting it instead as a formatting command. All formatting commands begin with a period in column one. All other lines are printed.

Formatting commands consist of the opening period, a two letter code indicating the type of the command, and one or more numeric or textual parameters which may be required by the specific command. For instance, the command:

```
 .PL 88
```

w[illegible]ld set the page length to 88 lines. (Note that we had to indent this example by one column to prevent our own copy of DynaForm from actually interpreting it as a command.) Similarly, the command:

```
 .fo This will be printed at the bottom of the page!
```

will cause the line:

```
This will be printed at the bottom of the page!
```

to be printed at the bottom of each page, instead of the page number. Notice that the two letter command code can be typed in either upper or lower case.

With this simple introduction, we will now explain each of the DynaForm commands relating to pagination and other vertical formatting. These will be discussed in a logical order which facilitates the understanding of how they relate to each other. The same commands are summarized in alphabetical order in an appendix at the end of this document.

.BP n   <u>Begin Page (# n)</u>

This command forces DynaForm to start a new page. To do this it will first print enough blank lines to get to the bottom margin area, and will print the footer if one is currently defined. It will then print enough blank lines to position the paper at the top of the next sheet. The next line of actual text encountered will cause the printing of any header for the next page, and the spacing down to the bottom of the top margin area. If DynaForm is operating in single sheet mode, a message will be displayed on the console prompting the operator to change sheets. The command may be followed by a single numeric parameter which if supplied will set a new page number to be assigned to the new page (it must be between 1 and 10000). If no new page number is specified, the current page number is incremented by 1.

.CP n   <u>Conditional Page if less than n lines left</u>

This command forces Dynaform to start a new page <u>if there are less than n lines left on the current page</u>. The purpose of this command is to allow you to prevent the appearance of "orphan lines" at the top or bottom of pages, and to prevent tables, etc. from being broken across a page boundary. The parameter n should always be supplied, and will typically have a value of 3 or 4. These commands, when liberally applied at the top and near the bottom of paragraphs, will make it possible to paginate modified documents with impunity without worrying about "orphans". (An "orphan" is a line that lands all by itself at the top or bottom of a page.)

.PN n   <u>set Page Number to n</u>

This command requires a single numeric argument between 1 and 10000 and sets the current page number to the number specified.

.PL n   <u>set Page Length</u>

This command sets the line count corresponding to the total length of a sheet of paper, perforation to perforation. This number can be computed as: page-length = (lines-per-inch) * (length-of-form). The default value is 66 (6 lines/inch x 11 inches). The page length cannot be set to less than 1 plus the sum of the top and bottom margins, ensuring that each printed page will contain at least one line of text.

.HE text — define HEader

This command defines a header line which is printed automatically at the top of each page. The remainder of the command line, beginning with the first non-blank character after the ".HE", is used as the header string, with one exception: if a leading single or double quote is encountered, it is discarded and the header definition begins with the next character after the quote. The main purpose of this feature it to make it possible to define a header which begins with leading blanks (to indent it). In addition, if the symbol "#" is encountered in a header (or footer), it is replaced by the current page number in decimal. The header is considered to be part of the top margin. The default header is a blank line.

.FO text — define FOoter

This command defines a footer line in the same manner as the header line, as described above. The footer is printed within the bottom margin, at the bottom of each page printed. Leading quotes and the symbol "#" are treated as above. The default footer is 31 blanks and the symbol "#", which results in the printing of page numbers in the center of an otherwise blank footer line, (on the assumption that the normal width of a line is approximately 64 characters).

.MT n — set Margin at Top to n

This command sets the size of the top margin (the number of lines between the top of the page and the first line of text). This margin includes the header line and header margin, and defaults to 3 lines. If the top margin is set to 0, the header line is omitted.

.HM n	<u>set Header Margin to n</u>

This command sets the number of blank lines in the top margin between the header line and the text. The default value is 2, which normally results in the header being printed on the top line of the page.

.MB n	<u>set Margin at Bottom to n</u>

This command sets the size of the bottom margin (the number of lines between the last line of text and the physical bottom of the page). This margin includes the footer line and footer margin, and defaults to 8 lines. If the bottom margin is set to 0, the footer is omitted.

.FM n	<u>set Footer Margin to n</u>

This command sets the number of blank lines between the last line of text on a page and the footer line. It defaults to 2 lines.

.PO n	<u>set Page Offset to n</u>

This command sets the page offset. The page offset can be thought of a left margin setting. It is assumed that column 1 in your text file corresponds to the left margin of your page. Since most printers start printing near the <u>edge</u> of the paper, this in general would not leave an adequate left margin for most purposes. When a page offset is established, <u>everything printed</u> will be shifted right that number of columns (i.e. n blanks are appended to the beginning of each line). The default is 8 columns, which will usually leave a left margin of about an inch.

.MS n	<u>Multiple Space n lines</u>

This command is used to cause multiple spacing of your text. If the numeric argument n is set to 2 or greater, (n-1) blank lines will be printed between each text line in your document. For example, ".MS 2" will cause double spacing. If the number is omitted, it defaults to 2 (i.e. double spacing). Setting a value of 1 returns to single space mode (this is equivalent to .SS).

3.0 FILE INSERTION, MACROS, AND INDEX GENERATION

DynaForm has a facility for chaining multiple files together in a single document, a rudimentary macro definition facility, and a facility for the automatic generation of an Index and Table of Contents. These will be discussed one at a time.

3.1 File Insertion

File insertion involves only one command, which causes the contents of another file to be inserted into the current document as if the two files were a single file. This is very useful for a large document, allowing you to break up the document into smaller sections which are more convenient to edit. The usual approach would be to make each chapter or section a separate file, and have a "starter" file, perhaps containing a title page, which "includes" each of the constituent files in turn. If each of the sub-files is small enough to fit in the DynaStar edit buffer at once, editing will be especially convenient. It should also be noted that DynaStar cannot edit a single file larger than one half of the capacity of a disk because of the scratch file it must generate in the process. Breaking up a document into a series of smaller files will let you use much more of the capacity of a single disk. Under these circumstances, DynaStar will be able to function effectively as long as there is free space on a disk at least as large as the largest sub-file you wish to edit.

.FI pathlist <u>File Insert</u>

This command attempts to open the file specified and diverts input temporarily to this "inserted" file. When the end of the new file is reached, the file is closed and processing resumes with the next line in the original file. Inserted files can themselves contain .FI commands, to a "depth" of approximately 3. However, any number of .FI commands can be used in sequence in a single file. If an "insert" file cannot be opened, an error message is displayed on the console and processing resumes with the next line in the original file.

3.2 Macros

A Macro is a user-defined command which is "shorthand" for a sequence of commands and/or text lines. Macros can be used for a variety of purposes, depending on the imagination of the user. It would be possible, for example, to define a macro for starting a new section or chapter which sets up the chapter heading and number, forces a new page, establishes a new header and footer, builds the entry for the Table of Contents, prints the chapter heading, and spaces down to the position of the first line of the new material. Such a macro was used in the production of this document.

Macros can contain any command recognized by DynaForm, including references to (but not definitions of) other macros. They can also contain lines of text to be printed. When macros are defined, they are stored in a "macro pool area" which has a maximum size dependent on the amount of memory given DynaForm when it is called. With the default memory allocation of 8k bytes, approximately 6000 bytes is set aside for the macro pool, which is shared with string variables (see the mail-merge facility) and index entries. If the macro pool is filled, DynaForm displays an error message on the console and then tries to carry on processing. If this happens, the document should be re-printed giving DynaForm a larger work area.

Macros may contain string variable references and print control characters. If included, these items are processed at the time the macro is <u>used</u>, not when it is defined.

.MA xx	<u>define MAcro "xx"</u> This command opens a macro definition for a macro to be called ".xx", where xx is any two character sequence. All input lines between the .MA command and the next .ME command encountered (see below) are saved in the macro pool under the name ".xx". Subsequent to this definition, use of the command .xx will cause insertion and processing of the entire macro. "xx" may be the name of a standard DynaForm command, in which case it redefines the command, or it may be a new name. In the name "xx", it does not matter if the letters are in lower or upper case, and the name may be longer than two characters, although only the first two characters are used.
.ME	<u>define Macro End</u> The command marks the end of a macro definition.

.XX Execute Macro .XX

XX is any two-character name which has been defined as a macro. The contents of the macro is inserted in the document in the place of the .xx command, and is processed as if it had been entered directly, including any string variable references or print control characters. Note that macros may contain references to other macros.

.IFE yy IF Even page number do .yy

This command causes conditional execution of the command .yy, if the current page number is even. yy may be a standard command or a macro reference. The IF command may be used at any time, including within a macro.

.IFO yy IF Odd page number do .yy

This command causes conditional execution of the command .yy if the current page number is odd.

3.3 Index Generation

DynaForm contains a facility for automatically generating indexes in either page number or alphabetical order. Multiple indexes may be built, using what is know as a "tag" character for each index. Index entries are built by including the .DX command at appropriate points within a document, and printed out later using the .XA and .XN commands. The .DX command includes a string (similar to the string supplied with .HE and .FO commands) which is "remembered" in the macro pool area, along with the page number on which the index entry was found and the "tag". The index is printed out, usually at the end of the document, using the .XA command (for an alphabetical index) or the .XN command (for an index in page number order, usually for the Table of Contents). At this time, all entries in the pool are sorted in the desired order, and then all entries with the right "tag" are printed one per line, along with the page number on which the entry was defined. Up to 400 index entries may be built in a single document, subject to space availability in the macro pool.

.DXt text define inDeX entry with tag "t"

This command enters the string "text" in the macro pool, along with the current page number and the "tag" character t. The tag character may be any character, including blank. The string "text" is processed in the same way as the string in HEader and FOoter commands, beginning with the first non-blank character after the first blank after the "DX". Up to 400 .DX commands may be used within a single document.

.XAt n

print inDeX for tag "t" in Alphabetic order

This command sorts all current index entries in alphabetical order (ignoring case), and then prints all entries which have the tag "t", one per line, at the left margin, followed by the page number of the page on which the corresponding .DX command was encountered. If a numeric argument "n" is supplied, the page number is printed in column n if possible. If the column number is omitted, or if the index entry itself extends beyond the specified column, the page number is printed immediately after the index text, separated by one blank. Normal page formatting is still in effect during the printing of an index. It is up to the user to provide any headings required.

.XNt n

print inDeX for tag "t" in Numeric order

This command is similar to the .XA command above, except that the index pool is first sorted in page number order prior to printing the index. If several entries in the index have the same page number, they will themselves be sorted in alphabetic order. This command will normally be used to generate a Table of Contents. It is generally a good idea to print any numeric indexes prior to alphabetic ones to avoid possible "mixing up" of entries which may have the same page number. Another technique (used in this manual) is to start Table of Contents entries with a section number, so that the "tie breaking" alphabetic sub-sort will guarantee that the entries will print in the correct order.

[illegible]BLES AND MAIL-MERGE

String variables are the key to one of the most powerful capabilities of DynaForm: the ability to create "customized" form letters and other repetitive documents such as legal contracts. Basically, the idea is to create a "standard" document file which is complete except for a small number of "blanks" which must still be filled in, and then "fill in the blanks" with data from a special file (or with special commands) at the time the document is actually printed. The <u>data file</u> is usually a mailing list consisting of a series of "records" containing information such as first and last name, mailing address, etc. The process is best explained by giving an example.

Suppose we have a <u>data file</u> containing the following lines:

```
\Mr. John Doe
Mammoth Corporation
123 Anystreet
Anytown, Anystate
12345
John
\Mrs. Joanne Peabody
[illegible]abody, Peabody, and Peabody
P.O. Box 432
Boston, Massachusetts
01234
Joanne
```

This file contains two <u>records</u>, each containing six lines, - <u>items</u>. The first line of each record starts with a special character, which in this case is a backslash ("\"). This special character is the <u>record delimiter</u>, and it is chosen by the user:)ynaForm reads <u>the first character in the file</u>, and uses this character as the delimiter for the rest of the file. The elimiter is used to help DynaForm "keep in synch" with the file.

The six "items" in each record will be assigned names for se in the document in which they will be used, but for now we ll simply describe them. The first item is always the full me of an individual, the second is the name of a company, the xt three items constitute the mailing address, and the last em [illegible]s the usual first name of the individual addressed.

We will now compose a short document, consisting of a form ter to be sent to each person/company in the data file. The ter will be written in a general fashion, using "string ables" at any place requiring specific information about the essee. String variables can be recognized by the fact that always begin with the symbol "<" and end with the symbol

```
..This is the standard "polite reminder" letter
.fo
.df late.list
.rv full-name,company,street,town,zip,first-name
.pv balance,Enter account balance for <full-name>
May 27, 1982


<full-name>
<company>
<street>
<town>
<zip>


Dear <first-name>,

In checking our credit records, it has come to our attention that
your account has recently fallen into arrears.  We know, <first-name>,
that you would like to maintain your usually good credit rating
with us, and hope that this has merely been an oversight on your
part.

Your current balance is $<balance>.  Please take a moment and put
your check for this amount in the mail today.


Yours truly,



William (Bill) Collector
Credit Manager
```

This form letter uses most of the mail-merge features of DynaForm. We will explain them in the order they appear. The first line is simply a comment line which is not printed because it starts with two periods. The second line (.fo) defines an empty page footer to prevent DynaForm from printing a page number at the bottom of each letter. The third line defines the _data file_ to be used to "fill in" the string variables. It is assumed that the data file has been saved under the name "late.list" in the current working directory (a full pathlist could have been used instead).

The next line in the document is key: it is a "read variables" command which causes DynaForm to read the next _record_ from the data file, and assign each of the _items_ in the record in turn to the _string variables_ "full-name", "company", "street", etc. Notice that the string variables are listed on the command line _without_ the standard delimiters "<" and ">". After processing all items listed in the command, DynaForm skips forward if necessary in the data file to the beginning of the next _record_, recognized by the next appearance of the _record delimiter_.

Now we see the first actual use of a string variable, which in this case happens to be within yet another DynaForm command. We have a .pv (Prompt-for-Variable) command which itself contains a string reference. The Prompt-for-Variable command lists a single string variable (in this case "balance"), followed by a comma and a prompt string. The prompt string in this case contains a string reference to the string variable <full-name>, which as it happens was defined in the Read-Variables command above. This prompt string is displayed on the system console, including the appropriate string substitution, so that the first time through the actual prompt will be:

Enter account balance for Mr. John Doe

At this point the program will wait for the operator to type something on the console. It expects a string of characters followed by a carriage return, and in this case it should be a number such as "123.45". This string will be assigned to the string variable <balance>. Now finally DynaForm will print the letter. Since we have tried to pick obvious names for each of the variables used in the letter, we will leave it as a mental exercise to visualize what the letter will actually look like. It might be a good idea, for that matter, to actually try the example on your system to see how it _really_ works. As a final point, note that the document will be repeated _automatically_ if a data file is open and has not yet reached end-of-file.

With this example for background, we can now give a more concise description of the string variable/mail merge facility of DynaForm:

A _string variable reference_ is any string of up to 40 characters beginning with "<" and ending with ">". Whenever DynaForm encounters a string variable reference in a document file, it checks to see if a _string variable_ has been defined with the same name. If a matching string variable is found, its current value (a string of characters) is _substituted_ for the string variable reference. If no such defined variable is found, the string variable reference is printed "as-is" without any change. (This permits the symbols "<" and ">" to be used within a document, as long as there is no confusion with currently defined string variables.) It should be noted that DynaForm will not re-justify a line containing a string substitution, so this feature is best used in text formatted with a _ragged_ right margin.

String variables are defined with one of the DynaForm commands .sv (Set Variable), .rv (Read Variables), or .pv (Prompt for Variable), described below.

.DF pathlist <u>open Data File</u>

This command opens the file described by "pathlist" and establishes it as the "data file" for this run. String variables can then be defined by reading from the data file with the .RV command (see below). At the time the file is opened, the first character in the file is read and used thereafter as the "record delimiter" character. This character should be a "special" character such as the backslash which will not be used for any other purpose in the data file. Whenever a data file is open and has not yet reached end-of-file, DynaForm's <u>auto-repeat</u> mode is enabled. If auto-repeat is enabled and end-of-file is reached in the document file, the document file is automatically rewound, starting a new copy of the same document from the beginning. Since the next pass through the document will again encounter the .DF command, DynaForm ignores .DF commands after the first.

.RV name1,name2,name3,....,name-n
<u>Read Variables <name1>, <name2>, etc.</u>

This command reads the next n <u>items</u> from the current data file, and assigns them to the <u>string variables</u> <name1>, <name2>,....,<name-n>. It then skips forward if necessary in the data file to the beginning of the next <u>record</u>, marked by the <u>record delimiter</u> character. If the end of the data file is reached, the file is closed, and <u>auto-repeat</u> mode is turned off.

.SV name,text <u>Set Variable <name></u>

This command assigns the string "text" to the string variable <name>. "Text" consists of all characters between the comma and the end of the line.

.PV name,text <u>Prompt for Variable <name></u>

This command displays the string "text" on the user's terminal and waits for the operator to type an answer string. The reply typed by the operator is assigned to the string variable <name>, not including the carriage return which marks the end of the reply.

.DM text   <u>Display Message</u>

This command displays the string "text" on the user's terminal as a message to the operator.

5.0 PRINT CONTROL CHARACTERS

DynaForm processes certain special character sequences generated by DynaStar to produce special effects such as underline and boldface. The OverPrint command, which also produces a special effect has already been described.

Control characters are recognized by DynaForm as a two character sequence consisting of a special form of the character "^" (with the high order bit set) and a capital letter. If the combination is one that DynaForm recognizes as a command for a special effect, it is processed as such. If the combination is not one of DynaForm's standard set, it is converted to an ASCII control character and sent to the printer. This facility can be used to take advantage of special features of some printers.

DynaForm's special effect control sequences are "toggle" commands: they alternately turn a special effect on or off. The effects can span only a few words or many lines of text, and the effects can be combined.

Currently, there are only three special-effect control sequences implemented by DynaForm:

^B	Boldface	Turns <u>boldface</u> on or off. Any text between a pair of these sequences will be printed in boldface. In the current version of DynaForm, this is implemented by triple-striking each character.
^D	Double-strike	Turns <u>double-strike</u> on or off. Any characters between a pair of these sequences will be struck twice.
^U	Underline	Turns <u>underline</u> on or off. Text between a pair of these sequences is underlined. Only non-blank characters are actually underlined.

6.0 RUNNING DYNAFORM

DynaForm is run by using the "P" command from the files menu of DynaStar (OS9 version only), or by typing a command line of the form:

OS9:

```
  Df pathlist [ first-page ] [ last-page ] [ :S ] [ pathlist ]
```

FLEX:

```
  Df filespec [ first-page ] [ last-page ] [ :S ] [ * ]
```

The square brackets delimit optional parameters. The mandatory pathlist/filespec at the beginning of the command is the file containing the document to be formatted.

The optional parameters "first-page" and "last-page" specify the first and last page numbers to be printed in the current run. This feature makes it possible to print only a selected section when minor changes have been made to a document which has already been printed once. These parameters must be positive decimal numbers. If they are omitted, the entire document is printed, and if only one of them is included, it is assumed to be the parameter "first-page" and all pages from that page to the end are printed.

The optional parameter ":S" specifies "single sheet" mode. If single sheet mode is specified, DynaForm pauses (with a message to the operator) after each page for the operator to insert a new sheet in the printer. The operator then types any character on the console to resume printing the next page. If single sheet mode is not specified, Dynaform assumes that continuous "fan-fold" paper is being used, and it prints the entire document without pausing.

Under OS9 only, if a second pathlist is specified at the end of the command, it is used instead of the system printer as the "printed" output path. This can be an alternate printer, the console ("/TERM"), or a disk file for later printing. If an alternate path is not specified, DynaForm routes its primary output to the standard device "/P". If DynaForm is unable to open a path to its specified output destination, a message is displayed on the console, and output is then directed to the console.

Under FLEX, if the last parameter on the command line is an asterisk ("*"), formatted output is directed to the console terminal. This is useful for previewing the document, or in conjunction with the FLEX "O" command to spool output to a file for later printing. If the asterisk is omitted, formatted output is directed to the system printer using the standard "print.sys" driver.

Under OS9, DynaForm normally requests 8K bytes of working memory, and this will be sufficient for most print jobs. However, documents making extensive use of macros or the indexing facility may run out of space in the string (or macro) pool. If this happens, you should give DynaForm a larger working memory by using the "#" option on the shell command line.

Under FLEX, DynaForm allocates approximately 10k bytes of working memory for file buffers and file control blocks, and the balance of available memory is allocated to the macro pool.

OS9 users should note that if output is routed to a <u>file</u> instead of a printer, the file produced will not be in the form of a standard text file, but will instead be an image of what would normally be sent to a printer. This means in particular that there will be <u>line</u> <u>feeds</u> in the file in addition to the normal <u>carriage</u> <u>returns</u> which signify end-of-line in a standard text file. If the OS-9 "LIST" command is used to list or print this file, it will appear to be double-spaced. If you wish to print a file produced in this way (for spooling purposes) you should use the "COPY" command instead, as follows:

```
Copy pathlist /P
```

It is worth noting that when using the "COPY" utility to print formatted files, you can minimize disk head loads by giving "COPY" a large working memory size.

6.1 Installation

The OS9 version of DynaForm consists of two files, both of which must be installed in the execution directory before use. The first is a file called "DF" which is the actual DynaForm program. The second is called "PINTERP" and is the p-code interpreter for Dynasoft Pascal (DynaForm is written in Dynasoft Pascal). When DF is run, it automatically loads and links to the interpreter as part of its initialization sequence.

The FLEX version of DynaForm consists of three files, all of which must be installed on the system disk. The file called "DYNAFORM.SYS" is the formatter itself, "INTERP15.SYS" is the p-code interpreter for Dynasoft Pascal, and "DF.CMD" is the command file executed by the user to load the other two files and run the program.

SUMMARY OF COMMANDS

COMMAND		page
.. text	Comment line	8
.BP n	Begin Page #n	5
.CP n	Conditional Page	5
.DF pathlist	open Data File for mail-merge	16
.DM message	Display Message	17
.DXt text	inDeX entry (t=tag)	11
.FI pathlist	File Insert	9
.FM n	Footer Margin [default 2]	7
.FO text	define FOoter line	6
.HE text	define HEader line	6
.HM n	Header Margin [default 2]	7
.IFE yy	IF Even page do .yy	11
.IFO yy	IF Odd page do .yy	11
.IG	IGnore to next "dot"	8
.MA xx	start MAcro .xx	10
.MB n	Margin at Bottom [default 8]	7
.ME	Macro End	10
.MS n	Multiple Space [default 2]	7
.MT n	Margin at Top [default 3]	6
.OP n	OverPrint n lines [default 2]	8
.PL n	set Page Length [default 66]	5
.PN n	set Page Number	5
.PO n	Page Offset [default 8]	7
.PV name,text	Prompt for Variable <name>	16
.RV name1,name2,...	Read Variables <name1>, etc.	16
.SP n	SPace n lines [default 1]	8
.SS	Single Space	8
.SV name,text	Set Variable <name>	16
.XAt n	print indeX Alphabetic, tag t	12
.XNt n	print indeX Numeric, tag t	12
.XX	execute macro .xx	11
^B	Boldface on/off	18
^D	Double-strike on/off	18
^U	Underline on/off	18