PRIME

Initial Documentation Release

IDR 3040

PRIME FORMS MANAGEMENT SYSTEM (FORMS)

This IDR is the first release of information on the Prime Forms Management System (FORMS) as distributed on Master Software Disk Revision 13.

Revision Ø June 1977

IDR 3040

PRIME FORMS MANAGEMENT SYSTEM (FORMS)

PRIME
Computer, Inc.

145 Pennsylvania Ave.
Framingham, Mass. 01701

CONTENTS

FOREWORD

The Prime Forms Management System (herein referred to as FORMS) provides a convenient and natural method of defining a form in a language designed for such a purpose. These forms may then be implemented by any applications program which uses Prime's Input-Output Control System (IOCS), including programs written in COBOL, FORTRAN, RPG-II, and PMA. Applications programs communicate with FORMS through input/output statements native to the host language. Programs that currently run in an interactive mode can easily be converted to use FORMS.

FORMS allows cataloging and maintenance of form definitions available within the computer system. To facilitate use within an applications program, all form definitions reside within a centralized directory in the system. This directory, under control of the system administrator, may be easily changed, allowing the addition, modification, or deletion of form definitions.

FORMS is device independent. There may be any mix of terminals attached to the Prime computer which, so long as they contain some basic properties discussed later in this document, may be used with the FORMS system. Terminal configuration is governed by a control file in the centralized FORMS directory. This file is read by FORMS at run-time to determine which device driver to use, depending on this user's terminal type. This means that multiple terminal types may be driven by the same applications program without change. Certain terminal types are supported by FORMS as released by Prime. Should the user have another terminal capable of supporting FORMS, all that he need do is write a low-level device driver for the terminal and incorporate it into the FORMS run-time library.

RELATED DOCUMENTS

The following listed documents may provide useful supplemental information to the reader of this User Guide.

Document Title	Order No.
PRIMOS Interactive User Guide	MAN 2602
Primos File System User Guide	MAN 2604
Program Development Software User Guide	MAN 1879

SECTION 1

SYSTEM CONCEPTS

1.1 Form Definition

All applications program/FORMS data transfer is handled through input/output statements in the host language. This means that the user must "tell" FORMS the format of the input/output data that is going to be transferred from and to the applications program. This definition, known as the data stream descriptor, serves two primary functions. First, it defines the location of each data item, called a field, within the input/output record(s). Second, it defines the length of each field. Optionally, it can include justification and validation parameters.

The programmer must then define the format of the data as it is to appear on the input/output device. This definition, known as the device format descriptor, describes the position of the data in terms of line/column coordinates, field length, justification, and optional display attributes. Display attributes include:

. write enable/write protect	(terminal)
. blink	(terminal)
. reverse video	(terminal)
. not displayed	(terminal/hard-copy)
. underline	(hard-copy)

The data stream descriptor serves as an interlude between the applications program and the device format descriptor. One stream descriptor is mapped to one format descriptor, which may contain format definitions for several devices (e.g., a terminal and the line printer). When requesting a certain form, the applications program names (in a FORMS command) the data stream descriptor, which in turn contains the name of the format descriptor to be used. This run-time binding among the applications program, data stream, and format descriptors allows the user to modify one of the descriptors without necessarily having to change, recompile, or reload the applications program or the other descriptor. For example, should the user wish to change the position of a field in the device format, he need recompile only the device format and replace the existing descriptor with the new one in the FORMS catalog.

Within a device format descriptor, each field which is to contain data from the applications program is assigned a unique 1-8 character name. Stream descriptor fields map data from the applications program to these fields. The format descriptor maps the data received from the stream descriptor to a defined position on the input/output device.

A block diagram showing data flow from the applications program to the input device might look like this:

```
 _____________         _____________          _____________
|             | input/ |             | field1 |             |   to
|applications| output |   stream    | field2 |   format    | device
|             | record |             | field3 |             |-------->
|   program   |        | descriptor  | fieldn | descriptor  |
|_____________|        |_____________|        |_____________|
```

To summarize, the data stream descriptor separates the user's input/output record into fields, which are then mapped onto fields in the device format descriptor. The device format descriptor defines the location, length, and attributes of the data on the input/output device. It is interesting to note that on input the same mapping is used as for output - from the stream descriptor to the format descriptor. No "backwards" mapping exists; the data flow is simply reversed.

1.2 Applications Programming with FORMS

A unique advantage of FORMS over other transaction-oriented processing systems is that it allows the user to program in standard languages such as COBOL and FORTRAN; no non-standard processing language exists. Commands are issued to the FORMS run-time package as output records from the applications program. Until a FORMS command is issued, all input/output is processed normally, using the standard I/O drivers resident in the Fortran library (FTNLIB). When a FORMS command is issued, it is trapped and processed by the run-time package. The run-time package also traps and processes all data I/O requests from the applications program when a form is being processed (is invoked).

The method of data transfer using FORMS is no different from that used with normal input/output. In FORTRAN, for example, one would still write to the terminal (logical unit 1) or line printer (logical unit 4) according to a FORMAT statement. In COBOL, a file would be selected and assigned to the TERMINAL for a user-terminal form. Because of compatibility problems between FORMS and COBOL, a new device type, called OFFLINE-PRINTER, was added to print a form on the line printer; this device type is now valid in a SELECT/ASSIGN statement.

When designing the input and output record formats for the applications program (and hence, the stream descriptor), the user must be careful not to exceed the default FORTRAN I/O buffer size of 132 characters. Should he choose to define his own I/O buffer and reset the record-size table entry for a particular device (namely, the terminal or printer) with the ATTDEV subroutine, he is warned that the absolute maximum record size that FORMS is able to handle is 160 characters. If the form contains more than 132 or 160 characters, or if the programmer deems it convenient to break the data up into smaller records, a facility exists for writing or reading the form in "pieces". This facility, known as a substream definition, allows the programmer to transfer the form to and from the applications program in multiple records. It is described in detail later in this document.

SECTION 2

FORMS SYSTEM COMPONENTS

2.1 Forms Definition Language

The Forms Definition Language (FDL) translates the user's form definition files into binary control block files. Form definition files, which are prepared with the Prime text editor (ED), describe the form in the FDL language described later in this document. The binary control block files emitted by FDL are placed in the central FORMS directory by the system administrator using the Forms Administrative Processor. These definitions are now available for use in an applications program. FDL optionally emits a listing file containing the user's source text listing, error codes and explanations (if any), and data formats as FDL perceives (from the user's form definition) they should "look" at run-time.

FDL statements, syntax, and usage are described later in this document.

2.2 Forms Administrative Processor

The Forms Administrative Package (FAP) is a utility program which allows the system administrator to manipulate and catalog the forms definitions within the system. Also, it allows him to change the terminal configuration tables when terminals are added to/removed from the system. The basic functions provided by FAP are:

- o Create FORMS directory
- o Add/replace form descriptions
- o Purge form descriptions
- o List catalog of form descriptions
- o Modify/list terminal configuration table

FAP commands and syntax are described later in this document.

2.3 FORMS Run-time Package

The FORMS Run-time Package is a collection of subroutines which is loaded as a library with the applications program. These subroutines perform all run-time functions, including forms lookup, buffer management, data manipulation, and input-output control.

The FORMS Run-Time Package must be loaded immediately before the FORTRAN library, as it contains replacements for some IOCS tables defined in FTNLIB (and VFTNLB). A typical load sequence for a COBOL program which uses FORMS and KI/DA might look like this:

```
OK, HILOAD
GO
$ COMMON 160000             (load common below 160000)
$ LOAD B_USRPRG             (user's COBOL program)
$ AU 40                     (use automatic Linkbase generation)
$ LIBRARY COBKID            (COBOL library with KI/DA)
$ LIBRARY RFORMS            (64R mode FORMS library)
$ LIBRARY                   (FORTRAN library)
LC
$ SAVE *USRPRG
$ QUIT

OK,    .
       .
       .
```

Two FORMS libraries exist: one for 64R mode (called "RFORMS"), and one for 64V mode (called "VFORMS"). In addition, a template command file exists for building a shared-procedure version of FORMS (for the Prime 400 only). These are discussed in detail later in the document.

All FORMS Run-Time Package libraries use COMMON initialization via FORTRAN "BLOCK DATA". This requires the user to place the COMMON area out of the area used by the loader, where it normally resides, when loading the library in 64R mode. Note that this does not hold true in 64V mode, as COMMON does not overlay the segmented loader (SEG).

Although it resides in the library as one file, the run-time package is made up of three separate parts, described below.

2.3.1 Run-Time Package Proper

The Run-Time Package Proper is the control portion of the run-time package. It processes all FORMS commands issued by the applications program, as well as all input/output data. This package contains the two routines that directly control the input and output of forms, as well as routines for form initialization, data transfer, text justification, validation, etc.

2.3.2 Buffer Pool Manager

The Buffer Pool Manager controls all form lookup and retrieval at run-time. This package is called by the run-time package proper to perform three functions: form lookup, form definition retrieval, and form termination. This package controls a buffer pool capable of holding multiple form definitions in either a single-user or shared-procedure environment. When a form definition is requested and the buffer pool becomes full, the pool manager writes least recently used form definitions to a temporary file and reads in the requested form. This "rollout" function is performed to prevent having to lookup the form definition again, which could be a lengthy process if there are several hundred forms in the system.

2.3.3 Input-Output System

The Input-Output System (IOS) is the collection of device drivers which control all devices supported by the FORMS system. Any user-written device drivers are incorporated into this package. The IOS also contains replacements for the write-ASCII and read-ASCII tables (WATBL and RATBL) in the Fortran Library IOCS. These tables cause calls to the write-ASCII (WRASC) subroutine for the terminal and line-printer and call to the read-ASCII subroutine (RDASC) for the terminal to be trapped and processed by a high-level FORMS subroutine. If a form is invoked on the given device, the data is processed by FORMS; if not, the call is reflected by FORMS to the standard I/O driver for the device (O$AA01, I$AA12, and O$AL06/04, respectively). Note that all input/output (i.e., READ and WRITE) statements in FORTRAN, COBOL, and RPG-II generate calls to RDASC and WRASC, respectively.

SECTION 3

FORMS DEFINITION LANGUAGE

3.1 General Syntax

FDL supports a free-format input line, much like the Prime Macro Assembler (PMA).

All form descriptor, substream, and field names must start in the first character position of the line and must be followed by at least one space. Descriptor statements may start anywhere after column 1, and may occupy columns 2 through 72. Columns 73 through 80 are ignored. Items in the input line may be separated by either a space or a comma, unless otherwise noted. Lower case characters are mapped to upper case, with the exception of characters in a text string (enclosed within single quotes).

Should an input record contain too many characters to fit on one line, the programmer may continue his source text by placing a semicolon (;) as the last character of the line to be continued. Note that input items (words, text strings, etc.) may not be split across two lines. There is no limit to the number of continuation lines in a source record; there is, however, a 240-character limit per record.

If the first character of a line is an asterisk, that line is treated as a comment, listed in the output file and ignored. If the first character is a single quote (´), the line is treated as a comment, but causes an eject in the listing and becomes the new page header.

In addition to full line comments (lines beginning with an asterisk or single quote), in-line comments are also supported. In-line comments are preceded by a fore-slash and asterisk (´/*´) and followed by an asterisk and a fore-slash (´*/´). Should the programmer place the in-line comment as the last item on the line, the terminating characters (´*/´) may be omitted. Note that, unlike FORTRAN, in-line comments may not occur within an item (eg, in the middle of a name or text string).

Examples:

```
* THIS IS A COMMENT LINE

' THIS WILL CAUSE A PAGE-EJECT AND WILL BECOME THE NEW HEADER

LABEL   FIELD ABC, LENGTH 6     /* THIS IS AN IN-LINE COMMENT

LABEL   FIELD ABC, /*THIS TOO IS AN IN-LINE COMMENT*/ LENGTH 6

NAME    FIELD 'FOUR SCORE AND SEVEN YEARS AGO... ' ;
        POSITION (10,10) PROTECT /* CONTINUATION LINE
```

3.2 Naming Conventions

The rules for naming form descriptors, fields, substreams, etc. are as follows:

- o Name length: 1-8 characters
- o First character must be alphabetic
- o Permitted characters: A-Z, 0-9

Examples:

Permitted	Not Permitted	Why
SHIPFORM	GAZORKLEFORM	name too long
FORM5	5FORM	bad 1st character (5)
AMTOWED	OWED$	illegal character ($)

3.3 Form Description Structure

The following diagrams represent the various form definition structures for both the stream descriptor and format desciptor.

<u>Stream Descriptor</u>

```
 Stream descriptor header              Stream descriptor header
           .                         Substream descriptor header
           .                                    .
           .                                    .
   Field definitions          -or-              .
           .                             Field definitions
           .                                    .
           .                                    .
Stream descriptor terminator                    .
                                    Substream descriptor terminator
                                      Substream descriptor header
                                                .
                                                .
                                                .
                                         Field definitions
                                                .
                                                .
                                                .
                                    Substream descriptor terminator
                                      Stream descriptor terminator
```

<u>Format Descriptor</u>

```
 Format descriptor header
  Device block 1 header
           .
           .
           .
    Field definitions
           .
           .
           .
 Device block 1 terminator
   Device block n header
           .
           .
           .
    Field definitions
           .
           .
           .
 Device block n terminator
Format descriptor terminator
```

3.4 Descriptor/Block Delimiter Statements

3.4.1 The STREAM Statement

This statement is used to define the beginning of a data stream descriptor control block. The name of the stream descriptor must appear starting in the left margin (column 1) and must conform to the rules specified above.

3.4.1.1 FORMAT Parameter (optional)

This parameter names the FORMAT control block to be used with this stream descriptor. If it is omitted, the FORMAT descriptor is assumed to be the same name as the STREAM descriptor. Note that two modules by the same 8-character name but of different types (such as a stream and format descriptor) may coexist in the FORMS catalog.

Examples:

```
TAXFORM   STREAM

SHIPFORM  STREAM, FORMAT SCREEN1
```

3.4.2 The END STREAM Statement

This statement terminates a data stream descriptor. It must be the last statement in the stream descriptor.

Example:

```
          END STREAM
```

3.4.3 The SUBSTREAM Statement

This statement defines one input or output record within a stream definition. It gives the capability of multiple input and/or output records (streams) under one stream descriptor. An optional name is permitted starting in the left margin (column 1). If present, the applications program can position directly to this substream instead of sequentially reading or writing multiple records to access or display a certain piece of data (see run-time package command description).

If this feature is to be used, no field definitions may reside outside of a substream block. Any fields that do are flagged as errors by FDL.

Examples:

```
          SUBSTREAM

     INPDATA   SUBSTREAM
```

3.4.4 The END SUBSTREAM Statement

This statement is used to define the end of a substream block. For each SUBSTREAM statement, there must be a corresponding END SUBSTREAM statement. Substreams cannot be nested.

Example:

```
          END SUBSTREAM
```

3.4.5 The FORMAT Statement

This statement defines the beginning of a format descriptor control block. The name of the format descriptor must appear starting at the left margin (column 1). Note that this does not start a device format descriptor; it only identifies the format for which the succeding device descriptors will be included. This statement must be followed by one or more device descriptors (see below).

Example:

```
     ORDENTRY  FORMAT
```

3.4.6 The END FORMAT Statement

This statement terminates a format descriptor. It must be the last statement in the descriptor.

Example:

```
          END FORMAT
```

3.4.7 The DEVICE Statement

This statement defines the beginning of a device descriptor. It must occur within a a format descriptor, but outside of any other device descriptor (i.e., after a FORMAT or END DEVICE statement), and must be followed by a 1-8 character device name.

All hard-copy devices (except user terminals) have hard-wired device names. At this release, the only hard-wired device name is "PRINTER", for the off-line (spooled) line printer. All device names for terminals are installation-dependent.

Examples:

```
DEVICE VISTAR3

DEVICE PRINTER
```

3.4.8 The END DEVICE Statement

This statement terminates a device descriptor block. It must be the last statement in the device descriptor and must occur before an END FORMAT or another DEVICE statement.

Example:

```
END DEVICE
```

3.5 Stream Descriptor FIELD Statement

The FIELD statement in the stream descriptor serves two primary purposes. First, as previously discussed, it defines the position of the field and its length in the user's input and/or output record. Secondly, it maps this field onto a field in the corresponding device format descriptor.

The FIELD statement also can perform several supplemental functions. Instead of processing data from an applications program output record, fields can output predefined (literal) data or special system information data (such as time, date, user-name, etc.), which is processed entirely by the FORMS run-time package. On input, not only can they return data from the input device, but also predefined (literal) data, or a literal string in the event of a device field being empty.

Should the programmer desire, he can declare certain fields to be input only and/or certain fields to be output only (i.e., fields that occur in either the input or output record, but not both). In most cases, the default is input-output.

Every direct, output-literal, or input/empty-conditional field (see below) has a name associated with it. Normally, this is the name of them mapped-to field in the device format descriptor. The programmer can, however, override this by placing a name starting in the left margin of the FIELD statement. Display attributes of a field may be modified by the applications program at run-time by referring to the field by its stream descriptor field name in a FORMS command. Note that the applications program <u>never</u> "talks" to the device format descriptor directly; it <u>always</u> communicates through the stream descriptor. This makes the device format fully invisible and therefore fully independent of the applications program.

The statement must be followed by a mapping or literal specification. The following is a list of possible specification types:

<u>Direct</u>: the most common specification. It maps (1:1) the stream descriptor field to a named device format descriptor field.

Specification:

```
FIELD device-field-name
```

<u>Input-Literal</u>: identifies the field to be input-only (it is ignored on an output request). The literal data is returned to the applications program as part of the input record, just as though it had been read from the device. A name is not permitted on a field with input-literal specification; it does not map to a device format field and therefore has no attributes which can be modified.

Specification:

```
FIELD 'text string'
```

<u>Output-Literal</u>: an output-literal field maps predefined data (a literal text string) to a field in the device format descriptor; it does not process data from the output record of the applications program. The OUTPUT parameter (see below) is required when this construction is used, as the <u>Input/Empty-Conditional</u> mapping uses the same type of specification.

Specification:

```
FIELD (device-field-name,'text string'), OUTPUT
```

<u>Input/Empty-Conditional</u>: This input-only field functions the same as a <u>Direct</u> input-only field with one exception. If the data in the device field is all blanks (i.e., if the field is empty), the literal string is returned to the applications program instead of

the blank input data. The INPUT parameter (see below) must be specified, as the Output-Literal mapping uses the same type of specification.

Specification:

```
FIELD (device-field-name,'text string'), INPUT
```

Filler: This identifies this field as being a dummy field. It is used only to fill a gap in the input or output record. The LENGTH parameter (see below) is required when this construction is used. A field name is not permitted, as no mapping to a device field is done.

Specification:

```
FIELD FILLER, LENGTH n
```

System Information Field: A system information field causes system- or user-dependent data, such as time of day, date, user login name, etc. to be mapped to a device format field as if it were a piece of output data. Note that this has no effect on either input or output records.

System information field names and formats are as follows:

DATE1:	date, YY/MM/DD	[8 characters]
DATE2:	date, DD-MMM-YY	[9 characters]
DATE3:	date, MM/DD/YY	[8 characters]
DATE4:	date, DD.MM.YY	[8 characters]
TIME1:	time, HH:MM	[5 characters]
TIME2:	time, HH:MM AM/PM	[8 characters]
USERNAME:	user login name, XXXXXX	[6 characters]
USERNUM:	user number, NN	[2 characters]
FORMNAME:	form name, XXXXXXXX	[8 characters]

Specification:

```
FIELD (device-field-name,sys-info-field-name)
```

The following parameters are position-independent. They may occur anywhere after the mapping or literal specification.

3.5.1 LENGTH Parameter

This parameter defines the length of the field. It must be followed by an integer number greater than zero, which represents the field length, in characters.

Usage:

Mapping Type	Remarks
Direct	Required
Input-literal	Optional - if omitted, defaults to text string length; if supplied, text string is padded/truncated as required to meet given length
Output-literal	Same as input-literal
Empty-conditional	Same as input-literal
Filler	Required
System-info	Ignored

3.5.2 JUSTIFY Parameter

This defines the justification for the input and/or output field. It must be followed by one of the following four options:

o NONE	specifies no justification
o LEFT	the field is left-justified, right-padded
o RIGHT	the field is right-justified, left-padded
o CENTER	the field is centered

Note that 'JUSTIFY NONE' has the same effect as not specifying the JUSTIFY parameter at all.

If justification is specified on both the stream descriptor and format descriptor fields, it is justified as per the stream descriptor field specification on input and as per the format descriptor field on output.

Usage:

Mapping Type	Remarks
Direct	Optional
Input-literal	Optional
Output-literal	Optional
Empty-conditional	Optional
Filler	Ignored
System-info	Ignored

3.5.3 INPUT, OUTPUT, and INPUT-OUTPUT Parameters

These parameters, which form a mutually exclusive trio, define in what mode(s) this field should be processed. If INPUT is specified, the field is only processed on input; if OUTPUT is specified, it is only processed on output; if INPUT-OUTPUT is specified (guess what), it is processed on both. Note that when a field is "not processed", it is ignored completely.

Usage:

Mapping Type	Remarks
Direct	Optional; default is INPUT-OUTPUT
Input-literal	Default to INPUT, if specified, must be INPUT
Output-literal	Must be specified as OUTPUT
Empty-conditional	Must be specified as INPUT
Filler	Default to INPUT-OUTPUT
System-info	Ignored

3.5.4 VALIDATE Parameter

This optional parameter defines input validation of a field. It must be followed by one or more validation masks, enclosed in single quotes and optionally separated by the word 'OR'.

When a field with a validation specification is transferred to the user's input record at run-time, the data is checked against the validation mask(s). If it fails all tests, the rest of the input record is processed and control is passed to the error return if supplied, else to the normal return. The applications program may then interrogate the FORMS run-time package to determine which field(s) failed the validation tests and which fields passed.

A validation mask consists of a string of characters, each defining a certain criterion for each character in the field. The following is a list of validation mask characters and their meanings:

Mask Character	Validation Criteria
9	Numeric (0-9)
A	Alphabetic (A-Z, a-z)
X	Alphanumeric (0-9, A-Z, a-z)
.	Period
/	Fore-slash
B	Space (blank)
$	Dollar sign
-	Dash
	Any character
N	Numeric character (0-9, +, -, or blank)
F	Floating numeric (0-9, +, -, ., blank)
U	Unsigned integer (0-9, blank)
P	Personal name (A-Z, a-z, ., ', or blank)
Z	Alphabetic character or space

Usage:

Mapping Type	Remarks
Direct	Optional
Input-literal	Ignored
Output-literal	Ignored
Empty-conditional	Optional
Filler	Ignored
System-info	Ignored

3.5.5 FIX, NOFIX parameters

When a field with one or more validation masks fails to meet any of the specified validation criteria, the user has the option of forcing the operator to correct the data before FORMS allows the applications programmer to see it.

If FIX is specified, the data must pass one or more of the supplied validation tests before it is returned to the application program. If NOFIX is specified, the data is returned to the program whether or not it passes any of these validation tests. In most cases, it is much more convenient to require the data to be in the proper format when it reaches the applications program, thus eliminating the task of inspecting multiple fields on a character-by-character basis, which may be unnatural or impossible in the host language.

Usage:

Mapping Type	Remarks
Direct	Optional
Input-literal	Ignored
Output-literal	Ignored
Empty-conditional	Optional
Filler	Ignored
System-info	Ignored

3.5.6 START Parameter

This parameter allows the programmer to position the input and/or output character pointer to a given character within the record. It is equivalent to 'T' format in a FORTRAN FORMAT statement. START allows overlapping of input/output fields, a function not available with the 'Filler' map specification, which only allows forward positioning.

The word START must be followed by an integer number, which represents the new absolute position of the character pointer.

Warning: if START is specified in an input-only record, the character pointer gets reset for the input record but not for the output record. The inverse is true for output-only records. This is reflected in the Input and Output Stream Descriptors generated by FDL if A register bit 6 ('2000) is set on entry.

Usage:

Mapping Type	Remarks
Direct	Optional
Input-literal	Optional
Output-literal	Optional
Empty-conditional	Optional
Filler	Optional
System-info	Ignored

Examples:

```
    *    DIRECT MAPPING
               FIELD IDNUM, LENGTH 5

    *    INPUT LITERAL, START IN COLUMN 30
               FIELD 'LITERAL INPUT STRING', START 30

    *    OUTPUT LITERAL
               FIELD (HEADER,'HEADER TEXT'), OUTPUT
```

```
*    INPUT/EMPTY CONDITIONAL, OVERRIDE DEFAULT LENGTH
          FIELD (EMPLNAME,'NO NAME SPECIFIED'), LENGTH 40 ;
          INPUT

*    FILLER
          FIELD FILLER, LENGTH 4

*    SYSTEM INFORMATION FIELD
          FIELD (OUTDATE,DATE3)

*    INPUT FIELD WITH VALIDATION, JUSTIFICATION
          FIELD AGE, LENGTH 3, JUSTIFY RIGHT ;
          VALIDATE '999', INPUT
```

3.6 Device Format Descriptor FIELD Statement

The FIELD statement in the device format descriptor defines the position of a data item on the input/output device. There are two types of device descriptor field statements:

Mapped: this type of field is mapped from a stream descriptor field. All mapped fields must contain a 1-8 character field name starting in the left margin. Note that any fields in the stream descriptor that are mapped to non-existent fields in the device descriptor are ignored. The same is true for mapped fields in the device descriptor that are not "mapped to" in the stream descriptor by either a direct, output-literal, or input/empty-conditional field.

Literal: this type of field contains literal data specified in the field definition. It does not have a name, as no mapping may be done from a stream descriptor field. The literal must immediately follow the word FIELD and must be enclosed within single quotes.

The following parameters may follow the FIELD statement in a mapped field and the literal specification in a literal field. They are all non-positional; i.e., they may occur anywhere in the field definition. Note that all parameters apply to both the mapped and literal device descriptor field types. All parameters are optional unless otherwise noted.

3.6.1 LENGTH Parameter

This parameter defines the length of the field as it is to appear of the device. It must be followed by an integer number greater than zero, which represents the field length in characters. This parameter is required on mapped fields and optional on literal fields. If omitted, the field length is assumed to be the length of the literal string. Note that the length of a field in the stream descriptor may be different from the length of a field in the device format descriptor. The stream field defines the length in the input/output record of the applications program and the device format field length defines the length of the field on the input/output device. If they differ, the data is truncated or padded accordingly.

3.6.2 POSITION Parameter

This parameter defines the absolute position of the field on the input/output device in terms of column/line(/page) coordinates. It must be followed by either two or three numbers, enclosed within parenthesis and separated by commas. The first two numbers represent the column and line (x & y) coordinates of the field. The optional third number represents the physical device page on which the field resides. If the page number is omitted, it is assumed to be 1. The POSITION parameter is mandatory on both mapped and literal fields.

3.6.3 JUSTIFY Parameter

This parameter defines the justification of the field on input or output. Refer to the description of the JUSTIFY parameter in the stream descriptor FIELD description for information on the arguments. This parameter is optional on both mapped and literal fields and is defaulted to JUSTIFY NONE if not specified.

3.6.4 Attribute Parameters

The following eight parameters are used to set the display attributes when a data field is output to a device. Note that if a device does not support a certain feature, such as reverse video or blink, the attribute is ignored at run-time. Note that a word in square brackets following the attribute name means that it is synonymous with the preceding attribute (eg., ENABLE is synonymous with NOPROTECT).

3.6.4.1 NOPROTECT [ENABLE] Parameter

This parameter, which is mutually exclusive with PROTECT, declares this field to be write-enabled upon display to the user terminal. When displayed on the line printer the field is underlined.

3.6.4.2 PROTECT Parameter

This parameter declares that this field is to be displayed write-protected when output to the user terminal. When output to the line printer, it is displayed normally (not underlined). If both PROTECT and NOPROTECT are not specified, the default is PROTECT.

3.6.4.3 BLINK Parameter

This parameter declares this field to be blinked when displayed on the terminal. It has no effect in a device descriptor for the printer.

3.6.4.4 NOBLINK Parameter

This parameter declares that this field is not blinked when displayed to the user terminal. If both BLINK and NOBLINK are not specified, the default is NOBLINK.

3.6.4.5 REVERSE VIDEO Parameter

This parameter causes the field to be displayed in reverse video when output to the user manual. It has no effect when output is to the line printer.

3.6.4.6 NORMAL VIDEO Parameter

This parameter declares the field to be displayed in normal (not reverse!) video when output. If both the REVERSE VIDEO and NORMAL VIDEO parameters are omitted, the default is NORMAL VIDEO.

3.6.4.7 NODISPLAY [HOLD] Parameter

This parameter causes this field not to be displayed when the form is output. It is valid on all terminal and line printer device types.

3.6.4.8 DISPLAY [FREE] Parameter

This attribute causes this field to be displayed when the form is output to either the terminal or the line printer. If both the DISPLAY and NODISPLAY parameters are not specified, the default is DISPLAY.

Examples:

```
*   MAPPED FIELD, NOT WRITE-PROTECTED
INVNUM    FIELD  POSITION (70,2), LENGTH 6, NOPROTECT

*   LITERAL FIELD
          FIELD 'Literal String Test', POSITION (1,4) ;
                REVERSE VIDEO
```

3.7 Programming Aids

Following is a list of statements designed to assist the programmer in form definition. At present they include a macro capability and iterative field generation.

3.7.1 The DEFINE [DEF] Statement

This statement allows the programmer to define a macro. At present, a macro consists simply of one text item replacing another item or text string. Future plans call for implementation of macro arguments.

A DEFINE (or DEF) statement must be preceded by the name of the macro, starting in the left margin. The statement name must be followed by one or more spaces, and then by the macro text.

Whenever the macro name is encountered as a single item within an input line (not in a literal text string), the macro name is replaced by the given definition. Because FDL is a one-pass processor, a macro must be defined before it is used.

Macro definitions are not retained between form definitions; i.e., they are 'erased' after each END STREAM and END FORMAT statement. They are, however, retained across device block definitions.

Examples:

```
FLD      DEFINE  FIELD
LEN      DEFINE  LENGTH
POS      DEFINE  POSITION
D1X      DEFINE  5
D1Y      DEFINE  10
*
*
*   FIELD DEFINITION USING ABOVE MACRO DEFINITIONS
DATA1    FLD, POS (D1X,D1Y), LEN 10
*
*   NOTE THAT THIS HAS THE SAME FUNCTION AS:
DATA1    FIELD, POSITION (5,10), LENGTH 10
```

3.7.2 Iterative Field Generation

This feature of FDL allows the programmer to generate multiple blocks of field statements with only one block definition. Fields to be generated in this manner must be enclosed within REPEAT and END REPEAT statements (see below).

Iterative field generation is permitted in both stream descriptor and device format descriptor definitions. In both stream and device format descriptors, a two-digit iteration number is appended to any field names found in the left margin. If the field name is seven or eight characters, it is truncated to six characters to permit this iteration number to be appended. The same is true for device format ("mapped to") field names encountered in direct, output-literal, and input/empty-conditional stream descriptor fields.

3.7.2.1 The REPEAT Statement

This statement defines the beginning of an iterative field generation (REPEAT) block. It must be followed by an integer number, greater than zero, which represents the number of iterations to make through the following field definitions. The iteration counter is initially set to one and is incremented by one each pass through the REPEAT block. When the counter exceeds the specified repeat count, the statement immediately following the END REPEAT (see below) is processed.

Only FIELD statements are permitted within a REPEAT block.

3.7.2.2 The END REPEAT Statement

This statement terminates a REPEAT block. For each REPEAT statement, there must be one END REPEAT statement. REPEAT blocks may not be nested.

3.7.2.3 Relative POSITION Parameter Specification

A device format descriptor field may specify relative positioning when defined within a REPEAT block. This allows the programmer to define such fields in a repeat block without having them overlay one another on the I/O device.

Relative positioning is specified by placing a plus or minus sign immediately preceding the line and/or column definition in the POSITION parameter. The absolute line or column number is formed by adding or subtracting the current iteration number to or from the specified offset.

Example:

```
*
*   THIS BLOCK WILL BE REPEATED 3 TIMES
*
          REPEAT 3
LASTNM    FIELD  LENGTH 20, POSITION (10,+7)
FRSTNM    FIELD  LENGTH 10, POSITION (35,+7)
MIDDIN    FIELD  LENGTH 1,  POSITION (50,+7)
          END REPEAT
```

```
*
*
*   NOTE THAT THIS HAS THE SAME FUNCTION AS:
*
LASTNM01  FIELD  LENGTH 20, POSITION (10,8)
FRSTNM01  FIELD  LENGTH 10, POSITION (35,8)
MIDDIN01  FIELD  LENGTH 1,  POSITION (50,8)
LASTNM02  FIELD  LENGTH 20, POSITION (10,9)
FRSTNM02  FIELD  LENGTH 10, POSITION (35,9)
MIDDIN02  FIELD  LENGTH 1,  POSITION (50,9)
LASTNM03  FIELD  LENGTH 20, POSITION (10,10)
FRSTNM03  FIELD  LENGTH 10, POSITION (35,10)
MIDDIN03  FIELD  LENGTH 1,  POSITION (50,10)
```

3.8 Listing Control Statements

3.8.1 The NOLIST Statement

This statement disables the listing of all FDL statements, macro and repeat block expansions, except for those containing errors. It is overridden only by the 'FULL LIST' option bit in the A Register on entry to FDL.

3.8.2 The LIST Statement

This statement enables the listing of FDL statements, macro and repeat block expansions, after being disabled by the NOLIST statement. It is overridden by the 'ERRORS ONLY' option bit in the A Register on entry to FDL.

3.8.3 The EJECT statement

This statement causes the listing to eject to the top of a new page when the listing file is output (spooled) to the line printer. The old page header is retained. For a new page header, refer to the section entitled 'General Syntax'. Note that this statement has no effect if the listing is turned off (via the 'ERRORS ONLY' option bit or 'NOLIST' statement).

3.9 Alternate Input File ($INSERT) Command

A method exists whereby the programmer can 'insert' the contents of another FDL source file into his standard input file at compile time. This is accomplished by placing the command '$INSERT' in the left margin of the input line, followed by at least one space and the tree name of the disk file to be inserted. When the end-of-file is reached on the alternate input file, FDL returns to the line following the $INSERT in the main input file. Note that the main input file is not modified. Input flow is merely switched from the main to the alternate input file temporarily.

The $INSERT command provides a convenient method of reading a common macro definition file into an FDL source file.

Example:

```
     $INSERT <SOFTWR> FORMS> MACROS
```

3.10 Using FDL

FDL is invoked by typing the command 'FDL' following the 'OK,' prompt issued by PRIMOS. The command is followed by the name of the input file (if input is from disk), followed by an optional A Register setting.

The A Register option bits currently implemented are:

```
|O|M|E|F|T|R|X|S|S|S|L|L|L|B|B|B|
 1                             16
```

O	=	List emitted object text
M	=	Extended macro listing
E	=	Errors-only listing
F	=	Full list (override NOLIST pseudo-op)
T	=	List errors on terminal
R	=	List I/O stream format and device format
X	=	Expanded REPEAT block listing
S	=	Source (input) device:
		0 > none
		1 > terminal
		2 > paper tape
		3 > card reader
		4 > undefined
		5 > magtape
		6 > undefined
		7 > disk file
L	=	list (output) device; same values as 'S'
B	=	binary (output) device; same values as 'S'

The default A Register setting is '7777.

The rules regarding input and output (source, listing, and binary) file usage that apply to FORTRAN and PMA also apply to FDL. If the input file is open on entry to FDL, the file is read from the current file pointer to end of file and not closed on exit. If the input file is not open, FDL opens the file named in the command line. If an output (listing or binary) file is open on entry, data is written to that file starting at the current file pointer. The file is truncated but not closed upon completion of the compilation. If the output file is not opened on entry, an output file name is generated by appending the first four characters of the source file name to the characters 'L_' in the case of a listing file and 'B_' in the case of a binary file. This file is truncated and closed upon completion of the compilation. The file units corresponding to source, listing, and binary files are 1, 2, and 3, respectively.

After each form definition is processed, FDL types the number of errors, followed by the FDL revision number. A sample interaction between the user and the computer for an FDL compilation might look like this:

```
OK, CLOSE 1 2 3             (ensure no file units open)
OK, FDL FORM1 1/47777       (st'd options + full macro listing)
GO
0000 ERRORS  (FDL, REV 01)
ERROR C#01    (0006)     LEN  DEFIN  LENGTH
**** STATEMENT NOT RECOGNIZED.
0001 ERRORS  (FDL, REV 01)

OK, SPOOL L_FORM1
      .
      .
      .
```

3.11 FDL Error Messages

The Forms Management System generates a variety of different types of error codes. All errors generated by the FDL compiler are of the form:

```
C#nn    text message
```

Where 'nn' represents a unique two-digit error code for each type of error. The message printed is a one-line diagnostic of why the error occurred and possibly what action has been taken by the compiler. Following is a table which elaborates on the error codes generated by FDL.

C#00	BAD STATEMENT FORMAT. The contents of the statement field is not an alphanumeric text item. This statement is ignored.
C#01	STATEMENT NOT RECOGNIZED. The statement field does not contain a valid FDL statement. This line is ignored.

C#02 ARGUMENT REQUIRED.
An argument is required following the statement name. This statement is ignored.

C#03 ARGUMENT TOO LONG.
A text item exceeds 80 characters. This statement is ignored.

C#04 MULTIPLY DEFINED MACRO.
A macro by the same name already exists. This statement is ignored and the previous macro definition is retained.

C#05 BAD NAME FIELD.
The name field (starting in the left margin) contains an illegal character. This statement is ignored.

C#06 NAME REQUIRED.
A name must be present in the name field (starting in the left margin). This error is generally issued because a mapped field in the device format descriptor is missing a name. This statement is ignored.

C#07 STATEMENT FIELD IS BLANK.
A name was present in the name field, but no statement followed. This statement is ignored.

C#08 NO END STATEMENT; END ASSUMED.
An end-of-file was encountered while processing a stream or format descriptor. An END STREAM or END FORMAT is assumed here.

C#09 NOT PROCESSING STREAM DESCRIPTOR.
An END STREAM or SUBSTREAM statement was issued and a stream descriptor is not being processed. This statement is ignored.

C#10 END SUBSTREAM MISSING. IT IS ASSUMED HERE.
An END STREAM statement was issued while a substream block was being processed. An END SUBSTREAM is assumed prior to the END STREAM.

C#11 NOT PROCESSING SUBSTREAM.
An END SUBSTREAM statement was issued while not processing a substream block. This statement is ignored.

C#12 NOT PROCESSING FORMAT.
An END FORMAT or DEVICE statement was issued while not processing a FORMAT block. This statement is ignored.

C#13 END DEVICE MISSING. IT IS ASSUMED HERE.
An END FORMAT was encountered while still processing a device block. An END DEVICE is generated prior to the END FORMAT.

C#14 NOT PROCESSING DEVICE BLOCK.
A FIELD definition was issued after a FORMAT statement, but before a DEVICE block was started. This statement is ignored.

C#15 END STATEMENT MISSING; IT IS ASSUMED HERE.
A stream or format descriptor was not terminated before another was started. An END STREAM or END FORMAT is generated prior to this statement.

C#17 BAD PARAMETER.
This indicates that an unrecognizable parameter was present on a FIELD statement. The entire field definition is ignored.

C#18 INVALID FORMAT NAME.
The name supplied following the FORMAT parameter in the STREAM statement does not conform to the naming conventions discussed earlier in this document. This statement is ignored.

C#19 NAME NOT PERMITTED.
A name was present on a statement which does not permit one. This usually means that a literal field in the device format descriptor contains a name.

C#21 ALREADY PROCESSING SUBSTREAM.
A SUBSTREAM statement was issued while already processing a substream block. This statement is ignored.

C#22 VALIDATION STRING MISSING.
The VALIDATE parameter is present on a stream descriptor field, but is not followed by any validation masks. The entire FIELD statement is ignored.

C#23 BAD JUSTIFY PARAMETER.
The JUSTIFY parameter in the field descriptor is not followed by one of its four valid arguments. The field statement is ignored.

C#24 MAPPING SPECIFICATION REQUIRED.
A stream field descriptor is not followed by a mapping specification. The FIELD statement is ignored.

C#25 BAD MAPPING SPECIFICATION.
A stream field descriptor is not followed by a valid mapping specification. The field definition is ignored.

C#26 BAD LENGTH SPECIFICATION.
The LENGTH parameter in either stream or device descriptor is not followed by a valid numeric argument. The field definition is ignored.

C#27 BAD INPUT-OUTPUT SPECIFICATION.
An INPUT, OUTPUT, or INPUT-OUTPUT parameter has been misused. This usually means that INPUT-OUTPUT or OUTPUT has been issued when processing an input-literal field. This statement is ignored.

C#28 MAP FIELD NAME TOO LONG.
The "map to" field name in a stream descriptor field is longer than eight characters. This field is ignored.

C#29 ALREADY PROCESSING DEVICE BLOCK.
A DEVICE statement has been issued while already processing a device block. This statement is ignored.

C#30 SYNTAX ERROR.
This general error message is issued whenever two items in a field definition are separated by an illegal character. This statement is ignored.

C#31 BAD POSITION PARAMETER.
The POSITION parameter in a device format descriptor field is not followed by a valid argument. This statement is ignored.

C#32 POSITION OUT OF RANGE.
One or more of the arguments in the POSITION parameter is/are zero. This statement is ignored.

C#33 LENGTH PARAMETER MISSING.
The length declaration for a stream or device format descriptor field is required but not supplied. This field is ignored.

C#34 POSITION PARAMETER MISSING.
The POSITION parameter in a device descriptor field is not supplied. This field is ignored.

C#35 UNRECOGNIZED SYSTEM INFORMATION FIELD NAME.
The name specified in a system information field is unrecognized. This statement is ignored.

C#36 INPUT/OUTPUT SPECIFICATION NOT PERMITTED.
An INPUT, OUTPUT, or INPUT-OUTPUT specification was included on a system information field definition. This statement is ignored.

C#37 UNRECOGNIZED PARAMETER.
See C#17.

C#38 NOT PROCESSING STREAM/DEVICE FORMAT BLOCK.
A field definition has been issued outside of a stream or device format descriptor. This and all other field declarations up to the next STREAM, FORMAT, or DEVICE statement are ignored. Note that this error message is issued only once per violation.

C#39 MULTIPLY DEFINED SYMBOL.
A field name has been redefined within the same stream or device descriptor. This field is processed normally, but will produce undesired results at run-time.

C#40 BAD START SPECIFICATION.
The argument following the START specification in the stream field definition is not numeric and greater than zero.

C#41 ILLEGAL MACRO ARGUMENT SPECIFIER.
The item following the argument reference symbol (#) is not numeric and greater than zero. Note that this error code should not be emitted by FDL at this release. It is generated by a macro pre-scanner that is already incorporated into the compiler.

C#42 EOF ENCOUNTERED BEFORE END REPEAT.
An end-of-file was encountered on the input file before a repeat block was terminated. This usually causes abortion of the compilation.

C#43 END REPEAT MISSING - REPEAT BLOCK IGNORED.
An END statement was encountered while processing a REPEAT block. The entire repeat block is ignored and the END statement processed.

C#44 STATEMENT NOT ALLOWED WITHIN REPEAT BLOCK.
A statement other than a FIELD statement was found within a REPEAT block. The statement is ignored; processing of the REPEAT block continues.

C#45 INPUT/OUTPUT SPECIFICATION REQUIRED.
A input/empty-condition or output-literal field did not contain a required INPUT or OUTPUT statement.

C#46 INCONSISTENT SUBSTREAM USAGE.
A field definition appears outside of a substream block in a multi-record stream definition -or- the user has attempted to start a substream definition when previously defined fields do not reside within a substream. This error message is only issued once per stream descriptor.

3.12 FDL Temporary Files

In the course of translating the source file to the binary, FDL may create any of the following four temporary files:

Name	Format	Contents
ER##uu	ASCII	Error definitions (*)
RP##uu	ASCII	Current repeat block
IN##uu	ASCII	Input stream/substream definition
OU##uu	ASCII/ binary	Output stream/substream format Device format map

Note that all files are created and deleted by FDL; the only way that the user can "see" these is if he quits (BREAK key or control/p) out of the translator.

* The 'uu' in the file name denotes the current user # - this permits multiple FDL translations simultaneously within the same directory.

SECTION 4

FORMS ADMINISTRATIVE PROCESSOR

4.1 Centralized FORMS Directory Information

The FORMS directory, called 'FORMS*', may be located on any disk in the computer system. It is strongly recommended, however, that due to the time element involved in accessing remote files, the directory be located on a local disk as opposed to a disk started remotely across the network.

The FORMS UFD, which may be created by FAP, contains two files. The first is the FORMS segment directory, called 'FMS.**' which contains all form definitions and internal directories used by FAP and the run-time package. This segment directory must be created using the FAP CREATE command. The second file, called 'DCF.AS', is the FORMS system device control file. It contains information for each device configured into the system. This file is created by the system administrator using the standard text editor. The format of the DCF is discussed in the section entitled "Device Mapping Scheme".

4.2 FAP Commands

The following is a description of the nine commands supported by FAP. All command names may be abbreviated to three characters.

4.2.1 The CREATE Command

The CREATE (or CREATE DIRECTORY) command allows the system administrator to create a skeleton FORMS directory. The CREATE command processor first checks to see if the FORMS UFD (FORMS*) exists on any started-up disk. If not, it inquires as to which disk the UFD is to be created on and requests the MFD owner password for the disk. Once created, or if already present, FAP creates a skeleton segment directory, with an empty catalog and terminal configuration block. The user may now insert form definitions into the directory.

If the FORMS UFD is already present on the system, FAP creates the segment directroy and prints the message 'DIRECTORY CREATED'. If the UFD is not a first-level directory (directly under an MFD) on any started-up disk on the system, FAP requests a disk volume-ID on which the UFD is to be created. The user must then enter the volume-ID (DSKRAT name) of the pack/partition which will contain the FORMS directory. FAP then asks the user for the MFD owner password on this volume. Once this has been supplied, the FORMS UFD and segment directory are created. Note that the CREATE command will produce an error if both the FORMS UFD and segment directory exist. The segment directory must be TREDELeted with FUTIL before it can be CREATE'd.

All error messages produced by the CREATE command processor are self-explanatory.

Following is an example of CREATE command dialogue. Note that all underlined data is entered by the user.

```
OK, FAP
GO

FORMS ADMINISTRATIVE PROCESSOR, REV 12.0-P

* CREATE

UFD "FORMS*" DOES NOT EXIST.
SHALL I CREATE IT? YES
ENTER DISK VOLUME-ID:  TS/A
ENTER OWNER PASSWORD  (IT WON'T ECHO):  ABCXYZ
THIS MFD IS FULL, TRY AGAIN.

ENTER DISK VOLUME-ID:  SOFTWR
ENTER OWNER PASSWORD  (IT WON'T ECHO):  XXXXXX
DIRECTORY CREATED.

*
```

On any input request within the CREATE dialogue, the user may type control/c to abort creation and return to the FAP command level.

4.2.2 The ADD Command

This command allows the addition of form definitions to the FORMS catalog. The name of the binary form definition file, generated by the FDL translator, must follow the ADD command. This file name usually starts with 'B_'. Note that one binary file may contain more than one form definition, eg. if there was one stream descriptor and three device format descriptors in the source file, the binary file contains those four form definitions. FAP considers each device descriptor under one format descriptor to be a separate form.

The ADD command adds only new modules to the FORMS catalog; any attempt to redefine a form already residing in the FORMS catalog with the ADD command causes the new form definition to be ignored and a message printed on the user terminal to that effect.

The input (binary) file name may optionally be followed by the word LIST or LIST UPDATES. If this is specified, all form definitions added to the FORMS directory are listed by name on the terminal.

When the entire binary file has been processed, the number of modules added and ignored (due to duplicate entries) is printed.

Should the message 'WARNING! "form-name" CONTAINS ERRORS' be printed, it denotes that this stream or format descriptor contains FDL errors. The user should fix the source file and recompile it with FDL. This binary form definition will probably generate undesirable results at run-time.

Examples:

```
     * ADD B-FM03
     01 DEFINITION ADDED.
     * ADD B-FM04 LIST

       DEDUCT    STR                V00    ADDED
       DEDUCT    FMT    VISTAR3     V00    ADDED
       DEDUCT    FMT    PRINTER     V00    ADDED

     03 DEFINITIONS ADDED.
     *
```

4.2.3 The REPLACE Command

This command functions the same as the ADD command, but causes any form definitions in the FORMS catalog which are redefined in the input (binary) file to be replaced with the new definition. Any form definitions in the binary file that are not defined in the catalog are added.

Examples:

```
     * REPLACE B-F019
     02 DEFINTIONS REPLACED.
     * REPLACE B-F020
     01 DEFINITION ADDED    03 DEFINITIONS REPLACED.
     *
```

4.2.4 The PURGE Command

This command purges form definitions from the FORMS catalog. The command must be followed by a form name specification (see below), which designates what form definition(s) is/are to be purged. It may also be followed by the word LIST or LIST UPDATES, which will cause all purged forms to be listed by name on the user terminal.

4.2.4.1 Form Name Specification

The form name specification designates the form definitions to which this command applies. Both PURGE and LIST commands use this option.

The form name specification is enclosed within parentheses and has the following formats:

```
form-name
form-name.type
form-name.type:device

type  =  STR   for stream descriptor
         FMT   for format descriptor
```

If only the form-name is specified, this command relates to all forms with the given name, any type and any device (if format). If the second specification is used, the command relates to all forms of the given name and type. If the type is FMT, it relates to all device descriptors within the format definition. If the third type of specification is used, the command relates to the one definition that contains the same name, type, and device. Note that this construction should only be used on format descriptors (there is no device definition for a stream descriptor).

If any item in the form name specifier (form-name, type, or device) is specified as an asterisk (*) or the word ANY, this will cause no check to be made on this item when scanning the FORMS catalog.

Up to 20 form names may be specified within the parentheses, separated by commas.

Examples:

(TAXFORM)	All forms of name 'TAXFORM', any type, any device
(TAXFORM.STR)	TAXFORM, stream definition
(TAXFORM.FMT:PRINTER)	PRINTER format definition for TAXFORM
(*.*:VISTAR3)	All VISTAR3 device format def's
(*.STR)	All stream'descriptors
(TAXFORM,SHIPFORM)	All forms with names TAXFORM or SHIPFORM, any type, any device

PURGE Examples:

```
* PURGE (FM0020)
01 DEFINITION PURGED.
* PURGE (FM0021,FM0022,FM0023,FM0024) LIST

  FM0021     STR               V02   PURGED
  FM0022     STR               V00   PURGED
  FM0024     STR               V00   PURGED
  FM0024     FMT    PRINTER    V00   PURGED
  FM0024     FMT    VISTAR3    V00   PURGED

05 DEFINITIONS PURGED.
*
```

4.2.5 The LIST Command

This command causes all or part of the FORMS catalog to be listed by name and type. This may be followed by a form name specifier (see above) to selectively list a part of the catalog. If the form name specifier if omitted, the entire catalog is listed. If the phrase FILE <filename> or ON FILE <filename> is included, the catalog listing is output to the specified file. If the phrase ON TERMINAL is specified, or if the ON FILE specifier is omitted, the listing is written to the user terminal.

The information listed in the catalog listing includes:

- o Form-name, type, and device (if any)
- o Version number
- o Creation, last access, last modified dates (file output only)

Examples:

```
* LIST

FORMS DIRECTORY LISTING ON WEDNESDAY, JANUARY 12, 1977 AT 9:45 PM

    NAME      TYPE   DEVICE    VER
    -----     ----   ------    ---

   HDRF01     STR              V02
   HDRF01     FMT    VISTAR3   V00
   HDRF02     STR              V00
   HDRF02     FMT    VISTAR3   V00

04 ENTRIES.
* LIST (HDRF01) ON FILE CATLOG
*
```

4.2.6 The QUIT Command

The QUIT command causes FAP to exit and return to PRIMOS command level. FAP may be re-entered by typing the START (S) command.

Example:

```
    * QUIT

    OK,
```

4.2.7 The JOURNAL Command

The JOURNAL command allows the user to log his transactions with the FORMS catalog in an ASCII file which can be spooled to the line printer. All ADD, REPLACE, PURGE, and TCB (see below) transactions are recorded in the JOURNAL file.

This command may be used to enable or disable the logging function. To disable it, the command JOURNAL or JOURNAL STOP may be issued. To enable it, the command JOURNAL <filename> or JOURNAL START ON <filename> may be issued.

Example:

```
    * JOURNAL LOG000
    * ADD B-F01
    08 DESCRIPTIONS ADDED.
    * JOURNAL STOP
    *
```

4.2.8 The TCB Command

The TCB command modifies the terminal configuration block table. The TCB is a 64 by 4 word file which contains the terminal type for each FORMS user on the (local) computer system. This file, contained in the FORMS segment directory, is used in conjunction with the device control file (DCF) at run-time to select the terminal device driver for a given FORMS user. Both TCB and DCF files are explained in detail in the section enitiled "Device Mapping Scheme".

The TCB command may be followed by the word LIST to dump the contents of the TCB on the user terminal. Optionally, LIST may be followed by a file name. If this is the case, the contents of the TCB will be dumped to the specified file.

To modify the TCB, the command may optionally be followed by one of three noise words to reflect the type of operation being performed: ADD, CHANGE, or DROP. This must then be followed by the user number for which this operation applies, from 1 to 64. If the user wishes to drop the current TCB entry for this user, he may terminate the command line by typing RETURN. If the user attempts to drop an non-existent entry, FAP prints a warning message and returns to command mode. If he wishes to add or change the terminal type, he must type the 1-8 character terminal name. If the specified user already had an entry, the name of the old terminal type is printed on the terminal.

Examples:

```
* TCB LIST

TERMINAL CONFIGURATION ON WEDNESDAY, JANUARY 12, 1977 AT 10:03 PM

   USER  TERMINAL
   ----  --------

    4    VISTAR3
   12    Z9003
   13    VISTAR3
   20    Z9003

* TCB 3 VISTAR3            (set user 3 = VISTAR3)
* TCB 12 B500              (change user 12 to B500)
WAS Z9003.
* TCB 13                   (drop user 13's entry)
* TCB LIST

TERMINAL CONFIGURATION ON WEDNESDAY, JANUARY 12, 1977 AT 10:04 PM

   USER  TERMINAL
   ----  --------

    3    VISTAR3
    4    VISTAR3
   12    B500
   20    Z9003

*
```

4.2.9 The GENERATE Command

This command generates three $INSERT files for the run-time device interlude subroutine.

The files generated are as follows:

- o DEVEXT - external declaration statements for run-time device drivers
- o DEVDAC - 64R mode driver dispatch table
- o DEVIP - 64V mode driver dispatch table

All three files must reside in the same UFD as the sources for the run-time input/output system in order to be assembled with the I/O package.

The GENERATE command should be issued and a new run-time I/O package assembled each time the device control file (DCF) is modified.

For more information on the device control file and input/output scheme, refer to the section entitled "Device Mapping System".

4.2.10 Using FAP

FAP is invoked by typing the command 'FAP' following the 'OK, ' prompt issued by the operating system. FAP prints a header line, followed by the current revision number. If bit 1 in the A Register is set when FAP is started, all updates to the FORMS directory and terminal configuration table are automatically recorded in a file called 'FAP.UP' in the FORMS control directory. It is strongly recommended that if this option is to be used all of the time, FAP be RESTORE'd and SAVE'd with the A Register set appropriately. When this option is used, the JOURNAL command is disabled.

4.2.11 FAP Error Messages

Like FDL, all FAP error messages are of the form:

```
t#nn   text message
```

The 't' in the error code represents the error type. At present, there are three such types:

- . F - file system/input file/control block error
- . S - syntax error
- . T - TCB or DCF format error

The 'nn' represents a 2-digit error number, unique for each error message generated by FAP.

Following is a list of error messages and explanantions:

F#00 CONTROL BLOCK UFD DOES NOT EXIST.
An operation other than CREATE was attempted and the FORMS UFD ('FORMS*') does not exist on the system.

F#01 CONTROL BLOCK DIRECTORY DOES NOT EXIST.
An operation other than CREATE was attempted and the FORMS segment directory ('FMS.**') does not exist within the FORMS UFD.

F#04 INPUT FILE IS EMPTY.
The input file specified in an ADD or REPLACE command is empty.

F#05 PREMATURE EOF.
An EOF was encountered on the input file in an ADD or REPLACE command before the end-of-data record. The module is deleted from the control directory. This is usually caused by the user depressing the BREAK key in the middle of an FDL compilation.

F#06 FILE DOES NOT EXIST.
The input file specified in an ADD or REPLACE command does not exist in the current UFD.

F#07 BAD INPUT FILE.
The input file specified in an ADD or REPLACE command is not a valid FDL binary file. No action is taken with this file.

S#00 FILE NAME REQUIRED.
An ADD or REPLACE command was issued, but no file name followed. The command is ignored.

S#01 BAD FORM NAME SPECIFIER.
The form name specifier contained a syntax error. This command is ignored.

S#02 BAD ARGUMENT.
One of the parameters in the command line was not recognized. The command is ignored.

S#03 BAD TYPE.
The form name specifier contained a type declaration other than STR (stream) or FMT (format). This command is ignored.

S#04 NO FORM NAME SPECIFIED.
A PURGE command was issued without a required form name specifier. The PURGE command is ignored.

S#05 MISSING ARGUMENT.
The TCB command was issued without any following user number. The command is ignored.

S#06 BAD USER NUMBER.

The user number specified in the TCB command is not an integer number greater than zero. The TCB command is ignored.

S#07 BAD TERMINAL NAME.
The user attempted to assign the name 'PRINTER' as a terminal type in a TCB command. This is not permitted and the TCB command is ignored.

T#00 DCF DEVICE INTERLUDE FIELD ERROR.
The device interlude number field in the given DCF entry is not numeric or greater than zero. The DCF must be edited and corrected before continuing.

T#01 DCF DEVICE NAME FIELD ERROR.
The device name field in the given DCF entry contains an illegal character or is empty. The DCF must be edited and corrected before continuing.

T#02 DCF DEVICE ABBREVIATION FIELD ERROR.
The device abbreviation field in the given DCF entry is empty or contains a space or illegal character. The DCF must be edited and corrected before continuing.

T#03 TCB LINE/COLUMN FIELD ERROR.
The line or column specification field in the given DCF entry is empty, contains a non-numeric value, or is less than 1. The DCF must be edited and corrected before continuing.

T#04 MAX DEVICE NUMBER EXCEEDED IN DCF.
The DCF contains an entry with a device interlude number greater than 50. This error is issued from the GENERATE command only. Only 50(!) devices may be in use at one time.

T#05 DEVICE CONTROL FILE EMPTY.
The DCF is empty and the user issued a TCB or GENERATE command.

T#06 TERMINAL UNDEFINED.
The terminal type specified in the TCB command is not present in the DCF.

SECTION 5

FORMS RUN-TIME PACKAGE

5.1 General Information

The FORMS run-time package invisibly performs all forms lookup, buffer management, data manipulation, and input/output for the applications program.

Before the applications program can input and output data from/to a form, he must first request the run-time package to retrieve the form definition from the FORMS catalog. When he is through with the form, he must release it; i.e., inform the run-time package that it is no longer needed. This function is mandatory in a shared procedure environment, as idle form definitions can take up unnecessary table space in the buffer pool manager, of which a finite amount is available.

As previously stated, the two devices available to the user at this release are the user terminal and the offline (spooled) line-printer. To use a form with the terminal, the user reads and writes to logical unit 1 in Fortran, or reads and writes data in a file selected and assigned to the TERMINAL in Cobol. For the line printer, the user writes to logical unit 4 in Fortran or writes a record in a file selected and assigned to the OFFLINE-PRINTER in Cobol. Note that both devices may be used simultaneously with the forms system.

5.1.1 Device I/O Processing

The FORMS run-time system handles all device input/output, but it may be convenient for the applications programmer to know when and how physical device I/O is going to take place. A form is initially output to a device when the entire stream descriptor has been output by the applications program. Subsequent stream descriptor output causes the form to be modified if on the user terminal (only those fields that are changed are actually modified on the CRT). On the line-printer, each time the end of stream descriptor is encountered, a new form is output.

For READ statements, on a single record form (no substream definitions), each READ causes the device driver to wait for and process input from the terminal. The input data is then transferred to the applications program. In a form with multiple substream definitions, the initial READ causes the device driver to process input from the terminal. The data for the first substream definition is then transferred to the applications program. Subsequent READs transfer the remainder of the data.

5.2 Command Descriptions

All commands to the FORMS run-time system are issued via output records from the applications program. FORMS commands described below must be immediately preceded by two hash marks ('##').

5.2.1 The INVOKE Command

This command invokes the usage of a specific form by the applications program. The stream descriptor name is specified as an argument, and must be separated from the command by at least one space.

The INVOKE command processor retrieves both stream and device format definitions from the FORMS catalog through the buffer pool manager. It then initializes an input/output data area, known as 'IOLST', which is used to pass both data and control information (such as position, attributes, etc.) to and from the device driver by the run-time package proper. Note that if a form was already invoked on this device, the old form is released before the requested form is retrieved. If the requested form is the same form that was previously invoked, the form is only initialized, that is, the attributes are reset to their original states and data is reset to spaces.

All errors (such as 'FORM NOT FOUND') usually cause the program to abort and print a suitable error message on the user terminal.

All future input/output requests are trapped and processed by the run-time system as data for the form in use. Usage of the form is terminated by a RELEASE command (see below).

Example:

```
##INVOKE SHIPFORM
```

5.2.2 The RELEASE Command

This command informs the form system that the form definition for this device is no longer needed. Future input and output data will be processed by the standard device drivers (O$AA01 and I$AA12 for the terminal; O$AL06/04 for the line-printer).

Example:

```
##RELEASE
```

5.2.3 The STAT Command

This command causes the run-time package to return the validation status of all input data on the next READ statement(s).

The status is returned in the form of a two-digit number for each input/empty-conditional or direct field that is not declared as output-only. It is not returned for input-literal fields. The two-digit number returned can represent one of the following three conditions:

Value	Condition
-1	The data failed all validation tests.
00	No validation specified for this field.
>0	This is the number of the first validation mask that the data passed. Validation masks are numbered in the order in which they appear in the field definition.

The validation status is returned in the same manner that data is returned on a READ statement. If there are multiple substream definitions, the user must do multiple READs to input the validation status for all fields. The STAT command causes the next READ statement to input the validation status of the first substream in the stream descriptor unless a SUBSTREAM command is issued before it. The STAT function is disabled and normal data input resumed when either the end of the stream descriptor is encountered or a SUBSTREAM command is issued.

5.2.4 The SUBSTREAM Command

This command sets the next substream to be processed on a READ or WRITE statement. The substream name must follow the command and be separated from it by at least one space. If the named substream does not exist in the stream descriptor, an error message will be generated but the program will be allowed to continue. In this case, the next substream to be processed will be the first one defined in the stream descriptor.

Example:

```
##SUBSTREAM EMPLDATA
```

5.2.5 The CLEAR Command

This command clears all unprotected data displayed on the user terminal. It also causes all data items marked as unprotected and displayed in the input/output list to be reset to spaces. This is a fast and convenient method to erase all operator-input data. The other way is writing spaces into all unprotected fields (sigh!).

If this command is followed by the word 'ALL', the entire display is erased. This should only be done before a RELEASE command, else catastrophe will certainly strike!! This option was added to allow the applications program to leave the terminal in a 'human' state before exiting to command level.

Examples:

```
##CLEAR

##CLEAR ALL
```

5.2.6 Attribute Modification Commands

The applications program may dynamically change the attributes of a device format field by naming its corresponding stream descriptor field in one of the eight attribute commands described below. From one to twenty stream descriptor field names may be placed as arguments, each separated by at least one space. The attribute change occurs the next time the form is output to the device.

The following table describes each of the eight attribute modification commands and three synonyms.

Command/Synonym	Description
PROTECT	Write-protects field
NOPROTECT/ENABLE	Write-enables field
RVIDEO	Field displayed in reverse video
NVIDEO	Field displayed in normal video
BLINK	Blinks field when displayed
NOBLINK	Field is not blinked when displayed
DISPLAY/FREE	Field is displayed when form is output
NODISPLAY/HOLD	Field is not displayed when form is output

Examples:

```
##PROTECT FIELD1 FIELD2 FIELD3

##BLINK ERRMESG

##RVIDEO BALANCE CREDIT
```

5.3 Run-Time Error Handling

All errors that occur at run-time are recorded in a file in the home UFD called 'FMS##E', where the '##' represents the user number.

The file format is as follows:

```
mmddyy  hhmm  ee  error message
```

Where 'mmddyy' represents the date (month, day, year), 'hhmm' represents the time that the error occurred (hours and minutes after midnight, local time), and 'ee' represents the error number. Each run-time error condition is assigned a unique error number. The following is a list of run-time error numbers, accompanying diagnostic messages, and a brief explanation of why the error occurred. Unless otherwise specified, all errors are fatal.

01 ROLLOUT DIRECTORY FULL (pool manager)
The FORMS UFD ('FORMS*') is full and the pool manager needs to write a form definition to a temporary file.

02 ROLLOUT FILE MISSING (pool manager)
The pool manager cannot locate a file that was temporarily rolled out to the FORMS UFD. This either indicates that a user has deleted it, or less likely, that the pool manager is sick.

03 OBJECT BLOCK TOO LARGE (pool manager)
A form definition is too large to fit into the entire buffer pool. The systems administrator must reconfigure the buffer pool manager for a larger buffer pool in order to use this form.

04 POOL ALLOCATION MAP FULL (pool manager)
A form definition was requested by the user, but the pool manager allocation map cannot hold it. This map is currently configured for 60 entries. This error message usually means that applications programs have not been RELEASE'ing form definitions when they are through with them.

05 STREAM/FORMAT DEFINITION MISSING (pool manager)
The user has requested a non-existent stream or format descriptor. The name of the requested definition is included in the error message.

06 USER LOCKED IN POOL MANAGER (pool manager)
A user became locked in the buffer pool manager. This means that the current user was waiting more than 50 time-slices for another user to exit it. This was probably caused by the "locked" user aborting his program (via BREAK or control/p) while using the pool manager. This is an informative message only - it is written to the error file but not to the terminal. The current user is allowed to continue running and enter the pool manager.

07 INVALID CALL TO FMS$DV (run-time package)
The form-initialization processor within the run-time package passed a bad internal unit number to the device translation subroutine. This error should never occur - it indicates that the run-time package is sick.

08 USER DOES NOT HAVE VALID TCB ENTRY (run-time package)
A user without a TCB entry attempted to invoke a form on the terminal. The cure for this error is to run FAP and, using the TCB command, insert a valid terminal name in the terminal configuration block entry for this user.

09 DEVICE DOES NOT HAVE DCF ENTRY (run-time package)
The run-time package could not find a DCF entry for the specified device. The system administrator should update the DCF and recompile the input/output system.

10 ILLEGAL CONTROL STATEMENT (run-time package)
The applications program issued an unrecognized control statement to the run-time package. The statement name is printed in the message text.

11 SUBSTREAM ERROR (run-time package)
An end-substream control block was encountered while not processing a substream block. This error usually indicates that the run-time package or FDL is sick.

12 TOO MANY SUBSTREAMS (run-time package)
The stream descriptor contained too many substream descriptions. At this release, the maximum number of substreams permitted under one stream descriptor is 30.

14 I/O LIST OVERFLOW (run-time package)
The storage required for control and data information for the current form exceeds the amount of space available. The cure for this error is for the run-time I/O list to be enlarged. At this release, 1000 words are allocated to the I/O list.

15 DCF FORMAT ERROR (run-time package)
A format error exists in a DCF field. The user may either inspect the DCF or invoke FAP and type the GENERATE command to determine which field or fields is/are in error.

16 SUBSTREAM NOT FOUND (run-time package)
A SUBSTREAM command was issued from the applications program, but the named substream does not exist within the current stream descriptor. The error is logged in the error file, but the program is allowed to continue. The next input or output statement will cause the first substream descriptor in the form to be used.

50 LINE SEQUENCE ERROR (V3$IO)
The Vistar/3 I/O driver found the I/O list to be out of sequence. This indicates that the run-time package proper, in particular FMS$LK, is sick.

90 LINE SEQUENCE ERROR (PR$IO)
The line-printer I/O driver found the I/O list to be out of order. See the description for error #50, above.

91 READ ATTEMPTED FROM PRINTER (PR$IO)
The user attempted to read a form from the line-printer. Consider yourself suitably scolded!

92 SPOOL DIRECTORY FULL (PR$IO)
The RELEASE command issued from the applications program caused the line-printer driver to attempt to insert the printer form into the spool queue, however, the spool directory is full.

5.4 Run-Time System File I/O

The run-time package (run-time package proper, buffer pool manager, and line-printer driver) all use file units 12-16 to perform form definition lookup, rollout file I/O, and other associated file system operations. This will not interfere with KI/DA which reserves file units 12-15, and the rev-12 file system bounce package, which uses file unit 16.

For each form definition invoked, FORMS generates a temporary file in the home UFD named 'FMlluu', where:

'll' represents the internal logical unit #:
  01 = user terminal
  02 = line-printer
'uu' is the current user #

These temporary files contain the current form status (iolist and related variables) and are "swapped" in and out of memory when multiple form definitions are in use. They are deleted when the form definition is released, and should never be visible to the user.

SECTION 6

INSTALLING FORMS

As shipped, FORMS resides on two directories. The first, called FORMS, contains all sources and command files to create the FDL translator, FAP utility program, and 64R and 64V mode libraries. The FORMS UFD contains the following files and directories:

File	Type	Description
FDL	subUFD	sources for FDL translator
FAP	subUFD	sources for FAP utility
RUN	subUFD	sources for run-time package proper
PMGR	subUFD	sources for buffer pool manager
IOS	subUFD	sources for input/output system
DOC	subUFD	source for this document
C_RLIB	cominp	creates 64R mode library from indiv. objects
C_VLIB	cominp	creates 64V mode library from indiv. objects
RFORMS	object	64R mode FORMS library
VFORMS	object	64V mode FORMS library
MACROS	insert	$INSERT file containing FDL macro definitions
C_INST	cominp	installs FDL, FAP, and libraries
C_SHAR	cominp	template for creating shared procedure memory image files
C_2010	cominp	creates seg 2010 shared procedure
C_4016	cominp	creates seg 4016 shared procedure (for test)
C_LOAD	cominp	template for loading shared procedure FORMS system

The second directory, called 'FORMS*', contains a skeleton FMS.** segment directory and a device control file containing information for all devices supported by Prime.

To install FORMS as shipped, the user need only run the command file called 'C_INST', which performs the following functions:

- o Copies FORMS>FAP>*FAP to CMDNC0>FAP
- o Copies FORMS>FDL>*FDL to CMDNC0>FDL
- o Copies FORMS>RFORMS to LIB>RFORMS
- o Copies FORMS>VFORMS to LIB>VFORMS

It is strongly recommended that the copy of FORMS as shipped be copied and saved in case of accidental damage or deletion of the FORMS source directory.

SECTION 7

DEVICE INPUT-OUTPUT SYSTEM

7.1 Device Mapping Scheme

All devices supported by FORMS are assigned a unique 'logical device number'. When the run-time package wishes to perform some function on a device, it calls an interlude subroutine passing the logical device number as an argument. The interlude subroutine then calls the corresponding device driver to perform the specified function.

FORMS uses a file known as the device control file (DCF) to maintain certain information about each device supported by the system. The primary purpose of the DCF is to assign each device name to a logical device number. When the device name becomes known to the run-time package, the DCF is scanned to extract the related logical device number, thus allowing the run-time package to "communicate" with the device.

The device interlude subroutine (DEV$IO) contains a table which points to the beginning address for each device driver. It uses the logical device number to index into this table (much like indexing into a Fortran array) to select the device driver to be used. Because of this relationship between the DCF and device interlude subroutine, we can conclude that any time the DCF changes, the device address table within DEV$IO must change accordingly. The FORMS Administrative Processor (FAP) has a facility for generating a new device driver address table any time the DCF is updated. The Input/Output System must then be rebuilt with the new address table (this process is discussed in detail later).

So far, we have discussed how the run-time system translates a device name into a device driver address. We have not, however, discussed how it goes about determining the device name. For the line-printer, IOCS logical unit 4, the hard-wired name 'PRINTER' is assigned, after which the DCF lookup and subsequent processing described above takes place. The terminal, however, involves yet another step.

Because FORMS permits multiple terminal types, another file, known as the terminal configuration block (TCB) is needed. This file assigns a terminal type (eight-character name) to each FORMS user on the system. The run-time package uses the user number, as returned from the operating system, to index into this file and extract the terminal name. After this is done, the DCF lookup and subsequent processing described above takes place.

The device control file resides in the FORMS UFD ('FORMS*') and is called 'DCF.AS'. It is an ASCII file and may be changed with the text editor (ED). The file contains one record for each device supported by FORMS.

The record format is as follows:

ldn, name, dna, lines, columns

ldn	=	logical device number
name	=	device name, 1-8 characters
dna	=	device name abbreviation, see below
lines	=	# lines on the device (max Y coordinate)
columns	=	# columns on the device (max X coordinate)

The device name abbreviation represents the first two characters of the name of the device driver. This must be two characters, the first being alphabetic. A space is not permitted, and will yield an error from the FAP GENERATE processor. The full device driver name consists of the two-character abbreviation followed by '$IO'. For example, the line-printer uses the two-character abbreviation 'PR', and hence the device driver name is 'PR$IO'. Note that each device name abbreviation must be unique.

The terminal configuration block lives in the FORMS segment directory ('FMS.**') and may be altered with the TCB command in FAP. Refer to the section entitled 'FORMS Administrative Processor' for more information.

7.2 User-Written Device Drivers

Should the user have a terminal not supported by FORMS as released by Prime, he may want to write his own device driver.

7.2.1 Terminal Requirements

Any terminals to be used with FORMS must have the following capabilities:

o Internally buffered (block transmission) mode

o Protected fields

o Absolute cursor positioning

o Both protected and unprotected data modification once displayed

o Clear entire screen/clear unprotected data commands

Other features that could be taken advantage of by the FORMS system or device driver include:

o Blink

o Reverse video

o Underlining

o Keyboard lock

o Input and/or output space compression

7.2.2 Device Driver Specification

Device drivers must be named 'XX$IO', where the 'XX' represents the two-character abbreviation used in the device control file. They have the following calling sequence:

CALL XX$IO (function, iolist)

Function is one of the following nine function codes:

1 -- Initialize device: Reset all device logic, clear the entire screen, and enter block transmission mode (if this is a software function).

2 -- Output initial form: Write the contents of the entire I/O data list (IOLIST) to the screen. The device driver should reset bits 1, 2, 3, and 4 of the attribute word for each entry and set bit 5 for each field displayed. It should not display any fields with the 'NODISPLAY' bit (bit 14) set. When the screen has been output, the cursor should be positioned to the first write-enabled location on the screen.

3 -- Input form: The device driver should wait for the operator to fill in the displayed form and process the input as it is transmitted from the terminal. As it receives the data, the driver is responsible for inserting it into data area in each field in the I/O list. Only fields with the 'DISPLAYED' and 'ENABLED' bits set in the attribute word should be input. It should be noted that on a full duplex line, the device driver should disable the echo and auto-linefeed generation with a call to DUPLX$; this must be restored after the data has been input. If possible, a brief prompt message should be output in an out-of-the-way place on the screen, informing the operator that there is an input request pending.

4 -- Modify existing form: The device driver must examine each entry in the I/O list and update those fields with attribute bits 1, 2, or 4 set. Following is the recommended logic for the modify processor:

- o if data changed, enable/protect changed, or field attribute changed bits are all reset, process next field, else
- o save current attribute word in a temp and reset bits 1-4 (data/attributes modified) of the attribute word in IOLIST, then
- o extract field length, and x,y coordinates from IOLIST, then
- o if the field is currently displayed and 'NODISPLAY' bit is set, erase this field from display and process next, else
- o if field is not currently displayed and 'NODISPLAY' bit is reset, display the field according to the supplied attributes and x,y coordinates, else
- o if 'NODISPLAY' bit is set, ignore this field and process next, else
- o if enable/protect changed bit is set and special handling is required to accomodate this change, perform this special handling; either way,
- o if attribute changed bit is set, update the field using the new attributes and process the next field, else
- o update the data and process the next field

5 -- Clear entire screen.

6 -- Clear unprotected data on screen.

7 -- Close device: This function code is used to terminate device usage after a RELEASE command and is applicable only to the line-printer driver; terminal device drivers should ignore this call.

8 -- Correct data: The device driver must scan the I/O list for the first field with the 'data-invalid' attribute bit set (see below), position the cursor to the first character position of this field, and allow the operator to re-enter the data. It is recommended that an error/prompt message be displayed in an out-of-the-way place, informing the operator that the

specified field has failed all validation tests and that it must be re-entered.

9 -- Print local: Write the contents of the entire screen to the local printer attached to the terminal; this feature must be supported by the particular terminal hardware in use. The device driver should return to the caller when the printer has completed printing.

Iolist is an array that contains the control and data definitions for each field in the form. It contains seven header words and at least one data word for each entry. The array should be accessed by the device driver using a pointer to the beginning of the field (supplied by the run-time package) added to an offset. This offset should be specified in the form of a PARAMETER'ed symbol, as defined below.

The following PARAMETERS represent each of the control words, plus the start of the data area. The device driver should be oblivious to their actual values, as these may change when new control information is addded. The parameter declarations may be made through an $INSERT file called 'IOPARM' in the directory containing the source of the I/O system (as released, FORMS>IOS>IOPARM).

IOLK -- Link to next entry in chain by position; this is not used by device drivers.

IOVP -- Stream definition field pointer for this entry; this is not used by device drivers.

IORP -- Format definition field pointer for this entry; this is not used by device drivers.

IOSZ -- Field length, in characters.

IOAT -- Field attributes, as follows:

Bit	Definition
1	Set by FORMS if data has changed since last display. Reset be device driver when data has been updated on device.
2	Set by FORMS if enable/protect attribute has changed since last display. Reset by device driver when field has been updated on device.
3	Set by FORMS if field has failed all supplied validation tests. Reset by device driver when field has been re-entered from device.

4	Set be FORMS if any field attributes has been modified since last display. Reset by device driver when field has been updated on device.
5	Set by device driver if field is currently displayed on device. Reset by device driver if field is currently not displayed on device (initially reset).
13	Set by FORMS if field should be blinked when displayed. Reset by FORMS if field should not be blinked when displayed.
14	Set by FORMS if field should not be displayed or should be erased if currently displayed. Reset by FORMS if field should be displayed.
15	Set by FORMS if field should be displayed in reverse video. Reset by FORMS if field should be displayed in normal video.
16	Set by FORMS if field should be write-enabled (not protected). Reset by FORMS if field should be write-protected.

IOYX -- line and column coordinates:
. left byte = line # (Y)
. right byte = column # (X)

IOPG -- physical page #; this is not used by device drivers

IODA -- start of text data; data is in ascii format, packed two characters per word, blank filled

The initialize, clear, close, and print functions (1, 5, 6, 7, and 9) are all relatively straightforward. These operations do not have to process data from the I/O list and therefore should assume it to be void.

The output, input, modify, and correct functions (2, 3, 4, and 8) all need to traverse the I/O list and process (or at least inspect) each field therein. The device driver must depend on the run-time system to provide a pointer for the start of each field definition in the I/O list. The run-time package contains two subroutines callable by the device driver for such a purpose. They are:

FMS$RE - Resets the internal (run-time package) field pointer to the beginning of the current page. This routine must be called at the beginning of the output, input, modify, and correct-data function processors. It may be called again to reset the pointer to the first field in the page when necessary (eg, on an input error).

Calling Sequence:

```
CALL FMS$RE
```

FMS$NF - Returns the pointer to the next field in the I/O list to be processed. if the pointer is 0, the end of page or end of I/O list has been encountered. Fields are returned to the caller in line/column sequence.

Calling Sequence:

```
CALL FMS$NF ( pointer )
```

For a sample I/O driver, the reader is referred to the VISTAR3 device driver in the input/output system source directory.

7.2.3 Error Message Generation

The user is allowed to generate run-time error messages from within the device driver, should he encounter an error condition. A subroutine called 'RTERR$' exists in the run-time package, which may be called to log an error in the error file and set up the user's error vector. The calling sequence is as follows:

```
CALL RTERR$ (enum, text, textsz, name)
```

enum	=	error number (see below)
text	=	text error message (2 characters/wd)
textsz	=	length of 'text', in characters
name	=	routine name (8 characters) or 0 if none

The user is allowed to use any error number between 60 and 89, inclusive. All others are reserved for future use by Prime. Should the user wish to exit to PRIMOS command level, he need only call the ERRSET subroutine as follows:

```
CALL ERRSET(:100000,0)
```

This causes the text message passed as an argument to RTERR$ to be typed on the user terminal, followed by the 'ER!' prompt from PRIMOS. If the error condition was non-fatal, the device driver should not call ERRSET, but rather perform whatever error recovery is necessary to continue.

7.2.4 Installing the Device Driver

To install a new device driver into the FORMS run-time library, the user should follow the steps outlined below:

a) Obtain a listing of the device control file and choose a free logical unit number above 10 (the first 10 are reserved by Prime). Append an entry to the DCF containing the selected logical unit number, device name, first two characters of the driver name (remember, the last three must be '$IO'), and the dimensions of the device in accordance with the format described in the section entitled "Device Mapping Scheme", above. For example, the Vistar/3 entry, whose logical unit number is 3, driver name is 'V3$IO', and dimensions are 24 by 80, would look as follows:

```
3, VISTAR3, V3, 24, 80
```

b) Attach to the directory containing the source for the Input/Output System and copy into it the source for the device driver to be installed.

c) Edit the C_IOR (64R mode) and C_IOV (64V mode) command files and insert a line to compile the new device driver after the PR$IO routine.

d) Run FAP and issue the GENERATE command to create the new device tables which will include the new driver.

e) Execute the C_IOR and/or C_IOV command file(s) to create a new input/output system.

f) Attach to the first-level FORMS source directory ('FORMS') and execute the command file 'C_RFMS' to create a 64R mode library and/or C_VFMS to create a 64V mode library.

The user may now modify the TCB entries for the users which have the new terminal and reload his applications program with the new version of the library. It is strongly recommended that the new library not be installed in the 'LIB' UFD until the new device driver has been proven to work.

7.3 Prime-Supplied Device Drivers

At present, the FORMS system as released by Prime supports the following two device drivers:

. Offline printer	(PRINTER)
. Vistar/3 (modified)	(VISTAR3)

7.3.1 Offline Printer Device Driver

The PRINTER device driver writes a form (or forms) to the line-printer spool queue. When the INVOKE command is issued to the line-printer (IOCS logical unit 4), a file called PR##nn (where 'nn' represents the user number) is opened. If it already exists, the file pointer is positioned to the end of file, where the new form definition will be written. If it does not exist, it is created, after a record is written containing the control code for the line-printer to enter Fortran forms-control mode.

When a form is output, one ASCII record is written for each line defined in the form. The first line contains a '1' in column 1, which causes the printer to eject to the top of a new page. Any enabled fields are underscored (with the '_' character).

When the form is released, the file is copied into the spool queue, with the appropriate spool file header and file name. It is then closed and deleted from the home UFD. Note that the PR##nn file should never appear in the home UFD after the program has been completed; if it does, it means that the PRINTER form was not released.

7.3.2 Vistar/3 Device Driver

The Infoton Vistar/3 device driver (V3$IO) is written for a specially modified Vistar/3 (with microcode and hardware updates) available through Prime.

The device dimensions are 24 lines by 80 columns (1920 characters), all of which except the 15 character positions in the lower right of the screen are available for use by the applications program. These character positions contain one of the following prompt or error messages from the device driver:

(spaces):
Input not allowed.

ENTER
Enter data into unprotected fields on form, depress 'XMIT PAGE' key when done.

ERROR, RE-ENTER (blinking)
A character was lost on the last transmission - depress 'XMIT PAGE' key.

DATA ERROR (reverse video)
A field (or fields) does not meet the specified

validation criteria - the cursor is positioned to the first character position of the offending field. Correct the data and depress the 'XMIT PAGE' key.

All unprotected fields are displayed in full - intensity, surrounded by square brackets ('[' and ']'). All protected fields are displayed in half-intensity. Note that care must be taken to allow for the square brackets on unprotected fields when designing the form.

To operate the Vistar/3 with a program using FORMS, the dip-switches in the rear of the display must be set as follows:

EOT character:	CR
Mode:	Block
Line-speed:	(User-selectable)
Sec channel:	Off
Parity:	None
Fdux/hdux:	(User-selectable)
Stop bits:	Two
Roll/page:	Roll

APPENDIX A

SAMPLE FORM DEFINITION SOURCE

```
*   ESFORM, FORMS, JRW, 76/12/30     stream descriptor definition
*   Employee Status Form Definition - **FORMS** Demonstration
*   This form definition is intended for use with 'EMPUPD'.
*
*
*
*    ******************************************
*    *                                        *
*    *   s t r e a m   d e f i n i t i o n    *
*    *                                        *
*    ******************************************
*
*
*---Macro Definitions File  ( FORMS > MACROS )
$INSERT FORMS>MACROS
*
*
*---stream definition for employee status form
*   last updated: 30-Dec-76
*
estatus  stream
*
*---header information (name, employee & dept #, marital status)
header   substream
         f name, len 30
empn     f empn, len 4, jus r, v 'N'
         f deptn, len 3, jus r, v 'N'
         f marital, len 1, v 'A' or 'B'
         end substream
*
*---address, phone numbers
addrfone substream
         f street, len 30
         f city, len 20
         f state, len 2, v 'AA' or 'B'
         f zipcode, len 5, v '9' or 'B'
*
         f homefone, len 12, v '999-999-9999' or 'B'
         f workfone, len 12, output
         f (workfone,'212-587-1234'), len 12, in ;
           v '999-999-9999' or 'B'
         f ext, len 3, v '9' or 'B'
         end substream
*
*---start, termination, and return dates
```

```
dates    substream
         f strtdate, len 8, v '99/99/99' or 'B'
         f termdate, len 8, v '99/99/99' or 'B'
         f retndate, len 8, v '99/99/99' or 'B'
         end substream
*
*---work status information
wstatus  substream
         f fullpart, len 1, v 'A' or 'B'
         f permtemp, len 1, v 'A' or 'B'
         f direct, len 1, v 'A' or 'B'
         f workstat, len 1, v 'A' or 'B'
         end substream
*
*---job title, code
jobinf   substream
         f jobtitle, len 20
         f jobcode, len 2
         end substream
*
*---salary information
salryinf substream
         f lastincr, len 8, v '99/99/99' or 'B'
         f amtincr, len 7, v 'N'
         f perincr, len 3, v 'N'
         f salary, len 8, v 'N'
         f speriod, len 1, v 'A' or 'B'
         end substream
*
*---error message facility
errmesg  substream
error    f error, len 30, output
         end substream
*
*
*
         end stream
```

```
*   ESFORM, FORMS, JRW, 76/12/30     format descriptor definition
*
*
*
*    ******************************************
*    *                                        *
*    *   f o r m a t   d e f i n i t i o n    *
*    *                                        *
*    ******************************************
*
*
*---Macro definition file ( FORMS > MACROS )
$INSERT FORMS>MACROS
*
*
*---device format definition for VISTAR/3 terminal
*   last updated: 30-Dec-76
*
estatus  format
         device vistar3
*
*---literal form header
         f 'Employee Status Form' pos (32,1)
*
*---header line >> employee name, id#, dept#
         f 'name' pos (1,3)
name     f len 30, pos (7,3), np
         f 'emp#' pos (47,3)
empn     f len 4, pos (53,3), np
         f 'dept#' pos (68,3)
deptn    f len 3, pos (75,3), np
*
*---marital information
         f 'marital status' pos (1,5)
marital  f len 1, pos (17,5), np
         f 'S=single, M=married, D=divorced, W=widowed' pos (20,5)
*
*---address information
         f 'address: street' pos (1,7)
street   f len 30, pos (18,7), np
         f 'city' pos (10,8)
city     f len 20, pos (18,8), np
         f 'state' pos (41,8)
state    f len 2, pos (49,8), np
         f 'zip' pos (54,8)
zipcode  f len 5, pos (60,8), np
*
*---telephone numbers...
         f 'telephone #: home' pos (1,10)
homefone f len 12, pos (20,10), np
         f 'business' pos (35,10)
workfone f len 12, pos (45,10), np
         f 'extension' pos (60,10)
```

```
ext      f len 3, pos (71,10), np
*
*---start date, termination date, return date
         f 'start date' pos (1,12)
strtdate f len 8, pos (13,12), np
         f 'term date' pos (24,12)
termdate f len 8, pos (35,12), np
         f 'return date' pos (46,12)
retndate f len 8, pos (59,12), np
*
*---work status
         f 'status:' pos (1,14)
fullpart f len 1, pos (10,14), np
         f 'F=full, P=part', pos (13,14)
permtemp f len 1, pos (38,14), np
         f 'P=permanent, T=temporary', pos (41,14)
direct   f len 1, pos (10,15), np
         f 'D=direct, I=indirect', pos (13,15)
workstat f len 1, pos (38,15), np
         f 'A=active, L=leave, T=terminated', pos (41,15)
*
*---job information
         f 'job title', pos (1,17)
jobtitle f len 20, pos (12,17), np
         f 'job code', pos (36,17)
jobcode  f len 2, pos (46,17), np
*
*---salary information
         f 'date last increase', pos (1,19)
lastincr f len 8, pos (21,19), np
         f 'amount', pos (33,19)
amtincr  f len 7, pos (41,19), np
         f 'percent', pos (52,19)
perincr  f len 3, pos (61,19), np
*
         f 'salary', pos (1,21)
salary   f len 8, pos (9,21), np
speriod  f len 1, pos (25,21), np
         f 'H=hourly, W=weekly, Y=yearly', pos (28,21)
*
*---error message field
error    f len 30, pos (1,24), blink
*
*
         end device
         end format
```

APPENDIX B

SAMPLE FORM DEFINITION

```
*   ESFORM, FORMS, JRW, 76/12/30  stream descriptor definition
*   Employee Status Form Definition - **FORMS** Demonstration
*   This form definition is intended for use with 'EMPUPD'.
*
*
*
*    *******************************************
*    *                                         *
*    *    s t r e a m   d e f i n i t i o n    *
*    *                                         *
*    *******************************************
*
*
*---Macro Definitions File  ( FORMS > MACROS )
*---Start FORMS>MACROS
*   $INSERT file for standard syntax abbreviations
*
*
f        def field
v        def validate
len      def length
pos      def position
in       def input
out      def output
jus      def justify
r        def right
l        def left
c        def center
np       def noprotect
*
*---End FORMS>MACROS
*
*
*---stream definition for employee status form
*   last updated: 30-Dec-76
*
estatus  stream
*
*---header information (name, empl & dept#, marital status)
header   substream
         f name, len 30
empn     f empn, len 4, jus r, v 'N'
         f deptn, len 3, jus r, v 'N'
         f marital, len 1, v 'A' or 'B'
         end substream
*
*---address, phone numbers
```

```
addrfone substream
         f street, len 30
         f city, len 20
         f state, len 2, v 'AA' or 'B'
         f zipcode, len 5, v '9' or 'B'
*
         f homefone, len 12, v '999-999-9999' or 'B'
         f workfone, len 12, output
         f (workfone,'212-587-1234'), len 12, in ;
           v '999-999-9999' or 'B'
         f ext, len 3, v '9' or 'B'
         end substream
*
*---start, termination, and return dates
dates    substream
         f strtdate, len 8, v '99/99/99' or 'B'
         f termdate, len 8, v '99/99/99' or 'B'
         f retndate, len 8, v '99/99/99' or 'B'
         end substream
*
*---work status information
wstatus  substream
         f fullpart, len 1, v 'A' or 'B'
         f permtemp, len 1, v 'A' or 'B'
         f direct, len 1, v 'A' or 'B'
         f workstat, len 1, v 'A' or 'B'
         end substream
*
*---job title, code
jobinf   substream
         f jobtitle, len 20
         f jobcode, len 2
         end substream
*
*---salary information
salryinf substream
         f lastincr, len 8, v '99/99/99' or 'B'
         f amtincr, len 7, v 'N'
         f perincr, len 3, v 'N'
         f salary, len 8, v 'N'
         f speriod, len 1, v 'A' or 'B'
         end substream
*
*---error message facility
errmesg  substream
error    f error, len 30, output
         end substream
*
*
*
         end stream

0000 ERRORS (FDL, REV 12.0 - PRE-RELEASE)
```

I N P U T S T R E A M D E S C R I P T O R :

SUBSTREAM	FIELD	LENGTH	START
1	NAME	30	1
1	EMPN	4	31
1	DEPTN	3	35
1	MARITAL	1	38
2	STREET	30	1
2	CITY	20	31
2	STATE	2	51
2	ZIPCODE	5	53
2	HOMEFONE	12	58
2	WORKFONE	12	70
2	EXT	3	82
3	STRTDATE	8	1
3	TERMDATE	8	9
3	RETNDATE	8	17
4	FULLPART	1	1
4	PERMTEMP	1	2
4	DIRECT	1	3
4	WORKSTAT	1	4
5	JOBTITLE	20	1
5	JOBCODE	2	21
6	LASTINCR	8	1
6	AMTINCR	7	9
6	PERINCR	3	16
6	SALARY	8	19
6	SPERIOD	1	27

O U T P U T S T R E A M D E S C R I P T O R :

SUBSTREAM	FIELD	LENGTH	START
1	NAME	3Ø	1
1	EMPN	4	31
1	DEPTN	3	35
1	MARITAL	1	38
2	STREET	3Ø	1
2	CITY	2Ø	31
2	STATE	2	51
2	ZIPCODE	5	53
2	HOMEFONE	12	58
2	WORKFONE	12	7Ø
2	EXT	3	82
3	STRTDATE	8	1
3	TERMDATE	8	9
3	RETNDATE	8	17
4	FULLPART	1	1
4	PERMTEMP	1	2
4	DIRECT	1	3
4	WORKSTAT	1	4
5	JOBTITLE	2Ø	1
5	JOBCODE	2	21
6	LASTINCR	8	1
6	AMTINCR	7	9
6	PERINCR	3	16
6	SALARY	8	19
6	SPERIOD	1	27
7	ERROR	3Ø	1

```
*   ESFORM, FORMS, JRW, 76/12/30  format descriptor definition
*
*
*
*    ******************************************
*    *                                        *
*    *   f o r m a t   d e f i n i t i o n    *
*    *                                        *
*    ******************************************
*
*
*---Macro definition file ( FORMS > MACROS )
*---Start FORMS>MACROS
*   $INSERT file for standard syntax abbreviations
*
*
f         def field
v         def validate
len       def length
pos       def position
in        def input
out       def output
jus       def justify
r         def right
l         def left
c         def center
np        def noprotect
*
*---End FORMS>MACROS
*
*
*---device format definition for VISTAR/3 terminal
*   last updated: 30-Dec-76
*
estatus   format
          device vistar3
*
*---literal form header
          f 'Employee Status Form' pos (32,1)
*
*---header line >> employee name, id#, dept#
          f 'name' pos (1,3)
name      f len 30, pos (7,3), np
          f 'emp#' pos (47,3)
empn      f len 4, pos (53,3), np
          f 'dept#' pos (68,3)
deptn     f len 3, pos (75,3), np
*
*---marital information
          f 'marital status' pos (1,5)
marital   f len 1, pos (17,5), np
          f 'S=single, M=married, D=divorced, W=widowed'
            pos (20,5)
```

```
*
*---address information
         f 'address: street' pos (1,7)
street   f len 30, pos (18,7), np
         f 'city' pos (10,8)
city     f len 20, pos (18,8), np
         f 'state' pos (41,8)
state    f len 2, pos (49,8), np
         f 'zip' pos (54,8)
zipcode  f len 5, pos (60,8), np
*
*---telephone numbers...
         f 'telephone #: home' pos (1,10)
homefone f len 12, pos (20,10), np
         f 'business' pos (35,10)
workfone f len 12, pos (45,10), np
         f 'extension' pos (60,10)
ext      f len 3, pos (71,10), np
*
*---start date, termination date, return date
         f 'start date' pos (1,12)
strtdate f len 8, pos (13,12), np
         f 'term date' pos (24,12)
termdate f len 8, pos (35,12), np
         f 'return date' pos (46,12)
retndate f len 8, pos (59,12), np
*
*---work status
         f 'status:' pos (1,14)
fullpart f len 1, pos (10,14), np
         f 'F=full, P=part', pos (13,14)
permtemp f len 1, pos (38,14), np
         f 'P=permanent, T=temporary', pos (41,14)
direct   f len 1, pos (10,15), np
         f 'D=direct, I=indirect', pos (13,15)
workstat f len 1, pos (38,15), np
         f 'A=active, L=leave, T=terminated', pos (41,15)
*
*---job information
         f 'job title', pos (1,17)
jobtitle f len 20, pos (12,17), np
         f 'job code', pos (36,17)
jobcode  f len 2, pos (46,17), np
*
*---salary information
         f 'date last increase', pos (1,19)
lastincr f len 8, pos (21,19), np
         f 'amount', pos (33,19)
amtincr  f len 7, pos (41,19), np
         f 'percent', pos (52,19)
perincr  f len 3, pos (61,19), np
*
         f 'salary', pos (1,21)
```

```
salary   f len 8, pos (9,21), np
speriod  f len 1, pos (25,21), np
         f 'H=hourly, W=weekly, Y=yearly', pos (28,21)
*
*---error message field
error    f len 30, pos (1,24), blink
*
*
         end device
         end format

0000 ERRORS (FDL, REV 12.0 - PRE-RELEASE)
```

APPENDIX C

SAMPLE FTN PROGRAM USING FORMS

```
C   EMPUPD, FORMS-DEMO, JRW, 77/01/02
C   **FORMS** DEMONSTRATION PROGRAM - EMPLOYEE STATUS UPDATE
C
C
C
C---THIS PROGRAM DEMONSTRATES SOME OF THE CAPABILITIES OF THE
C   PRIME FORMS MANAGEMENT SYSTEM.
C
C---OPERATION:
C
C    FROM COMMAND LEVEL:   R *EMPUP
C
C      . TO CREATE A NEW ENTRY (NEW EMPLOYEE):
C
C        ENTER THE EMPLOYEE NAME INTO THE NAME FIELD AND
C        DEPRESS THE TRANSMIT KEY.
C
C        THE EMPLOYEE WILL BE ASSIGNED THE NEXT SEQUENTIALLY
C        AVAILABLE ID# FROM THE CONTROL FILE (E*CTRL).
C        THE OPERATOR MAY THEN PROCEED TO ENTER THE DATA
C        FOR THE NEW EMPLOYEE INTO THE FIELDS PROVIDED.
C        WHEN DONE, DEPRESS THE TRANSMIT KEY TO UPDATE
C        THE KI/DA FILE.  TO IGNORE THE UPDATE, CLEAR THE
C        EMPLOYEE NAME FIELD AND DEPRESS THE TRANSMIT KEY.
C
C
C      . TO UPDATE AN ALREADY-EXISTING ENTRY:
C
C        ENTER EITHER THE EMPLOYEE NAME INTO THE EMPLOYEE
C        NAME FIELD OF THE EMPLOYEE ID# INTO THE ID# FIELD
C        AND DEPRESS THE TRANSMIT KEY.  THE KI/DA FILE WILL
C        BE SCANNED FOR THE GIVEN ENTRY AND, IF FOUND, THE
C        DATA FOR SAID EMPLOYEE WILL BE DISPLAYED ON THE SCREEN.
C        IF NOT FOUND, A SUITABLE ERROR MESSAGE WILL BE DISPLAYED
C        IN THE LOWER LEFT-HAND CORNER OF THE CRT.
C
C        WHEN THE USER HAS MODIFIED THE DESIRED DATA, HE MAY
C        DEPRESS THE TRANSMIT KEY TO UPDATE THE KI/DA FILE -OR-
C        HE MAY CLEAR THE EMPLOYEE NAME FIELD IF HE DOES NOT
C        WISH TO UPDATE THE FILE.
C
C
C      . TO EXIT THE PROGRAM, CLEAR BOTH EMPLOYEE NAME AND EMPLOYEE
C        ID# FIELDS (LEAVE THEM BLANK) AND HIT THE TRANSMIT KEY.
```

```
C
C
C
C
C   TO INITIALIZE THE DATA AND CONTROL FILES, THE USER CAN RUN THE
C    'C_DATA' COMMAND FILE.
C
C
C
$INSERT EQUIV
C
$INSERT SYSCOM>FILD.F
$INSERT SYSCOM>ERRD.F
C
C
       INTEGER VERRET, ERRCOD, ARY(14), XEMPN, I, VTBL(18)
C
C
C---INITIALIZE PROGRAM:
C   INVOKE FORM, OPEN DATA FILES.
C
      WRITE(1,100)
100   FORMAT('##INVOKE ESTATUS')
      CALL SEARCH (OPNRED, 'E*DATA', 1)
      CALL SEARCH (OPNBTH, 'E*CTRL', 2)
      GO TO 500                              /* READ INITIAL DATA
C
C
C---HERE TO CLEAR ERROR MESSAGE FIELD AND UNPROTECTED
C   DATA TO PROCESS NEXT EMPLOYEE.
C
300   WRITE(1,310)
310   FORMAT('##NOPROTECT EMPN'/
     +        '##SUBSTREAM ERRMESG'/
     +        /
     +        '##CLEAR')
C
C
C---INPUT DATA FOR NEXT EMPLOYEE.
C
500   ASSIGN 510 TO VERRET                   /* VALIDATION ERROR RETURN
      GO TO 530
C
C
C---HERE FOR VALIDATION ERROR RECOVERY WHILE
C   READING HEADER RECORD.
C
510   WRITE(1,520)                           /* RE-INPUT 1ST SUBSTREAM
520   FORMAT('##SUBSTREAM HEADER')
C
530   READ(1,540,ERR=90000) NAME,EMPN,DEPTN,MARTAL
540   FORMAT(15A2,I4,I3,A1)
C
```

```
C
C---EXIT IF BOTH NAME AND EMPLOYEE # FIELDS ARE EMPTY.
C
      IF(NAME(1).NE.'  '.OR.EMPN.NE.0) GO TO 600
      WRITE(1,550)
550   FORMAT('##CLEAR ALL'/'##RELEASE')
      CALL SEARCH(CLOSE, 0, 1)
      CALL SEARCH(CLOSE, 0, 2)
      CALL SEARCH(CLOSE, 0, 3)
      CALL EXIT
C
C
C---HERE IF EITHER EMPLOYEE# OR NAME SPECIFIED.
C
600   IF(EMPN.EQ.0) GO TO 2000
C
C
C---EMPLOYEE ID # SUPPLIED - LOOK UP ON INDEX 0.
C
      CALL FIND$(1,FILBUF,EMPN,ARY,:40000,$80000,0,0,0,0)
C
C
C---UPDATE FORM WITH DATA READ FROM FILE.
C
700   WRITE(1,710)
710   FORMAT('##PROTECT EMPN'/
     +        '##SUBSTREAM HEADER')
C
800   WRITE(1,810) FILBUF
810   FORMAT(
     +        15A2,B'####',B'###',A1/
     +        25A2,A2,2A2,A1,12A2,B'###'/
     +        12A2/
     +        4A1/
     +        10A2,A2/
     +        4A2,3A2,A1,B'###',4A2,A1/
     +        /
     + )
C
C
C---READ ANY UPDATED DATA FROM THE SCREEN.
C   THE OPERATOR AT THIS POINT HAS THE FOLLOWING OPTIONS:
C
C     A) CHANGE ANY DATA ANY HIT XMIT - THE UPDATES WILL BE MADE
C        TO THE KI/DA FILE.
C     B) CLEAR THE NAME FIELD (WITH THE 'CLEAR FIELD' KEY)
C        THIS WILL CAUSE NO UPDATE TO OCCUR.
C
1000  ASSIGN 1005 TO VERRET
1005  READ(1,1100,ERR=90000) FILBUF
1100  FORMAT(
     +          15A2,I4,I3,A1/
     +          25A2,A2,2A2,A1,12A2,I3/
     +          12A2/
     +          4A1/
     +          10A2,A2/
     +          4A2,3A2,A1,I3,4A2,A1
     + )
C
       IF(NAME(1).EQ.' ') GO TO 300          /* IGNORE UPDATE
C
C
C---UPDATE THE KI/DA FILE WITH THE USER'S CHANGES.
C   THIS IS DONE BY DELETING THE ENTRY BY DELETING THE PRIMARY
C   INDEX ENTRY AND THEREFORE DELETING ALL RELATED SECONDARY INDEX
C   ENTRIES.  THE DATA IS THEN ADDED VIA 'ADD1$'.
C
       CALL DELET$(1,FILBUF,0,ARY,:100000,0,0,0,0,0)
       CALL ADD1$(1,FILBUF,EMPN,ARY,:40000,$81000,0,0,0,0)
       CALL ADD1$(1,0,NAME,ARY,:100000,$81000,1,0,0,0)
       CALL ADD1$(1,0,DEPTN,ARY,:100000,$81000,2,0,0,0)
       CALL ADD1$(1,0,EXT,ARY,:100000,$81000,3,0,0,0)
C
       GO TO 300
C
C
C---NAME SUPPLIED WITH NO EMPLOYEE # -
C   FIRST LOOKUP IN FILE - IF NOT FOUND, GIVE THE OPERATOR
C   THE OPTION OF ADDING THE NEW EMPLOYEE.
C
2000   CALL FIND$(1,FILBUF,NAME,ARY,:40000,$4000,1,0,0,0)
       CALL FIND$(1,FILBUF,EMPN,ARY,:40000,$81000,0,0,0,0)
       GO TO 700                                /* FOUND...
C
C
C---NEW EMPLOYEE - FIRST GET NEXT AVAILABLE ID# FROM CONTROL
C   FILE AND DISPLAY IN EMPLOYEE # FIELD.  IF THE OPERATOR
C   LEAVES THIS FIELD ALONE, THE EMPLOYEE WILL BE ADDED TO THE
C   FILE.  IF THIS FIELD IS CLEARED, THE OPERATION IS IGNORED.
C
4000   CALL SEARCH(REWIND, 0, 2)
       READ(6,4010) EMPN
4010   FORMAT(I5)
       XEMPN=EMPN
C
       WRITE(1,4050) NAME, EMPN
4050   FORMAT(15A2,B'####')
C
       ASSIGN 4055 TO VERRET
```

```
4055   READ(1,1100,ERR=90000) FILBUF
       IF(EMPN.EQ.0.OR.NAME(1).EQ.'  ') GO TO 300
       CALL ADD1$(1,FILBUF,EMPN,ARY,:40000,$81000,0,0,0,0)
       CALL ADD1$(1,0,NAME,ARY,:100000,$81000,1,0,0,0)
       CALL ADD1$(1,0,DEPTN,ARY,:100000,$81000,2,0,0,0)
       CALL ADD1$(1,0,EXT,ARY,:100000,$81000,3,0,0,0)
C
C
C---UPDATE EMPLOYEE NUMBER IN CONTROL FILE IF THE
C   USER DIDN'T CHANGE IT ON US.
C
       IF(EMPN.NE.XEMPN) GO TO 300
       CALL SEARCH(REWIND, 0, 2)
       EMPN=EMPN+1
       WRITE(6,4090) EMPN
4090   FORMAT(B'#####')
       GO TO 300
C
C
C---HANDLE 'FIND' ERROR FROM KI/DA -
C
80000  IF(ARY(1).NE.7) GO TO 80500
       WRITE(1,80010) EMPN
80010  FORMAT('##SUBSTREAM ERRMESG'/
     +         'Employee ',B'####',' not on file')
C
C
C---HERE TO SET UP SCREEN AFTER ERROR.
C   THIS WILL CLEAR THE UNPROTECTED DATA, WRITE-ENABLE
C   THE EMPLOYEE # FIELD, AND RETURN TO ACCEPT INPUT FROM
C   THE OPERATOR.
C
80050  WRITE(1,80060)
80060  FORMAT('##NOPROTECT EMPN'/
     +         '##CLEAR')
       GO TO 500
C
C
C---HERE TO CHECK FOR DUPLICATE ENTRY ERROR
C
80500  IF(ARY(1).NE.12) GO TO 81000
       WRITE(1,80510) EMPN
80510  FORMAT('##SUBSTREAM ERRMESG'/
     +         'Employee # ',B'####',' already on file')
       GO TO 80050
C
C
C---HERE ON ANY UNEXPECTED KI/DA FILE HANDLER ERROR.
C
81000  WRITE(1,81010) ARY(1)
81010  FORMAT('KI/DA Err# ',B'##')
       GO TO 500
C
```

```
C
C---HERE OF VALIDATE ERROR FROM READ STATEMENT
C
90000  CALL GETERR(ERRCOD,1)
       IF(ERRCOD.NE.2HVA) GO TO 90050
       WRITE(1,90010)
90010  FORMAT('##SUBSTREAM ERRMESG'/'Validate Error')
       GO TO VERRET
C
90050  IF(ERRCOD.NE.2HFD) GO TO 90100
       WRITE(1,90060)
90060  FORMAT('##SUBSTREAM ERRMESG'/'Format/Data Mismatch')
       GO TO VERRET
C
90100  WRITE(1,90120) ERRCOD
90120  FORMAT('##SUBSTREAM ERRMESG'/'Error "',A2,'"')
       GO TO VERRET
C
C
C
       END
```

APPENDIX D

FORMS CONTROL DIRECTORY FORMATS

The FORMS segment directory consists of three control files and a variable number of user form definitions. The file formats are described below:

- Module Name File (MNF) - Segment 0

  This file contains the name and type (stream or format) of each form in the directory. Entry length is five words and file length is variable (will accommodate any number of entries).

  Entry Format:

Offset	Definition
0-3	Form name (left adjusted, right padded)
4	Entry type (1=stream, 2=format)

- Module Information File (MIF) - Segment 1

  The MIF contains information related to each entry in the directory. Each module information file entry has a corresponding module name file entry. They are the same entry number, but, because of the difference in entry sizes, they are not the same offset into the two respective files. Entry length is 32 words and file length is variable.

  Entry Format:

Offset	Definition
0-3	Device type (left adjusted, right padded)
4	Version #
5-9	Creation date/time
10-14	Modify date/time
15-19	Access date/time
20	Segment directory entry #
21-31	Reserved for future expansion

- Terminal Configuration Block (TCB) - Segment 2

The TCB contains the terminal name for each FORMS terminal on the the system. The file is a 64 by 4 matrix, allowing a 1-8 character terminal name for each user on a 16, 32, or 64 user system. The file is fixed length, 256 words.

File Format:

Offset	Definition
Ø-3	user 1 terminal type
4-7	user 2 terminal type
8-11	user 3 terminal type
	.
	.
	.
252-255	user 64 terminal type

Note: If the terminal type for a user is undefined, the first word of the entry is Ø.

INDEX

INDEX