Price: $5.00

TYMSHARE MANUALS
TYMCOM-IX

Information Management Library

NOVEMBER 1975

TYMSHARE, INC.
CUPERTINO, CALIFORNIA 95014

RH3

CONTENTS

SECTION 1
INTRODUCTION

The accumulation of information at a geometric rate is a fact of contemporary organizations. Perhaps even more significant is the inevitable task of organizing and utilizing the information. With this objective foremost in mind, Tymshare has developed an interrelated group of packages which organize, access, update, and quantify in a multitude of ways any information stored on computer-associated devices.

The Information Management Library, IML, provides an extensive set of data handling capabilities tailored to the TYMCOM-IX, resulting in unsurpassed performance and economy. At the same time, the Information Management Library maintains the traditionally simple syntax of Tymshare's software, offering the ease of use, straightforward interaction, and flexibility our users have come to appreciate.

The library consists of 11 procedures, which encompass and organize the functional requirements of a data base management system. Using a modest vocabulary of English words and phrases common to all Information Management Library activities, each package accomplishes a separate task.

- DEFINE handles the design and description of a data base, permitting optimum storage and access.
- CREATE facilitates data entry with an assortment of convenient features, allowing free format, optional prompts, empty fields, and the ability to append information to an existing data base. In addition, it incorporates an extensive editor similar to Tymshare's EDITOR.
- VERIFY permits the user to check the accuracy and completeness of a data base, optionally allowing the user to enter corrected information directly at the terminal when an error is located.
- SORT reorders the information in a data base according to the user's specifications, permitting mixed ascending or descending sorts on as many as 20 items.
- MERGE combines two sorted data bases into a single sorted data base.
- PURGE deletes information according to conditions specified by the user.
- REPLACE provides for the substitution of records within a data base, permitting complete or partial replacement of records.
- UPDATE assists in modifying a data base, allowing elaborate mathematical and conditional instructions.
- CONVERT allows for conversion of binary and symbolic files. The user may change the data type of the file, rearrange fields, and change fixed length records to variable length records (or the reverse).

- SELECT is a mini-information retrieval package enabling the user to extract records or fields based on user-specified conditions. It also offers a unique capability for reformatting an entire data base.
- REPORT provides a comprehensive reporting facility suitable for organizing and generating reports of any description, including those on preprinted forms. It incorporates a host of optional features, such as picture formats, running totals, and built-in functions.

Each IML procedure is called directly from the EXECUTIVE by typing its name after the EXECUTIVE dash (–).

The Information Management Library joins RETRIEVE, Tymshare's general-purpose retrieval system; STATPAK, the comprehensive statistics package; and TYMTAB, Tymshare's dynamic modeling tool, to form an integrated data management capability. Maintaining complete compatibility and transference, RETRIEVE, STATPAK, TYMTAB, and the Information Management Library offer an unmatched combination to meet virtually all information processing needs.

ABOUT THIS MANUAL

Section 2 introduces the basic concepts and design of the Information Management Library. The section begins with a sample application using several different commands to demonstrate the capabilities of IML, then discusses the components of the commands, which are the framework for understanding and utilizing the Information Management Library. By first studying this section, the user can learn the basic information necessary to use IML.

Section 3 presents the basic concepts of rules files. It discusses the simple command-level instructions for rules file creation, alteration, and execution. The section also details the statements common to all IML rules files, and outlines the structure of the VERIFY, UPDATE, and REPORT rules files.

Section 4 details the IML commands for designing, creating, and checking data files. Section 5 presents the commands for file maintenance; these include commands for sorting, merging, selecting, deleting, replacing, and updating records in IML or RETRIEVE data files.[1] Section 6 describes the IML reporting facility.

Various advanced features of IML are presented in Section 7. Topics include the use of command files, SUPER BASIC and IML, reformatting a data base, and numerous optional features of the DEFINE program.

Appendix A is an IML command summary. Appendix B contains a list of IML error messages.

SYMBOL CONVENTIONS

To indicate clearly the interaction at the terminal, everything typed by the user is underlined. The symbols for user-typed characters are:

Carriage Return:	↩
Line Feed:	↴
Alt Mode/Escape:	⊕

Control characters are denoted by a superscript c. For example, Y^C denotes Control Y. The method for typing a control character depends on the type of terminal being used. Consult the literature for your particular terminal or see your Tymshare representative.

1 – See the *Tymshare TYMCOM-IX RETRIEVE Reference Manual* for complete documentation of RETRIEVE.

When a general form of an IML command is presented, lowercase letters represent the actual information to be typed. For example, *description file name* in the command

```
         description
-DEFINE  file        ↵
         name
```

specifies that the user types the name of the description file at that point.

Braces in a command form indicate a choice of one of the items enclosed in the braces. The braces are not part of the command. For example,

```
         data   description
-CREATE  file : file         FROM  {        T        }
         name   name               {input file name  }↵
```

indicates that the user may specify T or an input file name as part of the command.

NOTE: When square brackets appear in a command form, the brackets are actually typed by the user. For example, in the command

```
-SORT [VERSION]↵
```

the user must type the brackets that enclose the word VERSION. On some terminals, the user types a Shift K and a Shift M for left and right square brackets, respectively. On some terminals, the bracket characters are shown on the terminal keyboard.

USER INTERACTION

Whenever IML requests the name of a file on which to write the results of a command, the system prints a NEW FILE or OLD FILE message in the form

```
file name . . message
```

where the file name is the name of the file to be written; the message tells the user whether a file with the same name currently exists in his directory. For example, if the name of a file to be created is specified as PERSONNELSORTED in an IML command and a file with that name exists in the user's directory, the system prints

```
PERSONNELSORTED . . OLD FILE
```

and waits for the user to confirm or cancel the file name. The user confirms the file name by typing a carriage return. If the user confirms an OLD FILE message, the previous contents of the named file are destroyed. If the user wishes to change the name of a file to be created, he types an alt mode/escape; the system then prompts for another file name. In the example below, the user cancels the original file name with an alt mode/escape, enters another file name; and confirms the name with a carriage return after the NEW FILE message.

```
PERSONNELSORTED . . OLD FILE⊕
SORTED DATA TO: STDPERSONNEL↵
STDPERSONNEL . . NEW FILE↵
```

The user may call any IML program and specify the appropriate components in a single command. For example:

```
-SORT PERSONNELDATA:PDESC TO PERSONNELSORTED BY EMP.NO↵
```

A line feed may be used at any time to continue a command onto the next line. For example, the command

```
–SORT PERSONNELDATA:PDESC TO PERSONNELSORTED ↓
BY EMP.NO ↵
```

is identical to the previous command. Note that the line feed serves only to continue the command and does not function as a space. For this reason, the user must include a space before or after the line feed if a space is normally required.

The user may, however, type an incomplete command and let the system prompt for the missing components. For example:

```
-SORT↵
UNSORTED DATA FROM: PERSONNELDATA↵
DESCRIPTION FROM: PDESC↵
SORTED DATA TO: PERSONNELSORTED↵
PERSONNELSORTED.. NEW FILE↵
NAMES OF KEY FIELDS: EMP.NO↵
```

Similarly, the user may type part of the command, provided the components appear in the proper order. For example:

```
-SORT PERSONNELDATA:PDESC TO PERSONNELSORTED↵
PERSONNELSORTED.. OLD FILE↵        The user destroys the previous contents of the named file.
NAMES OF KEY FIELDS: EMP.NO↵
```

When the system prompts for the first missing component, the user may enter the remainder of the command. For example:

```
-SORT PERSONNELDATA:PDESC↵
SORTED DATA TO: PERSONNELSORTED BY EMP.NO↵
PERSONNELSORTED.. OLD FILE↵
```

When the system prompts for the output file, the user enters the file name and also the key field name on the same line.

The Information Management Library contains a built-in editing capability similar to EDITOR. Using standard line-editing characters, such as Control A, Control W, and Control Q, the user may edit a command as it is typed.[1] For example, the command

```
–SELECT FFA^C←ROM PERSONNELDATA:PDESC IF DPETW^C\DEPT=5↵
```

is accepted as:

```
SELECT FROM PERSONNELDATA:PDESC IF DEPT=5
```

In addition, when the user enters a command and receives an error message, he may type a Control Y to use the incorrect command as an image for reentering the corrected command. In the following example, the user types a complete command, not realizing that he omitted the

1 – See the *Tymshare EDITOR Reference Manual* for documentation of control characters and their functions.

H from the word WITH at the start of the command. When he receives the error message and prompt, he types Control Y, then Control Z followed by a T which copies up to and including the T, Control E to insert the missing H, and finally, Control D to copy the rest of the command.

```
–MERGE NEWPERSONNEL:PDESC WIT PERSONNELSORTED TO TOTAL BY EMP.NO↵
COMMAND STARTING WITH 'WIT' IS BAD.
FILE DESCRIPTION FROM: Y^c
Z^cTPDESC WITE^c<HD^c> PERSONNELSORTED TO TOTAL BY EMP.NO
TOTAL.. NEW FILE↵

OK.
```

Note that part of the command is accepted. The user may first type Control H to print the portion of the command available for editing. Then he types Control Y. The previous example is repeated using Control H first, then Control Y to edit the incorrect command.

```
–MERGE NEWPERSONNEL:PDESC WIT PERSONNELSORTED TO TOTAL BY EMP.NO↵
COMMAND STARTING WITH 'WIT' IS BAD.
FILE DESCRIPTION FROM: H^cPDESC WIT PERSONNELSORTED TO TOTAL BY EMP.NOY^c
Z^cTPDESC WITE^c<HD^c> PERSONNELSORTED TO TOTAL BY EMP.NO
TOTAL.. OLD FILE↵

OK.
```

The user may interrupt command operation at any time by typing two consecutive alt mode/escapes.[1] The system asks

```
***CONTINUE?
```

and the user must answer YES or NO, followed by a carriage return. If the answer is NO, the command is aborted, and control is returned to the EXECUTIVE.

The [VERSION] option is available to determine the version number of any IML program. The command form is simply:

```
–program name [VERSION]↵
```

For example, to determine the version number of the current REPORT program, the user types:

```
–REPORT [VERSION]↵
```

The HUSH option, which may be used with many IML commands, allows the user to suppress IML return messages. This feature is intended to be used primarily with command files. It is available for use with all commands except DEFINE, NOTE, and PERFORM.

The HUSH option must appear immediately after the command name. The user must also type the word HUSH enclosed in square brackets as illustrated in the example below.

```
–REPORT [HUSH] SPERSONNEL:PDESC TO T AS PER PRUL↵
```

If the user includes another option, as is possible for many commands, the HUSH option should appear first in the command statement. For example:

```
–SELECT [HUSH] [SINGLE] A:AD FROM D:DD TO C BY KEY↵
```

1 – The IML programs are designed to prevent interruption from spurious line noise. The present interrupt routine stops with the first alt mode/escape; then, if a second alt mode/escape is not received within four seconds, the program continues at the point of interruption.

The following are examples of IML messages which are suppressed when the HUSH option is used:

```
OK.
REPORT  FINISHED
4  RECORDS  SELECTED  FROM  4000
```

SECTION 2
BASIC CONCEPTS OF IML

This section explains the design of the Information Management Library: the information files with which it works, the various components of its commands, and the vast capabilities available in IML to handle information processing and retrieving requirements. New IML users should read this section carefully and in sequential order to achieve a basic understanding of the capabilities, terms, and construction of all IML commands and components. Users who are familiar with the Information Management Library may wish to refer directly to subsequent sections.

Many of the sample commands discussed in this section first appear in the sample application which follows.

SAMPLE APPLICATION

As an introduction to the basic concepts of IML, an example is shown below. The user need not concern himself at this point with complete comprehension of all the commands used; after reading the entire section, the meaning of the commands should be apparent.

The introductory problem concerns an accounts receivable application, which highlights the extraordinary speed, flexibility, and design of IML to maintain, access, and handle information stored on files on the TYMCOM-IX. This example shows the use of some fundamental IML commands to manage the raw data in a RETRIEVE data base named MASTER.[1]

```
-TYPE MASTER↵

ABC SUPPLIES      4031    250.92  720215       720210      150.66
ABC SUPPLIES      4170    375.92  720401            0      375.92
CARTER PRINTING   4018    899.55  720115       711208      678.45
DAILY DESIGN      3921    298.50  711230            0      298.50
FORREST PRINTING  4207     77.77  720415            0       77.77
KAY DEALERS       4251   1150.27  720430            0     1150.27
KUBLA PRINTS      3999    567.67  720101       720115      311.51
LETTEREX          4119    153.61  720401            0      153.61
ODIN WINERY       4075    565.00  720301            0      565.00
T T & G           3877   2008.23  711115       711209      271.30
Account          Invoice  Invoice amount  Due date  Last payment date  Balance
-
```

1 – IML can use RETRIEVE binary and symbolic data bases; in fact, IML automatically finds the corresponding RETRIEVE structure file when the user specifies a RETRIEVE data base in an IML command.

MASTER contains the information for all outstanding accounts in alphabetical order by account name. Dates are written in the form

YYMMDD

where the year, month, and day are represented by two-digit integers. For example, 720210 represents February 10, 1972.

The file MASTER'STR.E', created by RETRIEVE and read by IML commands, describes the information contained in each record.

```
-TYPE MASTER'STR.E'↵

C 16 ACCOUNT
I 5 INVOICE
N 10.2 INVAMT
I 8 DUEDATE
I 12 LASTPAYMENT
N 10.2 BALANCE

-
```

Once a week, the accounts receivable clerk adds the new invoice information to the data base, using the CREATE command.

```
-CREATE APPENDING TO MASTER FROM T↵
MASTER.. OLD FILE↵

OK.

DATA MUST BE ENTERED IN THE FOLLOWING ORDER:
ACCOUNT C 16
INVOICE N 5
INVAMT N 10 2
DUEDATE N 8
LASTPAYMENT N 12
BALANCE N 10 2
```

CREATE accepts the data with no format requirements and adds the new records to MASTER in the proper RETRIEVE format.

```
1: MOORE ASSOC,4300,68.10,720501,0,68.10↵
2: FRANK SHIPPING,4290,97.50,720501,0,97.50↵
3: MARSHALL CO,4297,412.12,720501,0,1600.9←←←←←←←412.12↵
4: AARON PRODS,4304,1600.93,720501,0,1600.93↵
5: ↵
4 RECORDS CREATED.

-
```

A Carriage Return typed immediately after the prompt terminates the data entry.

CREATE has a built-in editing facility similar to EDITOR.[1]

1 – See the *Tymshare EDITOR Reference Manual.*

Now the user wishes to re-sort the appended file to alphabetical order by account name.[1]

```
-SORT MASTER BY ACCOUNT↵

OK.

SORT FINISHED.

-
```

The file named MASTER is displayed below. A star (★) indicates the new records added with CREATE.

```
-TYPE MASTER↵

★AARON PRODS         4304   1600.93  720501          0   1600.93
 ABC SUPPLIES        4031    250.92  720215     720210    150.66
 ABC SUPPLIES        4170    375.92  720401          0    375.92
 CARTER PRINTING     4018    899.55  720115     711208    678.45
 DAILY DESIGN        3921    298.50  711230          0    298.50
 FORREST PRINTING    4207     77.77  720415          0     77.77
★FRANK SHIPPING      4290     97.50  720501          0     97.50
 KAY DEALERS         4251   1150.27  720430          0   1150.27
 KUBLA PRINTS        3999    567.67  720101     720115    311.51
 LETTEREX            4119    153.61  720401          0    153.61
★MARSHALL CO         4297    412.12  720501          0    412.12
★MOORE ASSOC         4300     68.10  720501          0     68.10
 ODIN WINERY         4075    565.00  720301          0    565.00
 T T & G             3877   2008.23  711115     711209    271.30

-
```

1 – The usual and suggested form of SORT creates a new output file, preserving the original data file. The example form maintains the correspondence of a RETRIEVE file name with the name of the structure file containing the data description.

Now the user wishes to examine the accounts in the data file which have due dates before February 1, 1972. The SELECT command handles this task:

```
-SELECT FROM MASTER IF DUEDATE<720201 TO T:ACCOUNT,DUEDATE,¬
BALANCE↵

OK.

CARTER PRINTING    720115    678.45
DAILY DESIGN       711230    298.50
KUBLA PRINTS       720101    311.51
T T & G            711115    271.30


4 RECORDS SELECTED FROM 14

-
```

IML prints the requested fields from each selected record.

Once each month, IML is used to update the master data file. The accounts receivable clerk creates a RETRIEVE file, TRANSACTIONS, containing information necessary to locate and update certain records in the master data file.

```
-TYPE TRANSACTIONS↵

ABC SUPPLIES       4031        720215       150.66
CARTER PRINTING    4018        720220       500.00
DAILY DESIGN       3921        720225       150.00
KAY DEALERS        4251        720217       550.00
KUBLA PRINTS       3999        720213       200.00
T T & G            3877        720228       271.30
Account name       Invoice     Payment date Amount paid
-
```

This file contains information on full and partial payments made during the month for any outstanding account.

The structure file, TRANSACTIONS'STR.E', describes the information in TRANSACTIONS.

```
-TYPE TRANSACTIONS'STR.E'↵

C 16 ACCOUNT
I 5 INVOICE
I 12 PAYTDATE
N 10.2 AMTPAID

-
```

Using the information in TRANSACTIONS, the user performs two updating tasks on the data file MASTER: replacing the most recent date of payment and updating the outstanding balance.[1]

1 – Both tasks can be performed easily with a simple UPDATE procedure. For purposes of the example, however, REPLACE performs the first task, and UPDATE performs the second task.

The user types the REPLACE command to replace the date of last payment. For each record in the TRANSACTIONS file, REPLACE locates MASTER records with the same account name. The IF clause then checks for identical invoice numbers for the given account. Note that :A following a field name indicates an activity record field, and :D following a field name indicates a data record field. MASTER contains the data records, and TRANSACTIONS contains the activity records.

```
-REPLACE MASTER WITH TRANSACTIONS TO CURRENT:LASTPAYMENT WITH ¬
PAYTDATE IF INVOICE:A=INVOICE:D BY ACCOUNT↵
CURRENT.. NEW FILE↵
                                IML writes a new file, CURRENT, which contains the revised
OK.                             MASTER file records. The original file is unchanged.


6 RECORDS REPLACED OF 14

-TYPE CURRENT↵                  This file has the same format as the MASTER file. Note that
                                the last payment date is replaced in the relevant records.
AARON PRODS        4304  1600.93  720501            0  1600.93
ABC SUPPLIES       4031   250.92  720215      720215*   150.66
ABC SUPPLIES       4170   375.92  720401            0   375.92
CARTER PRINTING    4018   899.55  720115      720220*   678.45
DAILY DESIGN       3921   298.50  711230      720225*   298.50
FORREST PRINTING   4207    77.77  720415            0    77.77
FRANK SHIPPING     4290    97.50  720501            0    97.50
KAY DEALERS        4251  1150.27  720430      720217*  1150.27
KUBLA PRINTS       3999   567.67  720101      720213*   311.51
LETTEREX           4119   153.61  720401            0   153.61
MARSHALL CO        4297   412.12  720501            0   412.12
MOORE ASSOC        4300    68.10  720501            0    68.10
ODIN WINERY        4075   565.00  720301            0   565.00
T T & G            3877  2008.23  711115      720228*   271.30

-
```

Next, the user employs the UPDATE command to revise the outstanding balance information in certain records of the CURRENT data file, using the records in the TRANSACTIONS activity file. Note that the data file CURRENT consists of records written in the same RETRIEVE format as MASTER.

```
-UPDATE CURRENT:MASTER'STR.E' WITH TRANSACTIONS TO REVISED ¬
IF INVOICE:A=INVOICE:D BY ACCOUNT AS PER RULES1↵
: LIST↵              The rules file is listed for purposes of the example.
10 BALANCE-AMTPAID:A TO BALANCE
: RUN↵               The user executes the command.
REVISED.. NEW FILE↵

OK.


6 RECORDS UPDATED OF 14

-
```

UPDATE substitutes the new BALANCE information in the appropriate data file records as per the RULES1 instructions shown on the preceding page.

REVISED contains all the information from the file CURRENT with the outstanding balance revisions.

```
-TYPE REVISED↵

AARON PRODS        4304   1600.93  720501          0   1600.93
ABC SUPPLIES       4031    250.92  720215     720215       .00★
ABC SUPPLIES       4170    375.92  720401          0    375.92
CARTER PRINTING    4018    899.55  720115     720220    178.45★
DAILY DESIGN       3921    298.50  711230     720225    148.50★
FORREST PRINTING   4207     77.77  720415          0     77.77
FRANK SHIPPING     4290     97.50  720501          0     97.50
KAY DEALERS        4251   1150.27  720430     720217    600.27★
KUBLA PRINTS       3999    567.67  720101     720213    111.51★
LETTEREX           4119    153.61  720401          0    153.61
MARSHALL CO        4297    412.12  720501          0    412.12
MOORE ASSOC        4300     68.10  720501          0     68.10
ODIN WINERY        4075    565.00  720301          0    565.00
T T & G            3877   2008.23  711115     720228       .00★

-
```

Now the user deletes the accounts which are paid in full, that is, records whose BALANCE is zero.

```
-PURGE FROM REVISED:MASTER'STR.E' IF BALANCE=0 TO MASTER↵
MASTER.. OLD FILE↵
OK.

2 RECORDS PURGED FROM 14

-
```

The user writes the updated information over the previous contents of the file MASTER.

MASTER contains the up-to-date accounts receivable data. To continue his weekly and monthly information management and retrieval, the user repeats the same IML procedures, using the new MASTER file.

Finally, the user generates a report from the MASTER file, using the REPORT command.

```
-REPORT MASTER TO MREPORT AS PER RPTRULES↵
: LIST↵
10  INITIAL
20      SKIP 6
30      PRINT 23B, "ACCOUNTS RECEIVABLE SUMMARY",CR,29B,
           "AS OF ", @DATE, CR
35      SKIP 3
```

The report rules are listed for purposes of the example.

```
40      PRINT 22B, "ACCOUNT",11B, "BALANCE DUE",CR,22B,"-------",
           11B,"-----------",CR,CR
50   DETAILS
60      PRINT 22B, ACCOUNT
70      TAB 42
80      PRINT BALANCE,CR
90   FINAL
100     TAB 41
110     PRINT "-----------",CR
115     PRINT 22B,"TOTAL"
120     TAB 42
130     PRINT SUM BALANCE($$$$$$$.DD),CR
: RUN
MREPORT.. NEW FILE

OK.



REPORT FINISHED.


-
```

The user wants to print the account name and balance for each record in the data file.

The user executes the command.

The report produced is shown below.

```
-TYPE MREPORT

                              ACCOUNTS RECEIVABLE SUMMARY
                                    AS OF 03/01/72


                             ACCOUNT            BALANCE DUE
                             -------            -----------

                             AARON PRODS            1600.93
                             ABC SUPPLIES            375.92
                             CARTER PRINTING         178.45
                             DAILY DESIGN            148.50
                             FORREST PRINTING         77.77
                             FRANK SHIPPING           97.50
                             KAY DEALERS             600.27
                             KUBLA PRINTS            111.51
                             LETTEREX                153.61
                             MARSHALL CO             412.12
                             MOORE ASSOC              68.10
                             ODIN WINERY             565.00
                                                -----------
                             TOTAL                 $4389.68

-
```

IML COMMANDS

In addition to CREATE, DEFINE, and VERIFY for data creation procedures, the Information Management Library contains commands which encompass all aspects of information management to satisfy the rigorous and growing demands of data processing.

IML includes two fast sorting programs: SORT and MERGE. SORT sorts the contents of a file on one or more key fields. It is used in the introductory example on page 9 to arrange the records in a file in alphabetical order by account name. MERGE merges two sorted files into a single file in sorted order.

SELECT is a fast, versatile information retrieval program for both elaborate conditional information retrieval and simple, straightforward applications. In the sample problem, the SELECT command prints the requested information about overdue accounts with one IML command:

```
–SELECT FROM MASTER IF DUEDATE<720201 TO T:ACCOUNT,DUEDATE,BALANCE↵
```

IML offers three commands to handle the updating needs of an information management system: REPLACE, UPDATE, and PURGE. The REPLACE command allows replacement of records or specific fields in a record. On page 11, the user types a REPLACE command to change the payment date information in the records. The UPDATE command permits arithmetic operations in changing the data within a record. For example, UPDATE is used on page 11 to determine the new balance and replace the old balance with it. The PURGE command deletes entire records from a file. It is used on page 12 to delete accounts paid in full.

Finally, IML offers a comprehensive reporting facility to prepare reports of any description, including those on preprinted forms, based on the information in a data file. REPORT provides the ability to control paging and spacing, print titles and page headings, perform calculations, and accumulate totals, subtotals, and averages.

DATA FILES

The Information Management Library allows the user to store a large volume of related data, then access and update this information as required. The information accessed and stored by IML is written on a file.

All data files consist of records. A *record* contains one or more related pieces of information forming a unit. Each piece of information in a record is called a *field.* In the preceding example, each record in the file consists of six fields specifying the account name, invoice number, total bill, due date, date of most recent payment, and current balance due for a given invoice. Initially, the user names each field and specifies its location within the record, its length, and the type of data it contains: numeric or character.

IML is designed to handle information stored in specific locations in each record. For example, in MASTER, each record contains the invoice number in a specified location. IML requires, however, that only those fields which are used in the command operation appear in the same location in each record.

When all records in a file contain the same number of characters and have a constant length for the life of the file, they are called *fixed length records.* When the length of all records changes during the life of the file or when the records in the file are of different length, they are called *variable length records.*

The term *data file* refers to the information file containing the records. It may be a master data file, an activity data file, or a data file maintained alone. A *description file* contains no data but specifies the parameters of a data file, providing the following information: type of file, binary or symbolic; type of records, fixed or variable length; and record description.

Entering the Data

The user may create or append to RETRIEVE files as well as IML data files by entering the records at the terminal or from a file. The CREATE command offers convenient options, including prompts, error diagnostics, and complete editing facilities for data entry. The user may enter the data without regard to format or position in the record. This free-format capability permits the user to separate the data with commas or leave certain fields blank. CREATE then produces an IML or RETRIEVE file in the appropriate format.

The description of the data file is crucial for maximizing IML's extraordinary speed and economy. CREATE takes its instructions for storing the information from the description file whose name appears in the command.[1] For example,

```
-CREATE INVOICES:INVDES FROM T
```

creates a file named INVOICES, storing in fixed format the data entered in free format at the terminal. The file INVDES contains the description which CREATE uses to write INVOICES.

Describing a Data File

Before executing commands, IML requires certain descriptive information about a data file and its records. The user may describe his data file with a description file created previously by a DEFINE command, use a RETRIEVE structure file, or specify T as the description file and enter the description at the terminal immediately before command execution.

The DEFINE command assists the user in designing the initial description, and stores the format on an IML description file. To take greatest advantage of IML's economical information storage and handling, the user may design his own data file, then enter the data easily with CREATE. To access and update the data file with any subsequent IML command, the user specifies its description simply by entering the data file name followed by a colon (:) and the description file name. For example,

```
-SELECT FROM INVOICES:INVDES IF QUANTITY>100 TO T
```

instructs IML to read the information on INVOICES according to the description on INVDES.

When working with a RETRIEVE data base, the user may omit the description file name if an associated RETRIEVE structure file exists in the user's directory. For example, in the introductory example on page 10,

```
-SELECT FROM MASTER
```

instructs IML to seek the associated RETRIEVE structure file, MASTER'STR.E', to describe the information stored in MASTER.

The user may specify a RETRIEVE structure file name to describe a data file that is in RETRIEVE format but does not have a corresponding structure file. For example,

1 – As shown on page 8, CREATE can also work with RETRIEVE structure files.

```
-UPDATE CURRENT:MASTER'STR.E'↩
```

specifies that the file named CURRENT contains information in the format stated on the RETRIEVE structure file MASTER'STR.E'.

ACTIVITY FILES

Many IML commands allow the user to work with two data files. The primary data file always contains the information to be managed; the secondary data file may contain records to locate, update, replace, or, in general, facilitate the handling of the information in the master data file. The secondary data file is called an *activity file.* The introductory example shows the use of an activity file, TRANSACTIONS, to specify changes to certain records in the main data file.

Creating the File and Its Description

The user creates and describes an activity file as he would any data file in IML, using CREATE and DEFINE. The description file name is required in an IML command unless the activity file is a RETRIEVE data base described by a corresponding structure file.

Key Fields

An activity file instructs IML to perform commands only on the records in the master data file which match one or more records in the activity file.[1] The match is determined by key fields which the user specifies in the BY clause. For example, assume the files MASTER and TRANSACTIONS are used in an IML command. The BY ACCOUNT clause in the command instructs IML to read the ACCOUNT field in a record of TRANSACTIONS and locate the first record in MASTER which matches exactly in the ACCOUNT field. Then and only then does IML perform the command on the MASTER file record. In this way, IML is able to perform a massive amount of updating or retrieving with one IML command. The use of an activity file enhances the efficiency and power of the Information Management Library.

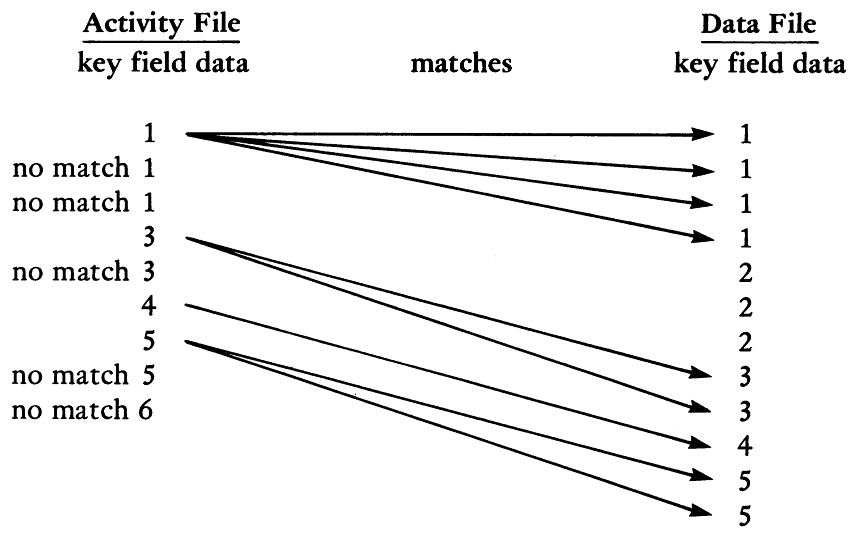
NOTE: The records in both files (a data file and a specified activity file) may be of differing types; that is, one may be binary while the other is symbolic. The records must be sorted on the key field or fields used to determine a match between an activity record and a data record.[2] *The key fields must have the same data type and length in the data and activity description files.*

Matching Data and Activity Records In REPLACE, SELECT, and PURGE

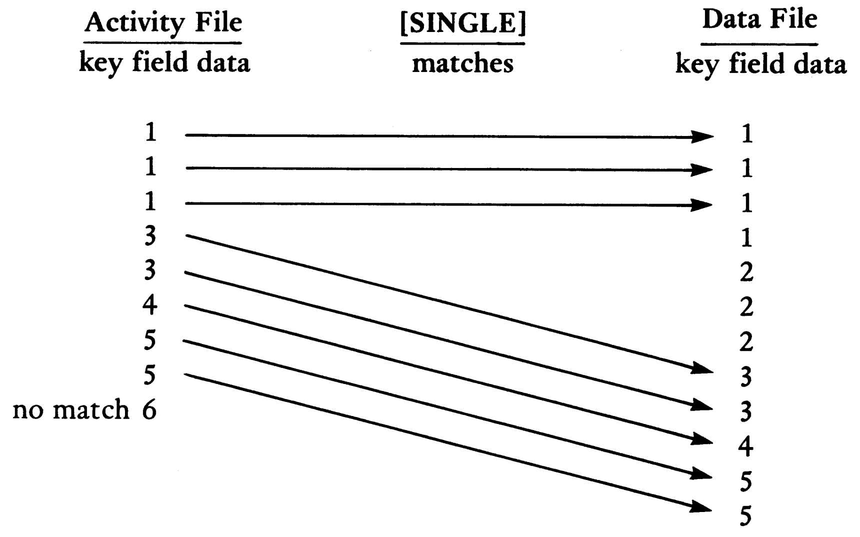
In REPLACE, SELECT, and PURGE, IML matches an activity record with all data records which specify the same key field data. The following diagram illustrates the normal matching procedure in REPLACE, SELECT, and PURGE.

1 – In UPDATE, the user may specify some instructions for all data records, and other instructions for only the records which are located by an activity record matching on the key fields.

2 – The SORT command may be used to order records based on a key field list. The SORT command is presented on page 83.



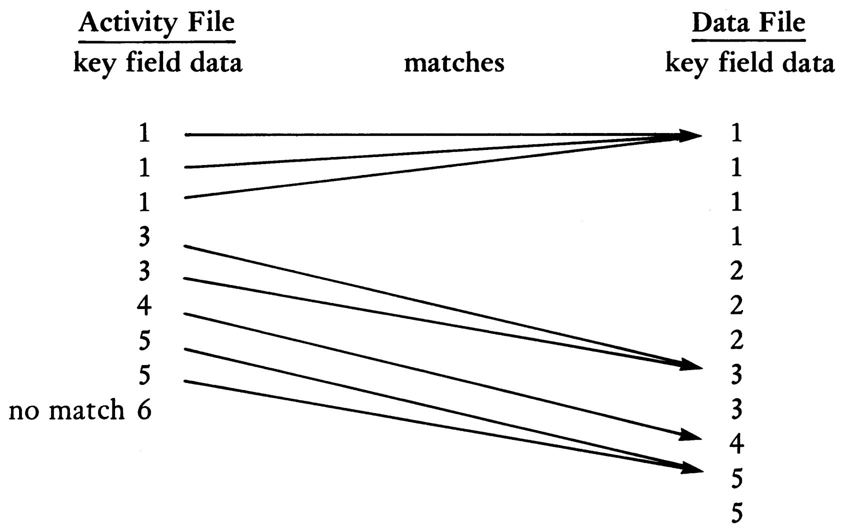
The SINGLE option in REPLACE, SELECT, and PURGE instructs IML to match an activity record with only the first unused data record which specifies the same key field data. The diagram below illustrates the matching procedure in REPLACE, SELECT, or PURGE when the SINGLE option is requested.[1]



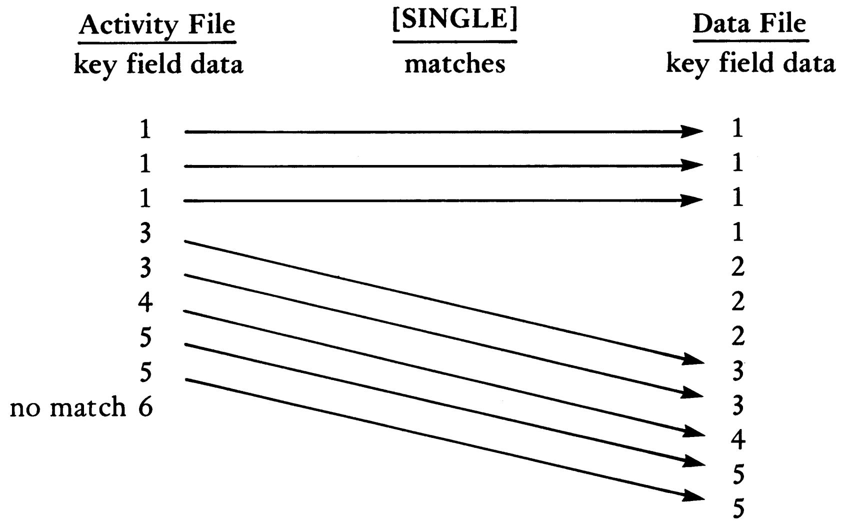
Matching Data and Activity Records in UPDATE

With the UPDATE command, the normal procedure for matching data and activity records on key fields is different from other IML commands. In UPDATE, all activity records which specify the same key field data are used to update the first matching data record. The diagram following illustrates the matching procedure in UPDATE.

1 – The REPLACE command form is presented on page 97; the SELECT and PURGE command forms are presented on page 90.

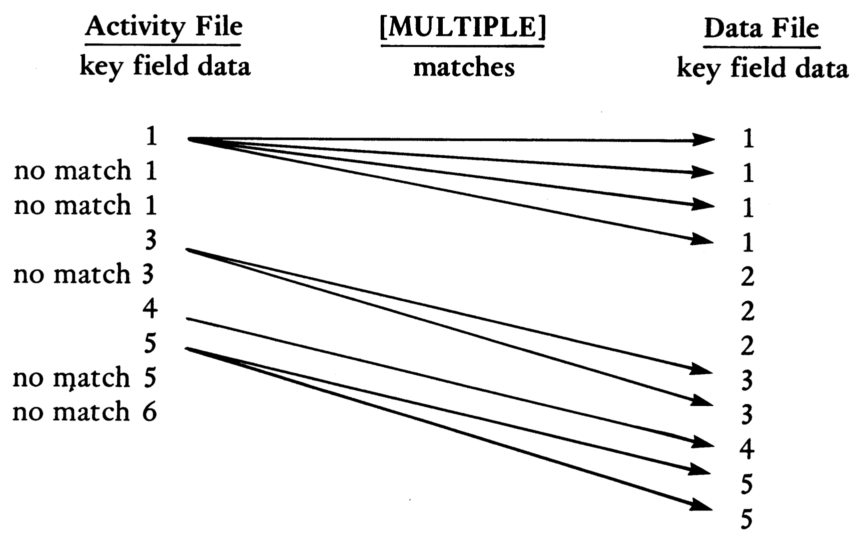


The SINGLE option instructs UPDATE to use only the first of several matching activity records to update a data record. If other data and activity records contain the same key field data, the process is repeated. The diagram below illustrates the matching procedure in UPDATE when the SINGLE option is requested.[1]



The MULTIPLE option provides an alternative method for matching data and activity records in the UPDATE procedure. Normally all activity records which specify the same key field data are used to update the first matching data record. When the MULTIPLE option is specified, a single activity record is used to update all data records that specify the same key field data. The following diagram illustrates the matching procedure in UPDATE when the MULTIPLE option is specified.

1 – The UPDATE command form is presented on page 104.



The UPDATE [MULTIPLE] matching procedure is the same as the normal matching procedure for SELECT, PURGE, and REPLACE.

THE OUTPUT FILE

The user need not concern himself with the possibility of inadvertently destroying his data files with any IML command. All commands request the user to specify the name of a file on which to write the results of the information management procedure. The original files remain unchanged, allowing the user to perform more than a single IML command with the original files and protecting the user from an erroneous command.

The user specifies a name for the output file in the IML command. After typing the specifications of the original files, the user types TO followed by an output file name. For example:

```
-REPLACE MASTER WITH TRANSACTIONS TO CURRENT↵
```

After performing the REPLACE command, IML writes all the data records on a file named CURRENT. The user may specify T as the output file to print all records directly at the terminal without creating a permanent file. The format of the output file is automatically the same as the master data file. In the previous command, the records in CURRENT contain the same fields as the records in MASTER; therefore, the user may type

```
-UPDATE CURRENT:MASTER'STR.E'↵
```

to specify CURRENT and its description in a subsequent IML command.

The user may reformat the output records with any SELECT or PURGE command by specifying, after the output file name, a list of desired fields from the master data file. For example,

```
-SELECT FROM MASTER TO T:ACCOUNT,DUEDATE,BALANCE↵
```

prints at the terminal only the specified fields from selected records. See page 10 of the introductory example for an illustration of this feature. The reformat capabilities are available to perform extensive restructuring of any information file. For complete instructions, see the discussion on page 87.

IML also makes it possible to create an output file containing records which are a combination of matched data and activity records. For example:

```
–REPLACE MASTER WITH TRANSACTIONS TO CURRENT:LASTPAYMENT ↴
WITH PAYTDATE↩
```

The output file, CURRENT, contains records consisting of all fields from the data record except LASTPAYMENT, which IML replaces with PAYTDATE from the matching activity record. See page 11 of the introductory example for an illustration of this feature; complete instructions are presented with the REPLACE command on page 97.

The IML reformatting features are an important facility for handling information files of different descriptions. This capability is particularly useful in RETRIEVE management procedures, as illustrated on page 151.

IML CONDITIONAL EXPRESSIONS

IML incorporates a total capability for qualifying data file records for command procedures. Similarly, the user may qualify activity records for the command procedure or qualify output records based on a conditional expression involving fields in the data and activity records. For example, the user wants to execute a SELECT command only for certain data records with BALANCE greater than 500:

```
–SELECT FROM MASTER IF BALANCE>500 TO T↩
```

The following command uses three different conditional expressions (IF or FOR clauses) to qualify data, activity, and output records, respectively, for the UPDATE procedure:

```
–UPDATE MASTER FOR DUEDATE>720301 WITH TRANSACTIONS ↴
IF PAYTDATE>720301 TO CURRENT IF INVOICE:A=INVOICE:D BY ACCOUNT ↴
AS PER UPDRULES↩
```

Only MASTER data records containing due dates after March 1, 1972, and TRANSACTIONS activity records with payment dates after March 1, 1972, are to be matched to produce CURRENT, the output file. Finally, IML writes an output record for the current data record and matching activity record only if the invoice number in the activity record is the same as the invoice number in the data record.

The third IF clause, that INVOICE:A equals INVOICE:D, is an interfile conditional expression. The interfile IF or FOR clause follows the output file specification. A data record field is indicated by :D following the field name; an activity record field is indicated by :A following the field name.

In most IML commands, the user may specify as many as three IF or FOR clauses containing conditional expressions with arithmetic, relational, and logical operators, if desired. The words IF and FOR may be used interchangeably to begin a conditional expression. *NOTE: In all cases in this manual, the phrase* IF clause *means an IF clause or a FOR clause.* Similar conditional and arithmetic expressions may appear in an UPDATE, VERIFY, or REPORT rules file.[1]

Command forms are presented with the relevant IML commands and rules statements throughout the manual. See Appendix A for a summary of forms showing the location and number of IF clauses in each IML command.

1 – See page 27 for the complete discussion of rules files. Identical conditional expressions may appear in IF, ORIF, and FOR rules; identical arithmetic expressions may be used with replacement, TYPE, ELSE, and DO rules.

The following hierarchy of operations is used in evaluating a conditional expression:

1. Arithmetic operators
2. Relational operators
3. Logical operators

Arithmetic Operators

Arithmetic expressions may appear in any conditional expression. An arithmetic expression consists of field names and/or numbers separated by arithmetic operators. The four arithmetic operators available in IML are shown below.

Arithmetic Operator	Function
+	Addition
−	Subtraction or unary minus
*	Multiplication
/	Division

The following rules apply to the construction and evaluation of arithmetic expressions.

When parentheses are not used, an arithmetic expression is evaluated as follows:

Hierarchy of Operations	
Hierarchy Level	**Operation**
1	Unary minus
2	Multiplication and division
3	Addition and subtraction

Parentheses may be used for readability or to override the hierarchy of evaluation stated above. Expressions within parentheses are evaluated first; within a nest of parentheses, the evaluation begins at the innermost set of parentheses and proceeds to the outermost set of parentheses. For example, the expression

```
(A+1)/((D+1)*100)
```

is evaluated as follows: First, D is added to 1; then the result is multiplied by 100 to determine the denominator; then A is added to 1 to determine the numerator; and finally, the numerator is divided by the denominator. Without parentheses, the expression

```
A+1/D+1*100
```

is evaluated using the implied hierarchy shown above: First, 1 is divided by D; then 1 is multiplied by 100; then A is added to the first expression; and finally, the last expression is added.

When the order of operations on the same hierarchal level is not completely specified, such as addition and subtraction, the order of operations is from left to right.

Relational Operators

A relational operator specifies a comparison between the values on either side of the relational operator. For example, the relational operator > in the conditional expression

IF A + B > C

specifies that A + B must be greater than C.

The table below presents the available relational operators for constructing IML conditional expressions. Note that the relational operators in the second part of the table are designed specifically for character data.

Relational Operator	Meaning	Example
	For Numeric or Character Fields	
<	Less then	IF ONHAND<50
>	Greater than	FOR HR.RATE>0
=	Equal to	IF CODE:A=1
#	Not equal to	FOR DIV # 0
<=	Less than or equal to	IF DUEDATE<=720525
>=	Greater than or equal to	FOR RECNO:A>=100
NUMERIC	Is a legal number[1]	IF NBALANCE NUMERIC
	For Character Fields Only	
HAS	Contains	IF DESC HAS '1X1/4 INCH'
IN	Contained in	FOR '1X1/4 INCH' IN DESC
STARTS WITH	Begins with	IF NAME STARTS WITH 'ROC'
STARTS	Begins	IF 'ROC' STARTS NAME
ENDS WITH	Ends with	IF S.S.NO ENDS WITH '–4065'
ENDS	Ends	IF '–4065' ENDS S.S.NO
NUMERIC	Contains only digits or blanks[1]	FOR PARTNO NUMERIC

NOTE: Character data within an expression must be enclosed in single or double quote marks.

1 – The NUMERIC operator for a numeric field allows integer, fixed point, and floating point scientific notation. For example, 12, –43.85, 1.7E+2, and 6.3D–2 are all legal numbers. The NUMERIC operator for a character field allows the digits 0 through 9 and blanks. For example, character data such as 123, 4 5, 179 123, and 42 are considered numeric.

Logical Operators

The logical operators NOT, AND, and OR may be used to reverse or combine simple conditional expressions. The user may precede any of the relational operators mentioned above with the word NOT to reverse the interpretation of the relational operator. For example,

```
IF ONHAND NOT <50
```

is the reverse of:

```
IF ONHAND <50
```

Simple conditional expressions may be separated by AND or OR to construct a complex conditional expression. For example:

```
IF ONHAND<50 AND "DISCONTINUED" NOT IN REMARKS OR ONORDER=0
```

The AND operator specifies that the simple expressions combined by AND must all be true for the combined AND expression to be true, whereas the OR operator specifies that any one or all of the expressions separated by OR must be true for the combined OR expression to be true. For example, the conditional expression

```
IF QUANTITY<70 AND CODE=4
```

specifies two conditions which must be true, whereas

```
IF QUANTITY<70 OR CODE=4
```

specifies that at least one of the two conditions must be true.

The user may construct conditional expressions using NOT, AND, and OR as often as required. The following hierarchy applies in evaluating conditional expressions with arithmetic, relational, and logical operators:

1. All arithmetic and relational operators
2. NOT
3. AND
4. OR

Note the effect of the AND operator taking precedence over the OR operator. The conditional expression

```
IF A=0 AND B=3 OR C=7
```

is evaluated as:

```
IF (A=0 AND B=3) OR (C=7)
```

Similarly, the expression

```
IF S=1 OR T>5 AND E=0 OR V=1 AND E<50
```

is evaluated as:

```
IF (S=1) OR (T>5 AND E=0) OR (V=1 AND E<50)
```

Parentheses may be used either to improve readability or to override the normal order of evaluation stated above. For example, the user includes parentheses to specify the condition:

```
IF (CODE=1 OR CODE=4) AND QUANTITY<70
```

Without parentheses, the normal evaluation is:

```
IF CODE=1 OR (CODE=4 AND QUANTITY<70)
```

Abbreviating Conditional Expressions

When the subjects are identical in a series of relational expressions, it is permissible to omit the subject from all but the first relational expression. For example, the IF clause

```
IF A=B OR A=C OR A>S OR A>T
```

may be shortened to:

```
IF A=B OR =C OR >S OR >T
```

When subjects and relational operators are identical in a series of relational expressions, both the subject and the relational operator may be omitted from all but the first relational expression. The preceding example may be shortened further to:

```
IF A=B OR C OR >S OR T
```

When NOT precedes a relational operator, both are assumed if a relational operator is not specified in subsequent expressions. For example,

```
IF CODE NOT=1 AND 2 AND 3
```

is equivalent to:

```
IF CODE NOT=1 AND CODE NOT=2 AND CODE NOT=3
```

The relational operators for character fields are paired so that the user can always state the condition desired without repeating the subject and the relational operator. For example,

```
IF "T" STARTS FIRSTNAME OR LASTNAME
```

could not be written using STARTS WITH without repeating the subject and the operator (IF FIRSTNAME STARTS WITH "T" OR LASTNAME STARTS WITH "T"). On the other hand,

```
IF NAME STARTS WITH 'R' OR 'T' OR 'P'
```

could not be written using STARTS without repeating NAME and the operator for each condition.

UTILITY FIELDS

IML provides several built-in utility fields which the user may specify in IML commands and rules files, as appropriate. The table following summarizes the available utility fields, specifying the data stored in each field and which IML programs provide that utility field.

Utility Field Name	Contains	Type and Length of Data	Available in IML
RECNO	Sequence number of record being processed	Numeric, 5	VERIFY SORT MERGE PURGE REPLACE UPDATE SELECT REPORT
CR	Carriage Return	Character, 1	VERIFY PURGE UPDATE SELECT REPORT
LF	Line Feed	Character, 1	Same as above
LENGTH	Number of characters in record being processed	Numeric, 5	VERIFY MERGE PURGE REPLACE UPDATE SELECT REPORT
@CALMONTH	Name of current month as MMM	Character, 3	Same as above
@CDATE	Current date as MMM.DD,YYYY	Character, 11	Same as above
@CTIME	Current time as HH:MM AM or PM	Character, 8	Same as above
@DATE	Current date as MM/DD/YY	Character, 8	Same as above
@DAY	Current day as DD	Numeric, 2	Same as above
@MONTH	Current month as MM	Numeric, 2	Same as above
@NDATE	Current date as YYMMDD	Numeric, 6	Same as above
@TIME	Current time as HH:MM	Character, 5	Same as above
@WEEKDAY	Name of current day of week	Character, 9	Same as above
@YEAR	Current year as YYYY	Numeric, 4	Same as above

SECTION 3
BASIC CONCEPTS OF RULES FILES

Three IML commands, VERIFY, UPDATE, and REPORT, perform their functions based on a user-created rules file. When entering one of these commands, the user includes an AS PER clause, which specifies the appropriate set of rules to be used in the command operation. For example, the command

```
-VERIFY ORDERS:ODESC AS PER ORDRULES↵
```

specifies a rules file named ORDRULES to be used in the VERIFY procedure. ORDRULES contains statements which determine whether a given field contains incorrect information, print error messages at the terminal, and accept new values for incorrect field information.

Although VERIFY, UPDATE, and REPORT have different functions, the creation and execution of their rules files are identical. The basic concepts of rules files are discussed within this section in several parts. First, rules file creation is presented, together with an explanation of the procedures and guidelines. The next discussion presents the simple command-level instructions for creating, listing, changing, deleting, and executing rules file statements. Next, the division of a rules file into several sections is described. Finally, the statements which constitute a rules file are detailed. The discussion presents instruction statements and control statements separately, outlining the various types of statements within each group and their functions.[1] The method of nesting statements is introduced at the end of the discussion of statements.

CREATING A RULES FILE

The user creates a rules file most easily by specifying the rules file name as T (for terminal) in a VERIFY, UPDATE, or REPORT command, allowing the system to prompt for and check each line of the rules file as it is entered. In this situation, the system requests the name of a file on which to save the rules. For example:

```
-REPORT ORDERS:ODESC TO ORDREPORT AS PER T↵
SAVE REPORT RULES ON: RPTORD↵
```

When the user finishes entering the rules, the system automatically writes the entered rules on the specified file.

NOTE: If the user does not want to save the rules but merely wishes to use them for the current command, he may type the word NOTHING, or any left subset of NOTHING, when the system prompts for the name of a file on which to save the rules.

1 – The ON and PRINT statements, which apply only to a REPORT rules file, are discussed on pages 138 and 147. Additional statements for use with command files are presented on page 172.

After the user enters a complete command, including the name of a file on which to save the rules, the system prints a colon (:), prompting the user to enter his rules. For example:

```
-VERIFY ORDERS:ODESC AS PER T↵
SAVE VERIFY RULES ON: ORDRULES↵
:
```

VERIFY indicates its readiness to accept a line of rules.

Each line of rules must begin with an integer from 1 to 10000, followed by at least one blank. For example:

```
10 TYPE "PLEASE REENTER HOURLY RATE FOR ",EMP.NO
20 INPUT RATE
```

Aside from requiring at least one blank after a line number, there are no formatting requirements beyond the specified statement forms. The user may type one or several blanks between words, as desired. Many examples in this manual include additional blanks to improve readability.

The system executes the rules in sequence according to the line numbers. The user need not, however, enter the lines in sequence; the lines are ordered automatically as they are entered. For example:

```
: 20 INPUT RATE↵
: 10 TYPE "PLEASE REENTER HOURLY RATE FOR ",EMP.NO↵
: LIST↵
10 TYPE "PLEASE REENTER HOURLY RATE FOR ",EMP.NO
20 INPUT RATE
:
```

It is suggested that the user assign line numbers which permit the insertion of additional lines at a later time, for example, 10, 20, 30, and so on.

The user may create a rules file with as many as 100 lines. A line may contain one or more statements.[1] A carriage return terminates a single statement on a line, whereas a semicolon (;) terminates a statement which is followed by another statement on the same logical line. For example,

```
10 IF RECNO=100
11 TYPE "AT RECORD 100"
```

is equivalent to:

```
10 IF RECNO=100; TYPE "AT RECORD 100"
```

A logical line may contain as many as 256 characters; the user continues a logical line onto another physical line with a line feed. For example:

```
: 10 IF ONHAND-QTY:A<50; TYPE "PART ",PARTNO,2B,DESCRIP↴
TION, "AT REORDER POINT"↵
```

NOTE: A line feed does not function as a blank. The user, therefore, must include a blank before or after a line feed when a blank is required.

All the line editing capabilities of Tymshare's EDITOR, such as Control A, Control Q, and Control W, are available to the user as he enters each line of rules.[2] For example, the line

1 – Statements are discussed on page 32. A rules line may contain as many as 256 characters.
2 – See the *Tymshare EDITOR Reference Manual* for an explanation of the numerous control characters and their functions.

: 10 IRFA^c←A^c←F RECNO=100; TIPEW^c\TYPE “AT RECORRA^c←D 100”↵

is accepted as:

10 IF RECNO=100; TYPE “AT RECORD 100”

In addition, the previous line, whether correct or in error, may be used as an image for the line being typed. Any of the EDITOR control characters, such as Control D, Control E, Control O, and Control Z, may be typed to copy and edit characters from the previous line to the line being typed. For example, the user enters line 10 and then enters line 15 using the image of line 10:

: 10 IF RECNO=100; TYPE “AT RECORD 100”↵
: 15 E^c<ORO^{c}1>IF RECNO=2Z^{c}D00; TYPE “AT RECORD 2D^{c}00”

Line 15 is accepted as:

15 ORIF RECNO=200; TYPE “AT RECORD 200”

COMMAND-LEVEL INSTRUCTIONS

When the colon prompt (:) appears, the user may add, delete, or change rules or enter a command to edit, modify, list, write, or execute the current rules. The following table summarizes the commands which the user may enter after the colon prompt.

Command	Meaning
: LIST $\left\{\begin{matrix}n\\ n1{:}n2\end{matrix}\right\}$↵ *or* : LIST↵	LIST alone prints all the current rules. LIST n, where n is a line number, prints the specified line. LIST n1:n2 prints a range of lines beginning with n1, and ending with n2.
: EDIT $\left\{\begin{matrix}n\\ n1{:}n2\end{matrix}\right\}$↵	Prints the first line to be edited (n or n1), returns the carriage, and waits for the user to reenter the line. This process is repeated until all lines specified, n1 through n2, are edited.
: MODIFY $\left\{\begin{matrix}n\\ n1{:}n2\end{matrix}\right\}$↵ *or* : MOD $\left\{\begin{matrix}n\\ n1{:}n2\end{matrix}\right\}$↵	Operates similarly to the EDIT command except that EDIT prints the line which is to be edited, whereas the MODIFY command does not.
: n statement(s)↵	Adds statement(s) as line n in the current rules.
: n↵	Deletes line n from the current rules.
: QUIT↵ *or* : Q↵	Writes the current rules on the specified file and returns control to the EXECUTIVE.

(Table Continues)

Command	Meaning
: RUN ↵	Writes the current rules on the specified file and executes the IML command using the current rules.
: SAVE {PRODUCTION / PRO} ON production file name ↵	Compiles and saves a copy of the entire command procedure. This command writes a copy of the procedure but does not execute it; control is returned to the EXECUTIVE where the PERFORM command is given to initiate execution of the procedure.

When new lines are added to an existing rules file, that is, when the user specifies an actual rules file name rather than T in the IML command, the RUN or QUIT command causes the system to prompt for another rules file name. This allows the user to preserve his original rules file, if desired. For example:

```
-VERIFY PAYROLL:DESC AS PER VRULES3↵
: 610 IF CODE="T"; DONE↵
: RUN↵
SAVE VERIFY RULES ON:
```

The user may type a different file name, preserving the original rules file; he may type the same file name as specified at the start of the command, saving only the current rules; or he may type the word NOTHING if he does not want to save the current rules on a file.

RULES FILE ORGANIZATION

A rules file may be separated into several sections of rules, providing the user with an efficient organization in which to detail his particular tasks. Although some rules sections are different in VERIFY, UPDATE, or REPORT rules file, the concept and procedure are the same.

Each section of rules has a special function and is executed when appropriate. For example, the HEADINGS section of a REPORT rules file specifies page headings and is executed every time the report skips to a new page. The FINAL section of all rules files, on the other hand, specifies final calculations and comments, and is executed only once after all records in the data file are processed.

The subsections which follow introduce the VERIFY, UPDATE, and REPORT rules files, presenting their general purpose and outlining their rules sections and corresponding functions. A rules file may contain all or some of the sections; however, they must appear in the order shown. When the user wants to create a particular rules file, he should refer to the appropriate IML command discussion of rules files where all sections are detailed.

NOTE: When no section name appears at the start of a rules file, IML assumes DETAILS or MATCHED, as appropriate.

The VERIFY Rules File

The VERIFY rules file directs the verification process. It specifies conditions to check in the records of the data file and allows the user to print error messages, correct field information directly, perform calculations, and accumulate counts. The table below presents the sections of a VERIFY rules file and summarizes their functions.

Section	Function
DECLARE	Creates working storage. Specifies new fields to save data or calculations during the verification.
INITIAL	Assigns or accepts starting values for declared fields and prints comments before any records are verified.
DETAILS	Processes each record, checking field information according to user-specified conditions. Prints or corrects errors as directed.
FINAL	Prints data or calculations from declared fields and comments after all records are processed.

The UPDATE Rules File

The UPDATE rules file specifies the calculations and replacements to be performed on the records in a data file, controls when an operation is to be executed, accepts data from the terminal, prints comments, and performs intermediate calculations.

The table below outlines the sections of an UPDATE rules file. The first part of the table applies when an optional activity file is not used. The second part pertains to updating procedures which use both a data and activity file.

Section	Function
Rules Sections Without an Activity File	
DECLARE	Creates working storage. Specifies new fields to save data or calculations during the updating.
INITIAL	Assigns or accepts starting values for declared fields, and prints comments before any records are updated.
DETAILS	Processes each record, updating fields according to user-specified conditions and calculations.
FINAL	Prints data or calculations from declared fields and comments after all records are processed.
Rules Sections With an Activity File	
DECLARE	Same as DECLARE above.
INITIAL	Same as INITIAL above.

(Table Continues)

Section	Function
BEFORE	Performs specified operations on a data record before seeking a matching activity record.
MATCHED *or* DETAILS	Performs specified operations if a matching activity record is found, using information from the activity and data records.
AFTER	Performs specified operations on a data record after the activity file procedure, whether or not a matching activity record was found.
UNAPPLIED	Provides access to unmatched activity records found during the UPDATE procedure.
FINAL	Same as FINAL above.

The REPORT Rules File

The REPORT rules file contains the user's report specifications, which define the layout, such as title, page headings, and margin; specify optional picture formats for printing individual fields; and request calculations, subtotals, and final totals. The table below outlines the sections of a REPORT rules file and summarizes their functions.

Section	Function
DECLARE	Creates working storage. Specifies new fields to save data or calculations during the report procedure, and may specify general picture formats.
INITIAL	Assigns or accepts starting values for declared fields, sets the margin and the number of lines per page, and specifies the title for the entire report.
HEADINGS	Specifies information to be printed at the top of each report page.
DETAILS	Processes each record; calculates and/or prints the desired field information according to user-specified conditions and instructions.
OTHERS	The OTHERS section processes each record that is excluded by conditional expressions in the REPORT command.
TOTALS	TOTALS specifies operations to be performed, such as subtotals or a new page, based on a change in the value of one or more fields.[1]
FINAL	FINAL specifies final calculations and comments to be printed after all records are processed.

STATEMENTS

The basic unit of a rules file is a statement. Each section of a rules file contains one or more statements. The user may write statements to change field information, print comments or data, accept data for a field from the terminal, and control the execution of one or more statements in the rules file.

1 – The TOTALS section is accessed for each record, but is executed only when a specified field(s) of the current record has a value different from that of the previous record. For example, if a field, DEPT, is specified in the TOTALS section, the TOTALS section is executed when DEPT changes from 1 to 2, 2 to 3, and so on.

The discussion of statements is organized in several parts. The first discussion presents all the instruction statements which may be used in VERIFY, UPDATE, and REPORT rules files: INPUT, TYPE, DONE, and replacement statements.[1] Next, the available control statements—IF, ORIF, ELSE, and DO statements—are outlined. The last discussion details the method of nesting statements in a rules file.

Instruction Statements

The instruction statements direct IML to perform calculations, replace field information, print comments or data at the terminal, accept data directly from the terminal, and terminate the execution of a rules section. The following are legal instruction statements illustrating the various capabilities mentioned above:

```
PRICE*QUANTITY TO AMT
TYPE "RECORD", RECNO, "CONTAINS PRICE AS ", PRICE
INPUT CODE
DONE
```

In addition, comments may be included in a rules file by preceding the comment with an exclamation point (!). Comments may appear alone on a line or may follow one or more statements on a line. For example, the following statements show valid comments:

```
10 ! NOW THE DATA FIELDS ARE UPDATED
10 DECLARE ! THESE FIELDS ARE COUNTERS
10 IF CODE=0; 0 TO HR.RATE ! FOR TERMINATED EMPLOYEES
```

Comments may not appear between statements on a line, nor may comments be followed by statements on the same line.

The user may write any instruction statement with a built-in limitation, using an IF or FOR modifier at the end of the statement. *NOTE: The words IF and FOR may be used interchangeably. Throughout this manual, the phrase* FOR modifier *means a FOR modifier or an IF modifier.* For example, the instruction

```
TYPE PARTNO,3B, "ONHAND LOW" FOR ONHAND<50
```

is executed only when the current value of ONHAND is less than 50. The user may type a FOR modifier to specify any relational expression.[2] For example,

```
0 TO HR.RATE FOR CODE=0 OR 99
PRINT "120 DAYS OVERDUE",3B,BALANCE IF LASTPAYT=0
DONE FOR HR.RATE=0 OR CODE #1 AND 2 AND 3 AND 4
```

are legal instruction statements with FOR modifiers.

The four basic instruction statements—replacement, INPUT, TYPE, and DONE statements—are detailed below. These statements may be used in all sections of any rules file except the DECLARE section.[3]

1 – The ON and PRINT statements for use in a REPORT rules file only are discussed on pages 138 and 147, respectively. Two additional statements for command file control are presented on page 172.
2 – The construction of relational expressions is detailed on page 22.
3 – The DECLARE rules section may contain only field definitions and general picture formats; it does not contain instruction or control statements.

Changing Field Information

The user enters a replacement statement to change the data in one or more fields. The form used is:

expression TO field list

The expression may be simply a number, or text in single or double quote marks, such as

```
35 TO LINES
"JONES" TO NAME
```

or the expression may be one or more field names in an arithmetic expression.[1] For example,

```
GROSS*.052 TO FICA
B*C+(D–E)/10.5 TO A,G,H
```

are legal replacement statements. The field list following the word TO specifies the field(s) to contain the expression value.

NOTE: When using an activity file, the user includes activity fields followed by :A in the replacement expression. The field list may not specify activity fields. For example,

ONHAND + SHPMT:A TO ONHAND

specifies an activity field named SHPMT in the replacement expression. Data fields may be followed by :D; it is not required, however, since fields are assumed to be data fields unless followed by :A.

Entering Data for a Field from the Terminal

The INPUT statement allows the user to enter data for a single field directly at the terminal. The form of the statement is

INPUT field name

where the named field may be a field in the data records or a declared field; it may *not* be an activity field. For example, the INITIAL section of a REPORT rules file might contain:

```
20 INITIAL
30 TYPE "HOW MANY LINES FOR BOTTOM MARGIN?",NCR
40 INPUT MARGIN
```

Line 30 prints the text in quote marks at the terminal, and line 40 accepts the value entered from the terminal for MARGIN.[2]

When IML reads an INPUT statement, the system pauses until a value is entered from the terminal. The user may, however, type only a carriage return to retain the current value of the field.

Printing Data and Comments at the Terminal

A TYPE statement directs IML to print information at the terminal. After a TYPE statement is executed, IML automatically returns the carriage. The form of the TYPE statement is

1 – The construction and evaluation of arithmetic expressions are discussed on page 21.
2 – MARGIN, an IML-declared field for REPORT, is presented on page 132.

TYPE item list

where a comma (,) is used between items.

The table below presents the items which may appear in a TYPE statement.

Item	Prints	Example
"any comments"	Text enclosed in single or double quote marks. As many as 80 characters may appear within quote marks.	TYPE "ERROR LOCATED"
field name	Value of specified field.	TYPE NAME
field name:A	Value of specified field in activity record.	TYPE SHPMT:A
utility field name[1]	Value of specified utility field.	TYPE @DATE,RECNO
nB	n blanks on line. The maximum is 80B.	TYPE ACCOUNT,2B,BALANCE
n'text'	Text enclosed in single or double quote marks n times. n times the number of characters in quote marks must be 80 or less.	TYPE 65'–'
CR	New line at this point, terminating logical line.	TYPE PARTNO,CR,DESC
LF	New line at this point, continuing logical line onto next physical line.	TYPE GROWTH,LF,SALES,UNITS
nothing	Blank line.	TYPE
NCR as last item	Suppresses carriage return at end of TYPE statement.	TYPE NAME,5B,NCR

Terminating a Rules Section

For some conditions, the user may want to ignore the remaining statements in a given rules section. The DONE statement provides a method for terminating a section of rules without executing any subsequent statements in that section.

The form of the DONE statement is simply:

DONE

Usually, the word DONE is followed by a FOR modifier, but it is not required. For example, suppose the user does not want to print page headings on the first page of his report; he might specify a DONE statement at the beginning of the HEADINGS section, such as:

```
DONE  FOR  PAGE=1
```

1 – Utility fields are presented on page 24.

When PAGE equals 1, the remaining statements in the HEADINGS section are ignored; on the other hand, after the first page of the report, the DONE statement is ignored and the HEADINGS statements are executed.

Control Statements

The user may control the execution of one or several instruction statements by preceding them with a control statement. IF, ORIF, ELSE, and DO are the control statements of the rules language. The overview which follows presents the function of each type of control statement and their interrelationship. The acceptable forms of each control statement are shown after the overview.

Overview

An IF statement is the first control statement used; for example,

10 IF MONTH=1 OR 2 OR 3

begins a set of conditional instructions.[1] The next statement might be an instruction such as

20 QTR1 + 1 TO QTR1

which is a replacement statement the user wants to execute only when the condition in the preceding control statement (that the current value of MONTH equals 1 or 2 or 3) is true.

ORIF statements are never required, but may follow to continue conditional instructions in an efficient manner. For example,

30 ORIF MONTH=4 OR 5 OR 6

continues the procedure when the preceding control statement specifies a false condition. The user may continue with as many ORIF statements and corresponding instructions as required; for example, the complete set of control statements and instructions to create counts for each quarter, based on the value of MONTH in each record, might be:

```
10 IF MONTH=1 OR 2 OR 3
20    QTR1 +1 TO QTR1
30 ORIF MONTH=4 OR 5 OR 6
40    QTR2 +1 TO QTR2
50 ORIF MONTH=7 OR 8 OR 9
60    QTR3 +1 TO QTR3
70 ORIF MONTH=10 OR 11 OR 12
80    QTR4 +1 TO QTR4
```

Note that an ORIF statement is evaluated only when the preceding IF or ORIF statement is false. Hence, for this task, the set of rules above is more efficient than a series of four independent IF statements, since IF statements are always evaluated.

1 – The statement is an abbreviation for IF MONTH=1 OR MONTH=2 OR MONTH=3. See page 24 for an explanation of abbreviating conditional expressions.

An ELSE statement is not required, but may follow an IF or ORIF statement and instructions to ensure that an instruction(s) is executed when all previous conditions (through the preceding IF statement) are false. For example, the user might include an ELSE statement, such as

```
90 ELSE TYPE "ERROR IN MONTH FIELD OF RECORD",RECNO
```

to print a message at the terminal when all conditions are false, locating an error in the data in this case.

A DO statement is needed when the user wants to terminate the effect of control statements and execute one or more instruction statements for all records being processed. Without a DO statement, an instruction statement is controlled by the preceding IF, ORIF, or ELSE statement. For example:

```
10 IF MONTH=1 OR 2 OR 3
20    QTR1 +1 TO QTR1
30 ORIF MONTH=4 OR 5 OR 6
40    QTR2 +1 TO QTR2
50 ORIF MONTH=7 OR 8 OR 9
60    QTR3 +1 TO QTR3
70 ORIF MONTH=10 OR 11 OR 12
80    QTR4 +1 TO QTR4
85 ELSE TYPE "ERROR IN MONTH FIELD OF RECORD",RECNO
90 DO
100 SALES + INVAMT:A TO SALES
110 UNITS + UNITS:A TO UNITS
120 COMM + INVAMT:A/6 TO COMM
•
•
•
```

The instructions on lines 100 through 120 are executed for each record processed. If the DO statement on line 90 were omitted, lines 100 through 120 would be associated with the ELSE statement on line 85.

The diagram on page 39 illustrates the conditions required to execute a set of instructions corresponding to an IF, ORIF, or ELSE statement, and the flow of control among IF, ORIF, ELSE, and DO statements. Note that a true condition causes the execution of all instructions which immediately follow; then IML proceeds to the next IF or DO statement, bypassing any intervening ORIF or ELSE statements. On the other hand, when a condition is false, IML ignores the immediately following instructions and proceeds to the next control statement: IF, ORIF, ELSE, or DO.

Forms

All control statements must appear at the beginning of a line; that is, a control statement may not follow a semicolon (;).

An IF or ORIF statement is specified in the form

$\left\{\begin{matrix}\text{IF}\\\text{ORIF}\end{matrix}\right\}$ condition where the condition is stated as a relational expression which may include field names and arithmetic and logical operators, if desired.[1] For example, the following are legal IF and ORIF statements:

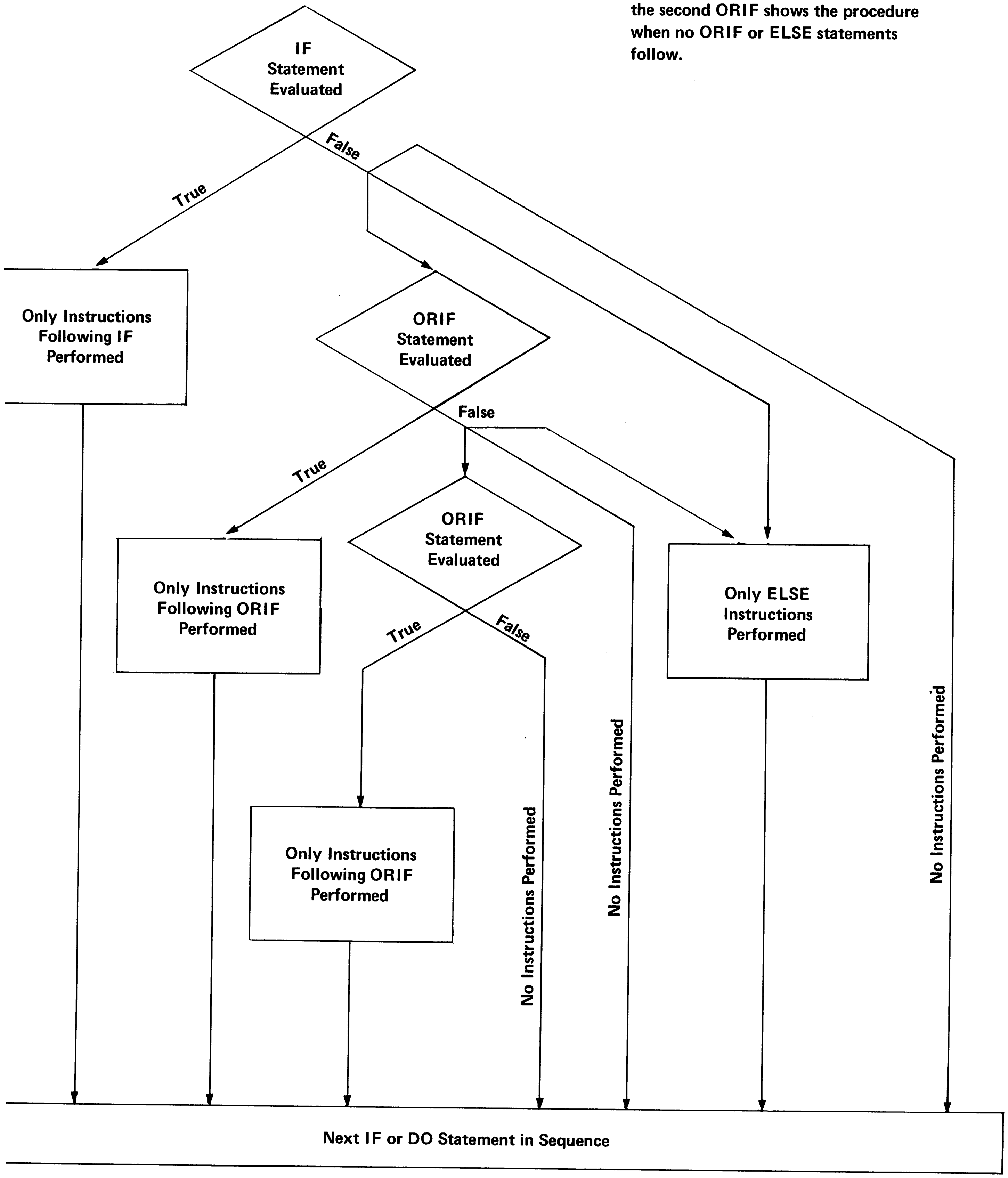
```
IF "CHEM" IN AREA OR AREA2
IF ONHAND–QTY:A<50
ORIF SALARY>300 AND <500
ORIF YTD.FICA>468.00
```

Note that an IF or ORIF statement is a complete statement. The instruction statements to be executed when the condition is true may appear in statements on the same line, separated by semicolons, or may appear on subsequent lines, or both. For example:

```
10 IF TCODE:A=1; ONHAND + QTY:A TO ONHAND
20 ORIF TCODE:A=2
30 ONHAND - QTY:A TO ONHAND
40 ORIF TCODE:A=3; 0 TO ONHAND
50 "DISCONTINUED" TO REMARKS; 0 TO STDORDER
60 TYPE "ITEM NUMBER", ITEMNO, " IS DISCONTINUED"
```

1 – Relational expressions are detailed on page 20.

NOTE: The first ORIF illustrates the general case; the second ORIF shows the procedure when no ORIF or ELSE statements follow.


IF Statement Evaluated
True
False
Only Instructions Following IF Performed
ORIF Statement Evaluated
True
False
Only Instructions Following ORIF Performed
ORIF Statement Evaluated
True
False
Only Instructions Following ORIF Performed
No Instructions Performed
No Instructions Performed
Only ELSE Instructions Performed
No Instructions Performed
Next IF or DO Statement in Sequence

An ELSE statement consists of the word ELSE. For example:

```
70 ELSE
```

The word ELSE may be followed by an instruction statement. For example,

```
70 ELSE TYPE "BAD TCODE IN ACTIVITY RECORD",RECNO:A
```

is equivalent to:

```
70 ELSE
80 TYPE "BAD TCODE IN ACTIVITY RECORD",RECNO:A
```

Additional instruction statements may appear on the same line, separated by semicolons, or may appear on subsequent lines, or both. For example:

```
100 ELSE TYPE "CODE ERROR"; ERRORCOUNT +1 TO ERRORCOUNT
110 TYPE "REENTER RATE CODE FOR", EMP.NO
120 INPUT RATE; TYPE "THANK YOU" FOR RATE='H' OR 'A'
```

The DO statement consists of the word DO. For example:

```
90 DO
```

Like the form of the ELSE statement, the word DO may be followed by one or more instruction statements, using semicolons between instruction statements on the same line. For example, the statements

```
90 DO
100 SALES + INVAMT:A TO SALES
110 UNITS + UNITS:A TO UNITS
120 COMM + INVAMT:A/6 TO COMM
```

are equivalent to:

```
90 DO SALES + INVAMT:A TO SALES; UNITS + UNITS:A TO UNITS
100 COMM + INVAMT:A/6 TO COMM
```

Nesting Statements

To test further conditions, when an IF, ORIF, or ELSE statement is true, the user may introduce lower level conditions. IML executes a lower level statement only if the preceding higher level IF statements specify a true condition for the current data. IF.1 is a first-level IF statement, as are ORIF.1, ELSE.1, and DO.1; IF.2 is a second-level IF statement; and so forth. For IML to execute a second-level statement, for example, a preceding first- and zero-level control statement must have true conditions. IML continues to execute successively lower levels of instructions until it encounters a condition that is false for the current data.

For example, assume M contains the month number, D contains the day number, and YR contains the last two digits of the year in the records being processed. The following statements, which check the accuracy of the D and YR fields when the month is February, illustrate the use of nested statements.

```
260   IF M=2
270     IF.1 D>29
280       TYPE "PLEASE REENTER DATE OF LAST REVIEW FOR ",NAME," EMP#",
           EMP.NO,CR,"DATE OF **",LAST.REV, " IS NOT POSSIBLE"
285       INPUT LAST.REV
286       TYPE
290     ORIF.1 D=29
300       IF.2 YR NOT = 72 AND 76
310         TYPE "PLEASE REENTER DATE OF LAST REVIEW FOR ",NAME,
             " EMP#",EMP.NO,CR,"FEBRUARY HAS ONLY 28 DAYS IN 19",
             YR,CR,"DATE OF **",LAST.REV," IS NOT POSSIBLE"
315         INPUT LAST.REV
316         TYPE
```

IML executes the first-level IF statement on line 270 only when the higher level IF statement condition on line 260 is true. When the level zero IF condition is false, IML skips lines 270 through 316, and so on until the next level zero IF, ORIF, ELSE, DO, or new section is encountered.[1] When the IF.1 statement on line 270 is true, lines 280 through 286 are executed, and the remaining statements associated with the ORIF.1 are ignored. When the IF.1 statement is false, IML proceeds to the ORIF.1 on line 290. When that is true, IML continues to the IF.2; otherwise, the remaining statements are ignored. When the IF.2 is true, the remaining statements are executed. Note that when the IF.2 statement is true, the ORIF.1 on line 290 and the IF on line 260 must also be true.

The first lower level instruction is an IF followed by a period (.) and an integer indicating the level. ORIF and ELSE statements followed by the same level number may be used to expand a lower level IF statement. The user may also include a DO statement with the lower level instructions in the same manner.

NOTE: When an IF statement is true, the next lower level DO statements are always executed.

The user must remember the way a set of IF statements is handled and apply this pattern in each level of instructions.[2] For example, when a third-level ORIF statement is false, IML skips the intervening statements and proceeds to the next ORIF.3, ELSE.3, or DO.3 statement. Any nesting in the preceding IF.3 and ORIF.3 statements is bypassed. Similarly, when a level zero IF statement is false, all following lower level instructions are ignored.

1 – Rules file sections are discussed on page 30.
2 – The diagram on page 39 applies within each level of control statements.

SECTION 4
FILE CREATION

The Information Management Library makes the initial data entry procedure simple, yet allows the user to describe the most economical way to store and access large quantities of information on the TYMCOM-IX system. First, the user specifies his format requirements; then he enters the data in free format with all editing and prompting facilities. Finally, the user may employ IML's verification capability to check the accuracy of the entered data.

This section presents IML's DEFINE, CREATE, and VERIFY programs, which handle file description, creation, and verification.

DEFINE

DEFINE allows the user to describe an efficient set of formatting specifications for information storage and retrieval. The user need not enter the data in fixed format; in writing the data file, CREATE, another IML program, obeys the description file created by DEFINE.

The discussion of DEFINE treats separately symbolic data files and binary data files, then describes editing a description file.

A symbolic file stores *all* information in the actual character representation. This allows the user to print the file directly with the EXECUTIVE TYPE command or read the file into EDITOR. Binary files store the information in a compact internal format, adding greater economy to file storage; however, a binary file cannot be printed directly. In IML operations, both symbolic and binary files offer the same convenient, direct accessibility.

All data files consist of records. A *record* contains one or more related pieces of information forming a unit. Each piece of information in a record is called a *field.* The user names each field and specifies its location within the record, its length, and the type of data it contains: numeric or character.

When all records in a file contain the same number of characters and have a constant length for the life of the file, they are called *fixed length records.* When the length of all records changes during the life of the file or the records in the file are of different length, they are called *variable length records.*

Fixed length records offer the most straightforward description and economy; every field is in a fixed location with no unnecessary storage between fields. Variable length records offer extra economy when the last fields are to be blank for part of the file's life, but require more detail in structuring.

The term *data file* refers to the information file containing the records; it may be a master data file, an activity data file, or a data file maintained alone. A *description file* contains no data, but specifies the parameters of a data file, providing the following information:

1. Type of file – BINARY or SYMBOLIC
2. Type of records – FIXED or VARIABLE
3. Record definition
4. Field descriptions
   a. Name
   b. Data type – CHARACTER, NUMERIC, and binary INTEGER, REAL, or DOUBLE
   c. Location
   d. Length
   e. Decimal places in numeric field

The DEFINE program incorporates an additional capability, allowing the user to store more than one data file description and set of rules on a single file.[1] Multiple description files significantly reduce the storage required for description files and rules files. After mastering this section, see page 161 for documentation of multiple description files.

Sample Problem

The example below is an introduction to the use of the DEFINE program. A user writes a description specifying a symbolic file containing fixed length records.

```
-DEFINE INVDES↵
INVDES.. NEW FILE↵
SHORT PROMPTS? (Y OR N): N↵          Other examples in this section show the short prompts.
FILE TYPE? (BINARY OR SYMBOLIC): SYMBOLIC↵
FIXED OR VARIABLE LENGTH RECORDS: FIXED↵
LENGTH OF RECORD IN CHARACTERS IS: 40↵


FIELD INFORMATION REQUIRED IS:
 (1)FIELD NAME.
 (2)TYPE OF DATA IN THE FIELD
  TYPES ARE:
  C=CHARACTER (ALPHA OR ALPHANUMERIC)
  N=NUMERIC
 (3)STARTING CHARACTER POSITION OF THE FIELD.
 (4)LENGTH OF FIELD IN NUMBER OF CHARACTERS.
 (5)DECIMAL PLACES IN NUMERIC FIELD.
```

1 – The complete discussion of rules files begins on page 27.

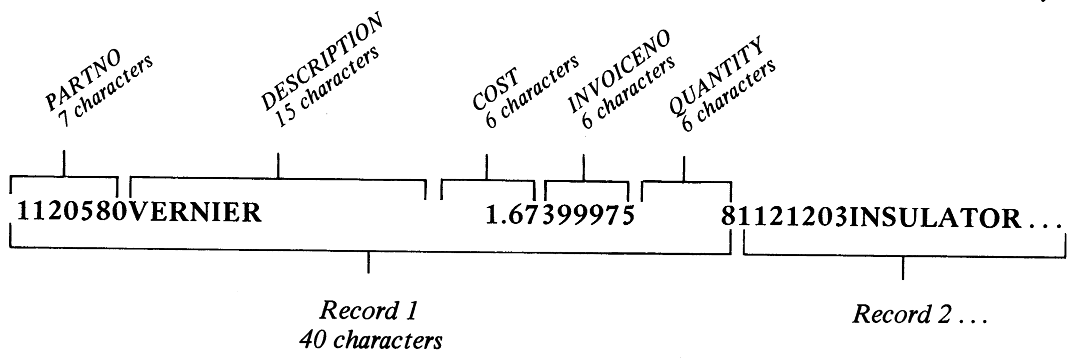
```
  NAME, TYPE, START, LENGTH, DECIMAL PLACES
: PARTNO,N,1,7↵
: DESCRIPTION,C,.,15↵
: COST,N,.,6,2↵
: INVOICENO,N,.,6↵
: QUANTITY,N,.,6↵
: ↵
DESCRIPTION FILE WRITTEN.

-
```

The first field, PARTNO, contains numeric information. The field starts in position 1 and contains a maximum of seven characters.

The period indicates that the field begins in the next position after the previous field.

The description file, INVDES, specifies a file which contains data stored in this way:



Calling DEFINE

The user calls the DEFINE program by typing

```
         description
-DEFINE  file        ↵
         name
```

in the EXECUTIVE. The user may also type

-<u>DEFINE</u>↵

and the system prompts for the name of a file on which to store the description.

After the user specifies the description file name, the system prints the file name plus the appropriate NEW FILE or OLD FILE message, which the user confirms by typing a carriage return or aborts by typing an alt mode/escape.

Next, the system prints

SHORT PROMPTS?

which the user answers with YES or NO, followed by a carriage return. If the response is YES, DEFINE uses an abbreviated form to request the description;[1] if the response is NO, the prompts appear as shown in the first example.

The system prints

FILE TYPE: BINARY OR SYMBOLIC:

1 – The YES response to SHORT PROMPTS is illustrated in subsequent examples.

to determine the type of data file. The user answers with BINARY or SYMBOLIC; these responses may be abbreviated to one or more letters, followed by a carriage return.

Describing a Symbolic File

A symbolic file consists of fixed length records or variable length records. Either type of record may contain as many as 1024 characters. To determine the type of records in the file, the system prints:

```
FIXED OR VARIABLE LENGTH RECORDS:
```

The user responds with FIXED or VARIABLE, abbreviated if desired, followed by a carriage return.

Describing Fixed Length Records

For fixed length records, the user must first specify the length of each record. The system prompts:

```
LENGTH OF RECORDS IN CHARACTERS IS:
```

The user enters a number to specify the record length. If the last character in the record is a carriage return, the carriage return must be included in the character count.[1]

Next, the user must define the fields within a record by entering the name of the field, the data type (C or N), its starting position in the record, and the length of the field in characters. The user must also enter a number of decimal places if he wants to reserve space for digits to the right of the decimal point. The system prompts for the field descriptions as follows:

```
FIELD INFORMATION REQUIRED IS:
  (1)FIELD NAME.
  (2)TYPE OF DATA IN THE FIELD
   TYPES ARE:
   C=CHARACTER (ALPHA OR ALPHANUMERIC)
   N=NUMERIC
  (3)STARTING CHARACTER POSITION OF THE FIELD.
  (4)LENGTH OF FIELD IN NUMBER OF CHARACTERS.
  (5)DECIMAL PLACES IN NUMERIC FIELD.


   NAME, TYPE, START, LENGTH, DECIMAL PLACES
 :
```

NOTE: The user need not describe all fields in the records. He must, however, describe each field he wants to use in subsequent IML procedures.

The field name may contain as many as 31 characters, must begin with a letter, and must not contain spaces. The field name may be any combination of letters, the digits 0 through 9, the period (.), and the character @, except the following words which are reserved for the IML vocabulary:

1 – To enter the carriage return character as data, the user must precede the carriage return with a Control V when entering the data at the terminal or on a file. The Control V does not, however, belong in the character count. This procedure is explained with the discussion of CREATE, page 61.

ABORT	DOES	HEADINGS	MIN	SKIP
ABORTING	DONE	HPOS	NCR	SPACE
AFTER	END	IF	NOT	START
AND	ENDING	IN	NUMERIC	STARTING
AS	ENDS	INITIAL	OFFHEADINGS	STARTS
AVG	ELSE	INPUT	ON	SUM
BASED	FINAL	IS	ONHEADINGS	TAB
BEFORE	FINISH	LENGTH	OR	TO
BY	FINISHING	LF	ORIF	TOF
CR	FOR	LINE	OTHERS	TOP
DECLARE	FROM	LINES	PAGE	TYPE
DELETE	HAS	MARGIN	PER	USING
DETAILS	HAVE	MATCHED	PRINT	VERIFY
DO	HAVING	MAX	RECNO	WITH

The data type may be either C (character field) or N (numeric field). A numeric field must contain a legal number. Legal numbers are expressed as integers, such as 268; fixed point, such as 2. or −268.173; or in real or double precision scientific notation, such as 5.7E+03,1.5D−12.[1] If a field contains any character other than those allowed in a legal number, it is a character field. For example, the DESCRIPTION field specified in the INVDES description file has C type data because it contains nonnumeric information. The user specifies the PARTNO field as N type data because it contains only numeric information.

NOTE: The user may specify any field as C type data; however, arithmetic operations are permitted only on numeric fields.

The starting position of the field is 1 plus the number of characters, including spaces, from the beginning of the record. For example:

: **PARTNO,N,1,7** ↵
: **DESCRIPTION,C,8,15** ↵

The DESCRIPTION field begins in position 8 because seven characters precede it.

A period (.) may be used as a starting position to indicate that a field immediately follows the last field described. For example,

: **INVOICENO,N,.,6** ↵

specifies that the INVOICENO field begins in the next position after the previous field.

NOTE: The user need not describe the fields in the order they appear in the records, provided that he specifies the starting position of each field with a number. When using a period for starting position, however, the user must describe the fields in record order.

The length of the field is the maximum number of characters allowed in the field. The field length must include space for minus signs, decimal points, and E or D scientific notation. In the example above, the user specifies the length of INVOICE as 6.

The specified number of decimal places indicates the number of digits allowed to the right of the decimal point. This number must be specified when the user wishes to enter data with decimal points in the field; otherwise, IML assumes that the number of decimal places is zero. For example,

: **COST,N,.,6,2** ↵

1 – A quantity is written in scientific notation as a number times a power of 10. For example, $5004=.5004\times10^4$ may be expressed as .5004E+04 or 5.004E+03 in E notation. D used instead of E indicates a double precision number in scientific notation.

specifies COST as a numeric field with a maximum length of 6, including a decimal point and two digits to the right of the decimal point. Therefore, the largest number allowed in the COST field is 999.99.

After the user enters the field descriptions, he types a carriage return in response to the colon prompt. The system responds

```
DESCRIPTION FILE WRITTEN.
```

and returns control to the EXECUTIVE. The user may wish to refer to page 44 for an example of this procedure.

Describing Variable Length Records

When the user specifies variable length records, the system prompts for the type of variable length record with the message:

```
LENGTH OF RECORD IS DETERMINED BY:
   C=COUNT OF CHARACTERS AS FIRST FIELD.
   L=NUMBER OF LINES PER RECORD.
   S=SPECIAL TERMINATOR CHARACTER.
WHICH TYPE:
```

The user enters C, L, or S, as appropriate, followed by a carriage return.

If the user enters C, each record contains a field at the beginning of the record, which contains the count of the characters in the record. For example:

```
                Record 1                 Record 2    Record 3
032THIS RECORD HAS 32 CHARACTERS007ABCD015XXXYYYZZZZZZ...
```

NOTE: The characters in the count field are part of the record and are included in the count.

The system prompts:

```
NUMBER OF DIGITS IN THE COUNT IS:
```

The user enters an integer, specifying the length of the count field. The count field length must accommodate the character count for the longest record in the file. For example, if the longest record contains 620 characters, the user should enter 3 as the number of digits in the count. In this case, if the count of another record is 49, it is written as 049. The count includes any carriage returns in the record, as well as the count characters.[1]

If the user types L, each record consists of a fixed number of lines. A line is defined as a series of characters terminated by a carriage return. For example, a file containing records as follows may be defined as a variable length record file with records consisting of three lines.

1 – The user who intends to use CREATE to write the data file need only be concerned with an estimated length of the longest record so he can determine the maximum number of digits for the count field. CREATE writes the actual character count for each record when the data is entered.

```
117644 705.95↵              ┐
JAN 95.00↵                  ├─ Record 1
EGYPT PUBLISHING↵           ┘
115673 159.73↵              ┐
JAN 65.00↵                  ├─ Record 2
KAY LITHOGRAPHERS↵          ┘
115091 254.95↵              ┐
JAN 21.00↵                  ├─ Record 3
NORTHERN PULP↵              ┘
:
```

The system prompts:

NUMBER OF LINES PER RECORD IS:

The user enters an integer specifying the number of lines in each record. Each record must contain the same number of lines.

NOTE: The L option with the number of lines greater than 1 should be avoided if the user wants to use the description file in a CREATE command. The L option is suggested only for describing an existing data file.

If the user types S, each record is terminated by a special character or the first carriage return after a special character. For example, a file containing records

```
       Record 1                Record 2                  Record 3
┌───────────┴───────────┐┌───────────┴───────────┐┌───────────┴───────────┐
95014CUPERTINO,CA/10021NEW YORK,NY/55414MINNEAPOLIS,MN/...
```

may be described as a variable length record file with records terminated by a slash (/).

The system prompts:

TERMINATOR CHARACTER IS:

The user enters the record terminator character. The system reads all characters up to and including the terminator character as the current record; the next record begins with the character immediately following the terminator character. If the user wishes to include all characters after the terminator character up to and including the next carriage return as part of the current record, he responds to the DEFINE request for the terminator character with the terminator character followed by a dollar sign ($). For example:

Sample Records in a File	Terminator Character
Record 1 / *Record 2*: 95014CUPERTINO,CA/10021NEW YORK,NY/...	/
JANE DRUSS ↵ 3614 14 AVE SE↵ MINNEAPOLIS, MN 55414↵ — *Record 1* FRANK RICHARDS↵ 555 MADISON AVE↵ NEW YORK, NY 10017↵ — *Record 2* *The system reads past the comma to the next carriage return.*	,$

Note that a carriage return should not be entered as the terminator character, since this would indicate that each record contains only one line. In that case, the user should describe the records with the L option and specify the number of lines as 1.

Next, the user defines the fields within a record, as described on page 46. The discussion which follows includes additional instructions for describing fields in variable length records.

The starting position specification for a field on the second or subsequent line in a record includes preceding carriage returns and blanks. For example, if a file contains records of three lines, a field position on the third line includes the two preceding carriage returns. To describe records such as

```
JONES ALEXA    ↵        The first line contains the name and three trailing blanks to fill the field.
175 CRESCENT DR↵        The address is on the second and third lines
FOREST HILLS, NY 11375↵
```

the user may specify two fields as follows:

```
: CLIENT,C,1,14 ↵      The CLIENT field contains 14 characters.
: ADDRESS,C,16,40 ↵    The ADDRESS field begins in position 16 because the carriage return following the
                       CLIENT field appears in position 15.
```

Note that the ADDRESS field length of 40 includes space for two carriage returns as well as the characters in the address.

If the user describes variable length records in terms of a character count at the beginning of the record, he *must remember the count field* when determining the starting position of all other fields. For example, if the number of digits in the count is 3, the starting position of the first data field would be 4.

The following example illustrates the use of DEFINE to describe a symbolic file containing variable length records with a character count field at the beginning of each record.

```
-DEFINE DESC↵
DESC.. NEW FILE↵
SHORT PROMPTS? (Y OR N): Y↵
BIN OR SYM: SYM↵
FIX OR VAR: VAR↵
VAR TYPE(C,L,S): WHICH TYPE: C↵
```

```
NUMBER OF DIGITS IN THE COUNT IS:2↵
  NAME, TYPE, START, LENGTH, DECIMAL PLACES
: ITEM.NO,N,3,4↵
: DESCRIPTION,C,.,20↵
: IN.STOCK,N,.,5↵
: YTD.SOLD,N,.,6↵
: JAN,N,.,4↵
: FEB,N,.,4↵
: MAR,N,.,4↵
: APR,N,.,4↵
: MAY,N,.,4↵
: JUN,N,.,4↵
: JUL,N,.,4↵
: AUG,N,.,4↵
: SEP,N,.,4↵
: OCT,N,.,4↵
: NOV,N,.,4↵
: DEC,N,.,4↵
: ↵
DESCRIPTION FILE WRITTEN.

-
```

The maximum record length is 99 characters including the count field.

The first data field begins in position 3, since the character count field requires two digits.

Describing a Binary File

A binary file may consist of fixed length records or variable length records. When the system prompts

FIXED OR VARIABLE LENGTH RECORDS:

the user responds appropriately with FIXED or VARIABLE, abbreviated if desired, followed by a carriage return.

Record Length

The record length depends on the types of data stored in the fields of the record. A binary record may contain as many as 1023 characters. Binary records consist of the data types in the table below; the required storage for each type is specified in number of characters.

Type of Data in Field	Number of Characters in Storage
I – Binary integer	3
R – Binary real	6
D – Binary double precision	9
C – Character	Equal to character count for symbolic representation of data, rounded up to a multiple of 3.[1]

1 – When a record contains several contiguous character fields, the individual field lengths need not be a multiple of 3; that character portion of the record, however, must be a multiple of 3.

If a user specifies fixed length records, he must specify the length of the records. The system prints:

LENGTH OF RECORDS IN CHARACTERS IS:

The user enters a number to specify the length of records. The record length must be a multiple of 3. The user computes the record length in accordance with the storage required for the various types of binary data. Note that the number of storage characters required for numeric data does not correspond to the number of symbolic characters.

For example, if records contain four contiguous fields—two integer (I) fields, one real (R) field, and one character (C) field which is 24 characters long—the user specifies the record length as 36. This is computed as follows:

Field	Type of Data	Storage Required	Example of Field Contents
1	I	3	14560
2	I	3	1000
3	R	6	1907.45
4	C	24	HORIZON DEVELOPERS INC.
		36	*Actual length of record written in binary form*

If the user specifies variable length records, he does not specify a record length in the description file. Instead, the system reserves the first three characters of each record for a binary integer which is a count of the number of characters in the record. When CREATE writes variable length records for a binary file, it automatically places the actual character count at the beginning of the record.[1] The record length includes the three characters needed for the character count.

Field Information

After the user specifies the type of record, fixed or variable length, he must define the fields within a record. He specifies the field information for binary files of fixed or variable length records as follows. The system prompts:

```
FIELD INFORMATION REQUIRED IS:
 (1)FIELD NAME.
 (2)TYPE OF DATA IN THE FIELD
```

1 – The use of CREATE is detailed on page 58.

```
  TYPES ARE:
  C=CHARACTER (ALPHA OR ALPHANUMERIC)
  I=BINARY INTEGER
  R=BINARY REAL
  D=BINARY DOUBLE
 (3)STARTING CHARACTER POSITION OF THE FIELD.
 (4)LENGTH OF FIELD IN NUMBER OF CHARACTERS.
 (5)DECIMAL PLACES IN NUMERIC FIELD.

  NAME, TYPE, START, LENGTH, DECIMAL PLACES
:
```

The rules for naming fields are detailed on page 46. The legal data types for binary fields are C, I, R, and D. The user specifies the starting position of a field in accordance with the type of data and the storage required, as detailed on page 52. Thus, if an integer (I) field begins in position 4, the next field starts in position 7. If a real (R) field begins in position 4, however, the next field starts in position 10.

The starting position of each field must be position 1 or must be one greater than a multiple of 3; for example, 4, 7, 10, 13, and 16 are valid starting positions.[1] The user may indicate easily the starting position of a contiguous field using the period (.). Typically, the user specifies all but the first starting position using the period. If, however, the user does not want the fields to follow each other immediately, he must specify the starting positions with numbers. The user need not describe the fields in the order they appear in the records, provided that he specifies the starting position of each field with a number. When using a period for starting position, however, the user must describe the fields in record order.

When describing variable length records, the starting position of the first field must be 4, since the first three characters in each record are reserved for the record length.

The field length, on the other hand, specifies the maximum number of characters needed to print the data in the field in symbolic form. The field length must allow for minus signs, decimal points, and E or D scientific notation.[2] The field length, therefore, is determined by the longest datum in the field. For example, if the longest datum in an R field is –14367.941, the user specifies the field length as 10.

The decimal places specification states the number of decimals to be printed. IML rounds to the appropriate number of decimal digits. For example, if the COST field is defined as

: <u>COST,R,.,7,2</u>↵

and the value of COST is 15.3864, the system prints the value of COST as 15.39. If COST is defined with three decimal places, the value of COST is printed as 15.386.

The examples following illustrate the creation of description files for binary fixed length and variable length records.

1 – Contiguous character fields are the exception to this rule.
2 – Scientific notation is described on page 47.

Example 1

The user wishes to describe a binary file consisting of fixed length records. Each record contains four contiguous fields. The table below shows the information used to enter the description in DEFINE.

Field Name	Type	Maximum Length (in symbolic form)	Binary Storage	
PART	I	6	3	
DESCRIPTION	C	24	24	
VALUE	R	8 *(including decimal point)*	6	
ONHAND	I	5	3	
			36	*Record Length*

```
-DEFINE INVENTORYDES↵              The system saves the description on a file
INVENTORYDES.. NEW FILE↵           named INVENTORYDES.
SHORT PROMPTS? (Y OR N): Y↵
BIN OR SYM: BIN↵
FIX OR VAR: FIX↵
LENGTH: 36↵                        The record length is always a multiple of 3.
  NAME, TYPE, START, LENGTH, DECIMAL PLACES
: PART,I,1,6↵                      DESCRIPTION begins in the next position after the PART
: DESCRIPTION,C,.,24↵              field (indicated by the period and equivalent to typing 4)
: VALUE,R,.,8,2↵                   and reserves space for 24 characters.
: ONHAND,I,.,5↵                    ONHAND is the fourth contiguous field.
: ↵
DESCRIPTION FILE WRITTEN.

-
```

Example 2

The user wishes to describe a binary file consisting of variable length records. Each record contains three contiguous fields: ACCOUNT, BALANCE, and ADDRESS. The ADDRESS field is of variable length and may contain a maximum of 36 characters.

```
-DEFINE ACCOUNTDES ↵
ACCOUNTDES.. NEW FILE↵
SHORT PROMPTS? (Y OR N): YES↵
BIN OR SYM: BIN↵
FIX OR VAR: VAR↵
  NAME, TYPE, START, LENGTH, DECIMAL PLACES
: ACCOUNT,I,4,7↵
: BALANCE,R,.,8,2↵
: ADDRESS,C,.,36↵
: ↵
DESCRIPTION FILE WRITTEN.

-
```

Note that the ACCOUNT field begins in position 4 because the first three positions contain the character count field.

Note that the number of characters in the ADDRESS field determines the length of a record. For example, when ADDRESS has 21 characters, the record length is 33. This is computed as follows:

Field	Storage (in characters)	
Character count as integer	3	
ACCOUNT as integer	3	
BALANCE as real	6	
ADDRESS as character	21	
	33	*Record length*

When the ADDRESS field has the maximum number of characters, that is, 36, the record length is 48.

Editing a Description File

The user may alter an existing description file by entering a DEFINE command of the form:

```
                   description
-DEFINE  [EDIT]    file          ↵
                   name
```

First, the general file characteristics are printed, and then the system prompts for new information on general file characteristics. Next, the field descriptions are printed individually, and after each field description is printed, the user may enter a new field description to replace it, leave the field description unchanged, delete the field description, or insert a new field description. Finally, the user may append new field descriptions when the colon prompt appears. After the following example, each possibility is explained in detail, including additional forms for performing a given procedure on several field descriptions at once.

The example below shows the user altering the INVENTORYDES description file created on page 54. The user wants to insert a new I field named TCODE before the VALUE field, change the VALUE field description, and append an R field named PRICE.

```
-DEFINE [EDIT] INVENTORYDES↵

OK.


$$$
***BIN OR SYM:
BIN                          The system prints the general file characteristics.
***FIX OR VAR:
FIX
***LENGTH:
36


SHORT PROMPTS? Y↵
BIN OR SYM: BIN↵
FIX OR VAR: FIX↵             The user increases the record length by nine characters, since he wants to add
LENGTH: 45↵                  an I field and an R field.
  NAME, TYPE, START, LENGTH, DECIMAL PLACES
PART,I,1,6                   The system prints the first field description.
↵                            The user does not want to change the field description.
DESCRIPTION,C,.,24
↵
VALUE,R,.,8,2
I↵                           The user wants to insert a new field description before the VALUE field.
: TCODE,I,.,1↵               The system prompts with a colon.
VALUE,R,.,8,2                The next field description is printed.
VALUE,R,.,6,2↵               The user changes the field length.
ONHAND,I,.,5
↵                            The system prompts after the last field description, and the user enters an
: PRICE,R,.,8,2↵             additional field description.
: ↵                          A carriage return terminates entry of field descriptions.

FINISHED.


-
```

The revised INVENTORYDES description file is shown below.

```
-TYPE INVENTORYDES

$$$
***BIN OR SYM:
BIN
***FIX OR VAR:
FIX
***LENGTH:
45                  Changed record length
***  NAME, TYPE, START, LENGTH, DECIMAL PLACES
PART,I,1,6
DESCRIPTION,C,.,24
TCODE,I,.,1         Inserted field description
VALUE,R,.,6,2       Changed field description
ONHAND,I,.,5
PRICE,R,.,8,2       Appended field description


-
```

The user changes an existing field description by reentering the description after the line is typed. The old line serves as an image for the new line, allowing the use of control characters to enter the changed field description. For example, the system prints a field description:

```
VALUE,.,8,2
O^c8VALUE,R,.,6,2     The user types Control O followed by 8, copying the old line up to but
                      not including the 8. He then types the rest of the new field description.
```

The user leaves an existing field description unchanged by typing only a carriage return after the line is typed. For example

```
PART,I,1,6        The system prints a field description.
                  The user wants to leave it as is.
```

The user may precede the carriage return with an integer specifying the number of consecutive field descriptions he wants to remain the same. For example:

```
FLD2,C,.,7        The system prints a field description.
3                 The user wants to preserve this field description as well as the next two field descriptions.
```

DEFINE then prompts with the next field description the user may want to change. Note that if the user specifies an integer greater than the number of field descriptions left, DEFINE assumes an end to editing and copies the remaining field descriptions. If the integer equals the remaining number of field descriptions, however, the remaining lines are copied, and DEFINE then prompts with a colon, allowing the user to append new field descriptions. The procedure terminates when the user types a carriage return after the colon prompt.

To delete a field description, the user types a D, followed by a carriage return, after DEFINE prints the line. For example:

```
FLD9,N,.,8,2
D
```

The user may precede the D with an integer specifying the number of consecutive field descriptions he wants to delete. If the integer exceeds the number of field descriptions remaining, the remaining lines are deleted and the editing procedure terminates. If the integer equals the number of remaining field descriptions, however, the remaining lines are deleted, and DEFINE prompts with a colon, allowing new field descriptions to be appended. The QUIT command may be used to delete all remaining field descriptions starting with the current position. Once the user types QUIT followed by a carriage return, DEFINE writes the new description file as entered to that point and returns control to the EXECUTIVE.

The user may insert a new field description before the one just typed by the system by typing an I followed by a carriage return. The user may precede the I with an integer specifying the number of field descriptions to be inserted. For example:

```
FLD12,C,.,21      The system prints a field description.
3I↵               The user wants to insert three new field descriptions.
:                 DEFINE prompts with a colon.
```

The user may type a carriage return after the colon prompt to terminate the insert mode before the specified number of field descriptions is entered. After the insertion is complete, DEFINE returns to the editing procedure by reprinting the field description typed before the insertion(s). For example:

```
RATE,C,6,1
3I↵               The user wants to insert three field descriptions before the RATE field.
: FLDX,C,.,2↵
: FLDY,C,.,2↵
: FLDZ,C,.,2↵
RATE,C,6,1
```

CREATE

CREATE writes IML or RETRIEVE data files while allowing the user to enter the information in free format. In addition, CREATE makes it possible to add new records to an existing data file. Before CREATE can write or append to a data file, the user must describe the characteristics of the data file with a description file, created by the DEFINE program, or a RETRIEVE structure file.

The CREATE command instructs the system to write the entered data on a file according to the specified description. The user calls CREATE from the EXECUTIVE. The general form of the CREATE command without options or additional clauses is

```
         data     description           { T               }
–CREATE  file   : file          FROM    {                 }↵
         name     name                  { input file name }
```

to create a new file, or

```
                             data     description           { T               }
–CREATE  APPENDING  TO       file   : file          FROM    {                 }↵
                             name     name                  { input file name }
```

to add records to an existing file.

When the user omits a description file name from the command or types a carriage return after the prompt for a description file name, IML searches for the proper RETRIEVE structure file in the user's directory. For example, in the command

```
-CREATE APPENDING TO ORDERS FROM T ↵
```

the system searches the user's directory for a RETRIEVE structure file corresponding to ORDERS. The CREATE program writes the new records in the same format as the existing RETRIEVE data base, in binary code if the existing file is binary and with a carriage return at the end of each record if the file is symbolic. RETRIEVE scrambled files are not compatible with IML.

The data entry procedure using CREATE with RETRIEVE data bases is identical to the procedure followed to prepare IML data files. The user may enter his data from the terminal, or he may specify the name of a file which contains the data. Both types of data entry are discussed below. Finally, the various options and clauses available for both data entry methods are presented on page 63.

The general form of the CREATE command is:

```
                                         data    description
-CREATE  {[PROMPT]   }  APPENDING TO   file  :  file         FROM  {T              }
         {[character]}                 name     name               {input file name}

{BY NAME        }
{BY LIST        } ↵
{WITH field list}
```

[PROMPT] and [character] are the CREATE options; APPENDING TO, BY NAME, BY LIST, and WITH field list are the optional clauses.

The examples in this section show the use of a description file, INVDES, and the CREATE program to generate various information files. INVDES describes a symbolic file consisting of fixed length records of 40 characters. Each record contains five fields: PARTNO, DESCRIPTION, COST, INVOICENO, and QUANTITY. All the fields contain numeric information except DESCRIPTION which contains alphanumeric information.

```
-TYPE INVDES ↵             This file was created without short prompts on page 44.

$$$
***FILE TYPE? (BINARY OR SYMBOLIC):
SYMBOLIC
***FIXED OR VARIABLE LENGTH RECORDS:
FIXED
***LENGTH OF RECORD IN CHARACTERS IS:
40
***  NAME, TYPE, START, LENGTH, DECIMAL PLACES
PARTNO,N,1,7
DESCRIPTION,C,.,15
COST,N,.,6,2
INVOICENO,N,.,6
QUANTITY,N,.,6


-
```

Data Entry

CREATE allows the user to enter data from the terminal or from an input file. Each data entry method is discussed separately below.

Entering the Data at the Terminal

CREATE assists the user entering data at the terminal by providing automatic prompts, immediate error diagnostics, and a complete editing facility.

In the example below, the user creates an invoice information file, entering the data at the terminal; he names the file INV1. The file named INVDES describes to CREATE the specifications for the new file.

```
-CREATE INV1:INVDES FROM T↵
INV1.. NEW FILE↵

OK.

DATA MUST BE ENTERED IN THE FOLLOWING ORDER:    The INVDES description file
PARTNO N 7                                      specifies this field information.
DESCRIPTION C 15
COST N 6 2
INVOICENO N 6
QUANTITY N 6

1: 1120557,FOCUS TUBE,9.60,407712,32↵     The user enters the data, terminating
2: 1120572,GASKET,.39,406678,50↵          each field with a comma and each
3: 1120545,AIR VALVE NUT,.45,404331,12↵   record with a carriage return.
4: 1120576,YOKE,30.55,399941↵   When the user omits a field in the
QUANTITY: 8↵                    record, the system prompts him.
5: ↵                      A carriage return after the
4 RECORDS CREATED.        colon terminates data entry.

-
```

The listing below shows the records that were written from the CREATE data using the INVDES description.[1] Each line represents one record.

```
PARTNO     DESCRIPTION      COST  INVOICENO  QUANTITY

1120557FOCUS TUBE         9.60407712      32
1120572GASKET              .39406678      50
1120545AIR VALVE NUT       .45404331      12
1120576YOKE              30.55399941       8
```

1 – This file cannot be printed directly with the TYPE command, since it does not contain the carriage return character between records. The SELECT command is used to produce the listings in this section.

When the system prompts the user for missing field information in the middle of the record, the user may enter all the remaining information after the prompt. For example:

```
1: 1120576,YOKE↵
COST: 30.55,399941,5↵
2:
```

Alternatively, the user may enter only the requested field information, followed by a carriage return, and let the system prompt for the next field. For example:

```
1: 1120576,YOKE,30.55↵
INVOICENO: 399941↵
QUANTITY: 5↵
2:
```

The user may indicate empty fields by typing a comma or a slash (/). If the rest of a record is to be blank, the slash terminates data entry without entering the remaining fields in the record.[1] The comma is used to selectively indicate empty fields within a record and is used whenever at least one field is to be given a value. The user may indicate empty fields by typing two commas to indicate one empty field, three commas to indicate two consecutive empty fields, and so on. A comma at the end of the record indicates that the last field is empty. When the user wishes to create records with numerous empty fields, the BY NAME clause is most helpful. See the explanation on page 65.

During data entry at the terminal, all the EDITOR line editing characters are available in CREATE.[2] These include the control characters to correct the current line being typed, as well as the control characters available to use the last line typed as an image for the current line. Control V is available to enter any instruction character literally. For example, to enter a comma or a carriage return in the data, the user must precede the character with a Control V; otherwise, such a character signals the end of a field or record.

The input process follows the same conventions as in the EDITOR APPEND mode, except that Control Q can delete only to the last carriage return. The up arrow (↑), Shift N on many terminals, is available to restart input for the current record if the user wishes to delete beyond the previous carriage return. The user may type an up arrow, followed by a carriage return, repeatedly to delete several preceding records. To use this capability, the user must type an up arrow as the first and only character of a line.

In addition, the user may type a back slash (\), Shift L on many terminals, to reenter the last field entered. To use this capability, the user must type a back slash, followed by a carriage return, at the beginning of the next field. For example:

```
1: 1120576,YOKE,55.30,\↵        The user wants to reenter the last field entered.
COST: 30.55,399941,5↵
2:
```

1 – Null values are placed in fixed length records; empty fields are not written in variable length records.
2 – See the *Tymshare EDITOR Reference Manual* for complete documentation of control characters.

A back slash, followed by a carriage return, may be used repeatedly to reenter several preceding fields.

In the example below, the user types various editing characters. First, the user types Control A to delete an R. In the next record, he inadvertently types a period instead of a comma after the second field. He types a carriage return, not realizing the data is entered incorrectly. When the system prints an error message, the user wishes to delete the entire record. He must use an up arrow in this situation to delete beyond the carriage return. In record 3, the user types a back slash to reenter the data for the DESCRIPTION field.

```
-CREATE INV2:INVDES FROM T↩
INV2.. NEW FILE↩

OK.


DATA MUST BE ENTERED IN THE FOLLOWING ORDER:
PARTNO N 7
DESCRIPTION C 15
COST N 6 2
INVOICENO N 6
QUANTITY N 6


1: 1120557,FOR←CUS TUBE,9.60,414344,32↩
2: 1120579,DIAL ELEV.3.29,411022,15↩
TOO MANY CHARS. FOR ... COST
COST: ↑↩
2: 1120579,DIAL ELEV,3.29,411022,15↩
3: 1120545,AIR VALVE,\↩
DESCRIPTION: AIR VALVE NUT,.45,408999,10↩
4:
```

The user reenters the data for DESCRIPTION and continues data entry in the usual way.

Entering the Data from a File

The user may enter the data for CREATE from a file. The input file must contain on one logical line the same number of fields for each record in the order specified in the description file.[1] The user must terminate each field with a comma.[2] He may type two consecutive commas (field terminators) in the middle of a record to indicate an empty field, three commas to indicate two consecutive empty fields, and so on. A comma at the end of a record indicates that the last field is empty. The user may continue a record with a line feed and must terminate a record with a carriage return.

NOTE: The input file must be symbolic.

If a record contains an error, such as alphabetic characters in a numeric field, CREATE prints the number of the record in error, ignores that record, and proceeds to the next record.

The example following illustrates entering the data from a file named INV3DATA to create a file named INV3, according to the description on file INVDES.

1 – A logical line is a series of characters terminated by a carriage return. A logical line may contain line feeds; thus, it may occupy more than one physical line.
2 – See the discussion on page 64 of the field terminator option to change the comma as field terminator.

```
-EDITOR                          The user creates the input file in EDITOR.
*APPEND
1121202,CASTING,3.60,393912,24
1120556,SCREW,.01,400003,1500
1120574,FOCUS RING,2.74,396666,12
1120579,DIAL ELEV,3.29,410111,15
*WRITE                           The user should write all IML input files with
TO: INV3DATA                     a line feed to preserve multiple blanks.
 NEW FILE
128 CHARACTERS
*QUIT

-
```

The user types the CREATE command to create the data file.

```
-CREATE INV3:INVDES FROM INV3DATA
INV3.. NEW FILE

OK.



4 RECORDS CREATED.


-
```

This listing shows the records written by CREATE. Each line represents one record.

```
 PARTNO      DESCRIPTION      COST   INVOICENO  QUANTITY

1121202CASTING             3.60393912     24
1120556SCREW                .01400003   1500
1120574FOCUS RING          2.74396666     12
1120579DIAL ELEV           3.29410111     15
```

Optional Components

CREATE provides several optional components which allow the user to request prompts, change the field terminator character, enter only specific fields for each record, and append records to an existing data file. These features are described on the following pages.

Requesting Prompts

When entering data at the terminal, the user may instruct the system to prompt him for each field in the record. He requests this option by typing [PROMPT] after CREATE, as shown below.

The general form of the command using this option is:

```
                    data   description
-CREATE [PROMPT]    file : file          FROM T
                    name   name
```

For example:

```
-CREATE [PROMPT] INV4:INVDES FROM T
INV4.. NEW FILE

OK.



1:
PARTNO: 1120577
DESCRIPTION: TRUNNION
COST: 3.12
INVOICENO: 405321
QUANTITY: 12
2:
PARTNO: 1120578               The user types Control A to delete the U.
DESCRIPTION: STUD LU←CK       He forgets to add an O in its place.
COST: \                       Realizing the error at this point, the user
DESCRIPTION: STUD LOCK        types a \ (Shift L) to reenter the last field.
COST: 1.74
INVOICENO: 400396
QUANTITY: 100
3:
PARTNO:                       A carriage return after the first prompt
2 RECORDS CREATED.            in the record terminates data entry.

-
```

NOTE: The PROMPT option does not allow the user to specify more than one field per line.

Changing the Field Terminator

The field terminator option allows the user to change the field terminator character from a comma to a user-specified character. This option is most helpful when the data contains commas in the fields. To avoid typing a Control V before each comma in the data, the user may merely change the field terminator character from a comma to another character. The general form of the command is:

```
                      data    description
-CREATE [character]   file  : file         FROM  { T               } ↵
                      name    name               { input file name }
```

The example below illustrates the use of this option to define an ampersand (&) as the field terminator.

```
-CREATE [&] INV5:INVDES FROM T↵
INV5.. NEW FILE


OK.

DATA MUST BE ENTERED IN THE FOLLOWING ORDER:
PARTNO N 7
DESCRIPTION C 15
COST N 6 2
INVOICENO N 6
QUANTITY N 6



1: 1120580&VERN,IER&1.67&399975&8↵
2: 1120586&AZI,MUTH↵
COST: 10.99&403902&32↵
3: 1121203&INS,ULATOR&&400007&28↵
4: 1121202&CAS,TING&3.60&402336&12↵
5: ↵
4 RECORDS CREATED.

-
```

Entering Specific Fields

CREATE provides three optional clauses to enter only specific fields for each record; the system fills the missing fields with blanks or zeros. The BY NAME clause permits the user to enter different fields for each record. The BY LIST clause performs the same function as the BY NAME clause: It lets the user enter data for different fields in each record, preceding the data for each field by the field name and a colon (:). The only difference between BY LIST and BY NAME is that BY LIST does not ignore an entire record when an invalid field name or illegal data are specified; instead, the invalid fields are ignored, and the data for the other fields are written in the record. The WITH clause allows the user to enter the same specific fields for each record. Each optional clause is discussed below.

NOTE: When entering only specific fields for variable length records, the record created is shorter than the full record length if data for the last field is not entered. This must be avoided when the data file is to be used with fixed format programs such as TYMTAB and RETRIEVE.[1]

The general form of the command using the BY NAME, BY LIST, or WITH clause is:

```
                   data    description                                { BY NAME         }
-CREATE [option]   file  : file         FROM  { T               }     { BY LIST         } ↵
                   name    name               { input file name }     { WITH field list }
```

1 – Refer to the *Tymshare TYMCOM-IX TYMTAB and RETRIEVE Reference Manuals* for complete documentation of TYMTAB and RETRIEVE, respectively.

The WITH clause allows the user to enter only the specified fields for each record. The order of the fields in the field list need not correspond to the order in the description file, but must correspond to the order of entry. The records written by CREATE contain the fields in the order specified by the description file. In the following example, the user wants to enter data for the PARTNO, INVOICENO, and QUANTITY fields and have the system fill the DESCRIPTION and COST fields with spaces or zeros.

```
-CREATE INV6:INVDES FROM T WITH QUANTITY,PARTNO,INVOICENO↵
INV6.. NEW FILE↵

OK.


DATA MUST BE ENTERED IN THE FOLLOWING ORDER:
QUANTITY N 6
PARTNO N 7
INVOICENO N 6


1: 12,1120575,410033↵
2: 32,1121202,397422↵
3: ↵
2 RECORDS CREATED.


-
```

The user must enter the data in the order specified in the field list.

The listing below shows the records written by CREATE. Each line represents one record. Note that unspecified fields are empty.

```
PARTNO     DESCRIPTION      COST   INVOICENO   QUANTITY

1120575                             410033       12
1121202                             397422       32
```

The user may include the PROMPT option in the command only when entering data at the terminal. The system prompts him for each field in the WITH clause. The field terminator option may be included in the command regardless of the data entry method.

The example below shows the use of the field terminator option and the WITH clause with data entry from an input file named INV7DATA.

```
-TYPE INV7DATA↵

1120552&404141
1121203&391232
1120888&410096
1120557&410455
1120578&410025

-
```

Each line contains PARTNO and INVOICENO data, separated by an ampersand (&).

The user calls CREATE, specifying the field terminator character, the file to be created and its description file, the input file, and the name of the fields to contain data.

```
-CREATE [&] INV7:INVDES FROM INV7DATA WITH PARTNO,INVOICENO↩
INV7.. NEW FILE↩

OK.



5 RECORDS CREATED.


-
```

NOTE: For each record to be created, the input file must contain on one logical line the data for all fields in the order specified in the field list. This order need not correspond to the order of the fields as specified in the description file.

The BY NAME clause permits the user to enter data for different fields in each record. Regardless of the data entry method, the data for each field is preceded by the field name and a colon. The fields for each record need not be arranged in the same order as specified in the description file.

NOTE: The PROMPT option may not be used with the BY NAME clause.

When entering the data from the terminal or input file, each field may be terminated with a carriage return or a comma; two consecutive carriage returns terminate the record; three consecutive carriage returns terminate the data entry procedure. For example:

```
-CREATE INV8:INVDES FROM T BY NAME↩
INV8.. NEW FILE↩

OK.



1:
PARTNO:1120552↩     The user separates each field with a carriage return.
COST:1.53↩
INVOICENO:404446↩
↩                   Two consecutive carriage returns terminate a record.
2:
INVOICENO:407713↩
↩
3:
DESCRIPTION:YOKE,INVOICENO:410000,QUANTITY:1000↩     The user separates
↩                   A second carriage return terminates the record.     each field with a
4:                                                                      comma.
↩
3 RECORDS CREATED.  A carriage return at the beginning of a record terminates data entry.


-
```

The listing below shows the records written by CREATE. Note that each field appears in the proper position in the record.

```
 PARTNO      DESCRIPTION        COST  INVOICENO  QUANTITY

1120552                        1.53404446
                                   407713
         YOKE                      410000   1000
```

The following example shows the use of the BY NAME clause when entering the data from an input file.

```
-EDITOR↵                     The user creates the input file in EDITOR.
*APPEND↵
DESCRIPTION:GASKET↵          ┐
COST:.39↵                    ├ Record 1
INVOICENO:411321↵            ┘
↵                            The second consecutive carriage return indicates the end of a record.
PARTNO:1120557↵              ┐
DESCRIPTION:FOCUS TUBE↵      ├ Record 2
INVOICENO:391177↵            │
QUANTITY: 12↵                ┘
↵
INVOICENO:410990↵            ]- Record 3
↵
INVOICENO:410991↵            ┐
QUANTITY: 100↵               ┘- Record 4
↵
DESCRIPTION:CASTING↵         ┐
INVOICENO:397891↵            ┘- Record 5
↵                            End-of-record carriage return.
↵                            The third consecutive carriage return indicates the end of the input file.
*WRITE↴                      The user writes the IML file with a line feed.
TO: INV9DATA↵
 NEW FILE↵
204 CHARACTERS
*QUIT↵

-CREATE INV9:INVDES FROM INV9DATA BY NAME↵
INV9.. NEW FILE↵

OK.


5 RECORDS CREATED.

-
```

NOTE: The user should remember the record terminator, that is, two carriage returns. If he fails to indicate the end of a record, the data is entered incorrectly.

Duplicating Fields

A capability in the CREATE data entry procedure permits the user to copy the contents of a field from the previous record to the same field in the current record. This feature is available with data entry from the terminal or an input file. The double quotation marks (") is the duplication character and must be the only character specified for a given field when the feature is used.

The following example illustrates the field duplication character used with the CREATE command.

```
-CREATE INV5:INVDES FROM T↵
INV5.. NEW FILE↵

OK.


DATA MUST BE ENTERED IN THE FOLLOWING ORDER:
PARTNO N 7
DESCRIPTION C 15
COST N 6 2
INVOICENO N 6
QUANTITY N 6


1: 1120557,FOCUS TUBE,9.60,407712,32↵
2: ",",",406678,50↵
3: 1120545,AIR VALVE NUT,.45,404331,"↵
4: 1120550,DIAL ELEV,3.42,405531,"↵
5: ↵

4 RECORDS CREATED

-
```

The user duplicates the first three fields used in the previous record.

By using the duplication character, he copies the last field from the previous record to two consecutive records.

The records created are listed below using the SELECT command.

```
-SELECT FROM INV5:INVDES TO T↵

OK.


1120557FOCUS TUBE       9.60407712    32
1120557FOCUS TUBE       9.60406678    50
1120545AIR VALVE NUT     .45404331    50
1120550DIAL ELEV        3.42405531    50

4 RECORDS SELECTED FROM 4

-
```

Appending Records to an Existing Data File

When the user wants to add records to an existing IML or RETRIEVE data file, he includes an APPENDING clause in the command. The general form of the command is:

```
                                     data    description
-CREATE [option] APPENDING TO file    :  file
                                     name    name
_______________________________________________________

                      (BY NAME        )
FROM  (       T      ) (BY LIST         )
      (input file name) (WITH field list )
_________________________________________
```

Note that the first file name specifies the existing data file to which the user wants to append records from the terminal or input file. The input file contains the records in free format, as illustrated in all the examples for entering data from a file.

The APPENDING clause may appear in a CREATE command with one option and one optional clause. For example, the user wants to append the records from INV9DATA, shown on page 68, to the file INV1, created on page 60. The resulting INV1 file after this command contains the records from the original INV1 and the new records appended from INV9DATA.

```
-CREATE APPENDING TO INV1:INVDES FROM INV9DATA BY NAME
INV1.. OLD FILE

OK.



5 RECORDS CREATED.


-
```

Note that the user includes the BY NAME clause because the INV9DATA input file contains data for only specific fields in each record and specifies the field name before each data item. The listing below shows the records in INV1 after the appending procedure.

```
 PARTNO     DESCRIPTION        COST  INVOICENO  QUANTITY

1120557FOCUS TUBE          9.60407712       32   ┐
1120572GASKET               .39406678       50   ├  Original INV1 records
1120545AIR VALVE NUT        .45404331       12   │
1120576YOKE               30.55399941        8   ┘
       GASKET               .39411321            ┐
1120557FOCUS TUBE             391177        12   │
                              410990             ├  Records appended
                              410991       100   │  from INV9DATA
       CASTING                397891             ┘
```

The following example illustrates the APPENDING clause with data entry at the terminal, using the PROMPT option and a WITH clause.

```
-CREATE [PROMPT] APPENDING TO INV5:INVDES FROM T WITH DESCRIPTION,COST↵
INV5.. OLD FILE↵

OK.



1:
DESCRIPTION: CASTING↵
COST: 3.60↵
2:
DESCRIPTION: GASKET↵
COST: .39↵
3:
DESCRIPTION: ↵
2 RECORDS CREATED.          INV5 now contains the original INV5 records
                            plus the two records created at the terminal.

-
```

Creating an ERRORS File

The CREATE command provides an additional clause, the ERRORS clause, which causes incorrect records to be written on a separate file. The ERRORS clause is permitted *only* when data entry is from an input file. The ERRORS clause always creates an error file when it is used with CREATE. If there are no errors, CREATE deletes the errors file at the end of the command procedure.

The complete form of the CREATE command for using the ERRORS clause is:

```
                       data   description       input ⎧BY NAME        ⎫            errors
–CREATE [character]    file : file         FROM file  ⎨BY LIST        ⎬ ERRORS TO  file ↵
                       name   name              name  ⎩WITH field list⎭            name
```

Example

The user wants to create an IML data file with data entry from an input file. The data file is to be written according to the SDESC description file shown below.

```
-TYPE SDESC↵

$$$
***BIN OR SYM:
BIN
```

```
***FIX OR VAR:
VAR
***  NAME, TYPE, START, LENGTH, DECIMAL PLACES
ITEM.NO,I,4,4
DESCRIPTION,C,,21
IN.STOCK,I,,5
YTD.SOLD,I,,6


-
```

Note that IN.STOCK is a numeric field.

The input file, STOCKINPUT, contains two errors.

```
-TYPE STOCKINPUT↵

42,DISTRIBUTOR CAP,47Y,28
28,SOLENOID SWITCH,500,209
19,SPARK PLUG,192W,276
11,RADIATOR,28,4
77,UNIVERSAL JOINT,212,44

-
```

In these records, the data for the third field is incorrect because IN.STOCK is a numeric field.

The user enters the CREATE command and requests that an errors file be created.

```
-CREATE STOCKFILE:SDESC FROM STOCKINPUT ERRORS TO ERRSTK↵
STOCKFILE.. NEW FILE↵
ERRORS TO ERRSTK.. NEW FILE↵

OK.


VALUE NOT NUMERIC IN.. IN.STOCK OF RECORD 1
VALUE NOT NUMERIC IN.. IN.STOCK OF RECORD 3
3 RECORDS CREATED
2 "ERRORS"" RECORDS WRITTEN


-
```

The errors file, ERRSTK, is shown below.

```
-TYPE ERRSTK

42,DISTRIBUTOR CAP,47Y,28
19,SPARK PLUG,192W,276


-
```

VERIFY

VERIFY offers the user a convenient tool for checking the accuracy and completeness of a data file. Using basic IML rules statements and commands, the user may request data verification procedures as simple or as complex as needed.

The complete general form is:

```
        data   description  { IF conditions  }     output      key             rules
-VERIFY file : file         { FOR conditions } TO  file    BY  field AS PER    file
        name   name                                name        list            name
```

Any information file, including a RETRIEVE file, may be specified as the data file in the command. The description file may be a file created with DEFINE, or it may be a RETRIEVE structure file. Only the data file, description file, and rules file names are required. The user may, however, omit the description file name if the data file is a RETRIEVE file and if a corresponding structure file exists in the user's directory. The user specifies the rules file name as T (for terminal) to enter the rules file statements initially.

It is suggested that users unfamiliar with IML rules files read pages 27 through 41 before proceeding with the discussion of VERIFY.

The functions of the optional IF clause, output file name, and BY clause are presented on page 76, following the sample problem. The structure of a VERIFY rules file is discussed on page 77.

Sample Problem

The sample problem checks the data in a file named PERSONNELDATA which contains records for all employees in a small company. After entering all the data, the user calls VERIFY to check the accuracy of his data.

The key to the verification procedure is the rules file PERSONNELVERULES which details all conditions to be checked and, optionally, corrected at the terminal.

The user begins the verification process with a single command specifying the file to be verified and its description file, an optional output file for the corrected data file, and the name of the VERIFY rules file.

```
-VERIFY PERSONNELDATA:P TO VPERSONNEL AS PER PERSONNELVERULES↵
: LIST↵              The actual rules file is shown first and then executed with the RUN command.
10  DECLARE
20    MALES,I,3
30    FEMALES,I,3  The user defines three temporary fields for the duration of the verification procedure.
35    TOTAL,I,3
40  INITIAL            This message is printed before any records are verified.
45    TYPE "BE PREPARED TO REENTER DATA AT THE TERMINAL",CR,
           "DURING THE VERIFICATION PROCEDURE.",CR,CR
50  DETAILS         Each record is processed by the DETAILS section of the rules file and corrected if in error.
60    IF AGE<16 OR >70
65      TYPE "ENTER CORRECT AGE FOR ",NAME, " EMP#",EMP.NO
70      INPUT AGE
72      TYPE
75    IF SEX NOT HAS "F" AND "M"
80      TYPE "ENTER M OR F FOR SEX OF ",NAME,"  EMP#",EMP.NO
84      INPUT SEX
85      TYPE
86    IF SEX='F'
87      FEMALES +1 TO FEMALES
88    ELSE MALES +1 TO MALES
90    IF MAR.STAT NOT HAS "S" AND "M" AND "D" AND "W"
95      TYPE "ENTER CORRECT MARITAL STATUS FOR ",NAME," EMP#",EMP.NO
100     INPUT MAR.STAT
105     TYPE
110   IF SALARY >25.
120     IF.1 SALARY <4000 OR >50000
130       TYPE "PLEASE REENTER THE ANNUAL SALARY FOR ",NAME," EMP#",
           EMP.NO
135       INPUT SALARY
136       TYPE
140   ORIF SALARY <1.75
145     TYPE "REENTER HOURLY RATE FOR ",NAME," EMP#",EMP.NO
150     INPUT SALARY
151     TYPE
155   IF (RATING<1 OR > 5) AND LAST.REV>0
160     TYPE "PLEASE REENTER THE RATING FOR ",NAME," EMP#",EMP.NO,
         CR," FOR REVIEW AS OF ",LAST.REV
163     INPUT RATING
164     TYPE
165   IF PROMOTABILITY # 0 AND 1
170     TYPE "REENTER PROMOTABILITY CODE FOR ",NAME," EMP#",EMP.NO,CR,
         " FOR REVIEW AS OF ",LAST.REV
175     INPUT PROMOTABILITY
176     TYPE
200   IF (YR<70 OR >72 OR M<1 OR>12 OR D<1 OR >31) AND LAST.REV >0
210     TYPE "PLEASE REENTER DATE OF LAST REVIEW FOR ",NAME," EMP#",
         EMP.NO
220     INPUT LAST.REV
221     TYPE
230   IF (M=4 OR 6 OR 9 OR 11) AND D>30
240     TYPE "PLEASE REENTER DATE OF LAST REVIEW FOR ",NAME," EMP#",
         EMP.NO,CR," DATE OF **",LAST.REV," IS NOT POSSIBLE"
250     INPUT LAST.REV
```

```
251       TYPE
260     IF M=2
270       IF.1 D>29
280         TYPE "PLEASE REENTER DATE OF LAST REVIEW FOR ",NAME," EMP#",
             EMP.NO,CR,"DATE OF **",LAST.REV, " IS NOT POSSIBLE"
285         INPUT LAST.REV
286         TYPE
290       ORIF.1 D=29
300         IF.2 YR NOT = 72 AND 76
310           TYPE "PLEASE REENTER DATE OF LAST REVIEW FOR ",NAME,
               " EMP#",EMP.NO,CR,"FEBRUARY HAS ONLY 28 DAYS IN 19",
               YR,CR,"DATE OF **",LAST.REV," IS NOT POSSIBLE"
315           INPUT LAST.REV
316           TYPE
400 FINAL
410     TYPE CR,MALES," MALES EMPLOYED",2B,FEMALES," FEMALES EMPLOYED"
415     FEMALES + MALES TO TOTAL
420     TYPE TOTAL, " TOTAL EMPLOYEES"
: RUN↵
```

The RUN command begins execution of the verification process.

```
VPERSONNEL.. NEW FILE↵

OK.


BE PREPARED TO REENTER DATA AT THE TERMINAL
DURING THE VERIFICATION PROCEDURE.


ENTER M OR F FOR SEX OF CLOUTIER   EMP#   9
F↵

ENTER CORRECT AGE FOR FLINKER   EMP#  58
28↵
```

The questions and answers show the location and correction of the data in PERSONNELDATA according to the PERSONNEL VERULES file.

```
ENTER CORRECT MARITAL STATUS FOR KINNEY    EMP#  62
M↵

PLEASE REENTER THE ANNUAL SALARY FOR VOGELSANG EMP#  65
12000.00↵


PLEASE REENTER DATE OF LAST REVIEW FOR PENTEL    EMP#  68
 DATE OF ** 710931 IS NOT POSSIBLE
710905↵

PLEASE REENTER THE RATING FOR LEMBERG   EMP#  71
 FOR REVIEW AS OF  710215
3↵

PLEASE REENTER THE RATING FOR BORNEMAN  EMP#  72
 FOR REVIEW AS OF  710515
3↵

REENTER PROMOTABILITY CODE FOR BORNEMAN  EMP#  72
 FOR REVIEW AS OF  710515
0↵
```

```
PLEASE REENTER DATE OF LAST REVIEW FOR CLIFF       EMP#  83
FEBRUARY HAS ONLY 28 DAYS IN 1971
DATE OF ** 710229 IS NOT POSSIBLE
710212↩

REENTER HOURLY RATE FOR YORK       EMP#  85
5.00↩

ENTER CORRECT MARITAL STATUS FOR HANSON     EMP#  87
M↩


 39 MALES EMPLOYED   16 FEMALES EMPLOYED
 55 TOTAL EMPLOYEES


55 RECORDS VERIFIED OF 55

-
```

The FINAL section of the VERIFY rules file requests that this information be printed after verifying all records.

Optional Components

VERIFY contains several optional features which allow the user to qualify records for the verification procedure, generate an output file, and perform a sequence check on the records. Each component is discussed separately below.

The user should request an output file for the corrected data file when he intends to reenter field information at the time an error is discovered.[1] For example,

–VERIFY PERSONNELDATA:P TO VPERSONNEL AS PER VRULES↩

specifies an output file named VPERSONNEL on which VERIFY writes the verified and corrected data.

An IF clause is available to qualify each record before performing the verification procedure. The clause immediately follows the data file specification in the command. For example,

–VERIFY PERSONNELDATA:P IF REV.DATE>720531 AS PER VRULES↩

causes only records which specify a review date after May 31, 1972, to be verified. The user may specify numerous conditions in one IF clause.[2]

The user may request a sequence check on the arrangement of the records at the same time he executes the verification procedure. The BY clause specifies a key field or fields in the records to be verified; these key fields must contain data in numerical or alphabetical order. For example,

–VERIFY PERSONNELDATA:P BY EMP.NO AS PER VRULES↩

specifies that records in PERSONNELDATA must be arranged in numerical order by employee number. The BY clause follows the output file specification, if any, and may contain as many as 20 key fields.

The following command illustrates all the optional features of the VERIFY command and shows the position of the optional components in the command.

1 – The INPUT statement, discussed on page 34, allows the user to enter new data for a field directly at the terminal.
2 – See page 20 for details on conditional expressions.

```
-VERIFY PERSONNELDATA:P FOR REV.DATE>720531 AND DEPT NOT=5 ↴
TO VPERSONNEL BY DEPT,NAME AS PER VRULES↵
```

The command instructs VERIFY to check all records in the PERSONNELDATA file described by P which contains a review date after May 31, 1972, and which also do not specify the department as 5. The VRULES file contains the statements which direct the verification procedure and allow the user to correct any errors discovered. At the same time, a sequence check is performed on the records being verified; they must be in numerical order by department and within the same department in alphabetical order by name. Finally, the user includes the name of an output file, VPERSONNEL, on which VERIFY writes all verified records, including any changes made by the user during the verification process.

Structure of a VERIFY Rules File

The statements in a VERIFY rules file are divided into four separate sections.[1] The rules file may contain all or some of the sections; however, the sections must appear in the file in the following order:

DECLARE
INITIAL
DETAILS
FINAL

The DECLARE section makes it possible to specify new fields to save data from records or intermediate calculations for the duration of the verification process. The INITIAL section assigns starting values to declared fields, if necessary, and may also be used to print introductory messages. The DETAILS section specifies the actual verification procedure to be performed for each record. The FINAL section allows the user to print any accumulated data or messages after all records are processed.

NOTE: When no section name appears at the start of the rules file, VERIFY assumes it is the DETAILS section.

The discussion which follows treats each rules section separately, specifying available capabilities, legal statements, and examples.

The DECLARE Section

The DECLARE section allows creation of working storage. It specifies new fields which do not exist in the input records, but which can be used to save data from records or calculations. For example,

```
10  DECLARE
20     MALES,I,3
30     FEMALES,I,3
35     TOTAL,I,3
```

creates temporary storage for three fields which do not appear in the input records. The user may declare as many fields as desired, the only limitation being that a maximum of 100 lines is permitted in a rules file.

1 – Rules file statements are presented on page 32.

The form used to specify new fields is:

field name,data type,field length,decimal places

Note that only one field declaration may appear on a line.

The field name may contain as many as 31 characters; must begin with a letter; and may be any combination of letters, the digits 0 through 9, the period (.), and the character @. See page 47 for a list of reserved field names.

The data type may be C, N, I, R, or D. The data type indicates the contents of the field, as follows:

Data Type	Meaning
C	Character data
N	Numeric data
I	Integer number
R	Real number
D	Double precision number

NOTE: With declared fields, the data types I, R, and D do not indicate binary data, but rather specify the most efficient storage for an integer, real, or double precision number, respectively.

The field length specifies the maximum number of characters needed to write the data in symbolic form.

The decimal places specification is necessary only for those fields that are to contain numbers with decimal points; it specifies the maximum number of digits to the right of the decimal point.

NOTE: Declared numeric fields are automatically initialized to zero; declared character fields are initialized to blanks.

Refer to page 141 for a description of the ROUNDIN and ROUNDOUT statements which may be used to round numeric data and/or arithmetic results to conform to the field descriptions.

The INITIAL Section

The INITIAL section is executed at the beginning of the verification procedure, before any records are processed. Data fields, therefore, may not appear in this section. The user may assign starting values for declared fields in this section. For example,

```
40 INITIAL
50 100 TO BASE
```

assigns an initial value of 100 to a declared field named BASE. It is also possible to assign initial values to declared fields directly at the terminal, using an INPUT statement.[1] For example,

```
50 INITIAL
60 TYPE "PLEASE ENTER THE VERIFY CODE"
70 INPUT VCODE
```

1 – The INPUT statement is detailed on page 34.

assigns the value entered at the terminal to VCODE. Note that VCODE must be defined in the DECLARE section.

The DETAILS Section

The DETAILS section is executed for each record in the file except for those which are ignored because of an IF clause in the original VERIFY command.[1] In this section, the user specifies the details of the verification procedure. With control statements and instructions, the user states the fields to be verified, messages to be printed at the terminal, and corrections to be made, if desired. For example:

```
50  DETAILS
60    IF AGE<16 OR >70
65      TYPE "ENTER CORRECT AGE FOR ",NAME, " EMP#",EMP.NO
70      INPUT AGE
72      TYPE
75    IF SEX NOT HAS "F" AND "M"
80      TYPE "ENTER M OR F FOR SEX OF ",NAME,"  EMP#",EMP.NO
84      INPUT SEX
85      TYPE
```

Lines 60 through 72 handle the verification and correction, as needed, of the AGE field. The user specifies that if AGE is less than 16 or greater than 70, he wishes to correct the data in AGE. If, for example, a record contains 12 in the AGE field, lines 65 through 72 are executed. The system prints the information specified in the TYPE statement, accepts a value for AGE from the terminal, prints a blank line, and proceeds to evaluate the next IF statement, which handles the verification of the SEX field in the same manner.

In addition, the user may access information from a record to be used in a later calculation by saving it in a temporary storage field created in the DECLARE section. For example:

```
86    IF SEX='F'
87      FEMALES +1 TO FEMALES
88    ELSE MALES +1 TO MALES
```

Lines 86 through 88 cause a continuous calculation of the number of male and female employees. The counts are stored in temporary fields, MALES and FEMALES, created in the DECLARE section.

The FINAL Section

After all records are processed, the FINAL section is executed. It is here that the user specifies any final calculations to be performed, as well as any accumulated calculations and comments to be printed. For example:

1 – The optional IF clause is described on page 76.

```
400 FINAL
410   TYPE CR,MALES," MALES EMPLOYED",2B,FEMALES," FEMALES EMPLOYED"
415   FEMALES + MALES TO TOTAL
420   TYPE TOTAL, " TOTAL EMPLOYEES"
```

Line 410 prints the number of males and females employed; line 415 calculates a final total of employees; and line 420 prints the final total.

SECTION 5
FILE MANAGEMENT

The Information Management Library offers a wide range of programs to handle all aspects of file manipulation. SORT performs a sort on the contents of a file. MERGE combines two sorted files to create a single sorted file. The user may update the information in a file using SELECT, PURGE, REPLACE, or UPDATE. SELECT retrieves specified field information or entire records; its complementary command, PURGE, deletes specified records in a file. REPLACE allows the user to substitute field information or entire records in a file. Finally, UPDATE provides a thorough procedure to test for specified conditions, perform arithmetic operations, print remarks or information at the terminal during the process, and replace information in a data file.

This section discusses each program separately. The sorting commands, SORT and MERGE, are presented first, since they are basic to all data management procedures. When using two data files, the user must order the records in both files before entering any IML command. The updating commands, SELECT, PURGE, REPLACE, and UPDATE, are presented last. It is suggested that the user refer to REPLACE for simple changes to a data file, and to UPDATE for changes requiring arithmetic operations or multilevel conditional testing.

EXAMPLE DATA FILE AND DESCRIPTION

Because this section uses the same personnel application for most examples, the description file, PDESC, and the PERSONNEL data file are printed below to inform the reader of the names, locations, and contents of the records on which the sample commands operate.

```
-TYPE PDESC↵

$$$
***BIN OR SYM:
S
***FIX OR VAR:
V
***WHICH TYPE:
L
***NUMBER OF LINES PER RECORD IS:
1
```

```
***  NAME, TYPE, START, LENGTH, DECIMAL PLACES
NAME,C,1,9
EMP.NO,N,.,4
AGE,N,.,3
SEX,C,.,2
MAR.STAT,C,.,1
CHILDREN,N,.,2
LAST.DEG,N,.,2
AREA,C,.,5
AREA2,C,.,4
DEPT,N,.,3
JOB,N,.,3
SALARY,N,.,10,2
LAST.REV,N,.,7
RATING,N,.,2
PROMOTABILITY,N,.,2
PCT.RAISE,N,.,5,1
DATE.HIRE,N,.,7


-
```

The PERSONNEL file contains these records:

```
-TYPE PERSONNEL↵
```

Name, *Emp. No.*, *Age*, *Sex*, *Marital status*, *Children*, *Last degree*, *Area*, *Second area*, *Department*, *Job*, *Salary*, *Last review date*, *Rating*, *Promotability*, *% raise*, *Date of hire*

```
ABNER      92 38M M 0 4ELEC PHYS  5  3  12000.00      0 0 0   .0 720415
AVERY      36 37M M 0 4MBA  HIST  2  2  16800.00 720217 2 1  8.5 640212
BORNEMAN   72 29M S 0 5PHYS MATH  3  7  12800.00 710515 3 0  5.5 700515
BOSUNG     73 39M D 3 4PHYS MATH  5  7  13000.00 720101 3 0  5.5 700615
CHEVES     49 39M D 3 4ENGR MATH  1  5  17000.00 720306 2 0 10.0 650306
CLIFF      83 31M M 4 4LAW  ENGR  3  2  14000.00 710212 4 0  4.0 701217
CLOUTIER    9 60F D 2 1ACAD ART   3  8      5.25 720415 1 0  8.3 600823
CUMMINGS   80 29M M 2 5PHYS MATH  2  1  20000.00 711115 1 1 15.0 701112
DUNCAN     64 29M S 0 3MATH       5  6  13500.00 711215 2 0  8.0 690630
DURBAN     52 27M S 0 4CHEM MATH  4  6  16500.00 720312 2 0  6.5 660310
ELBERT     63 51M D 3 4MBA  ACCT  3  2  13000.00 710303 2 1  7.6 690429
FLINKER    58 28F M 0 4EDUC ECON  2  4  11000.00 711215 1 1 10.0 680715
FLOYD      77 27F S 0 4MBA  ACCT  2  2  15000.00 711018 1 1 14.0 701112
FULMER     46 29F S 0 5MATH MBA   4  1  22000.00 720118 1 1 15.0 650122
GOLDEN     70 29M D 2 5PHYS MATH  5  7  18000.00 720301 1 1 12.2 700301
GOOD       88 26F S 0 4CHEM ACCT  2  6  14675.00 711202 1 1  7.0 711119
HANSON     87 35F M 0 3EDUC ACAD  5  4  10000.00 720415 1 1  8.5 710415
HARTMAN     2 43F D 1 3ENGL STEN  1  4  15000.00 711012 1 0 10.0 600525
HUGHES     42 29M D 2 2ELEC ACAD  3  3      6.95 720517 1 0  8.5 640523
JOHNSON     4 62M M 4 4ENGR MATH  5  5  16800.00 720115 2 0 10.0 600525
KINNEY     62 28F M 1 3ENGL EDUC  1  4  10500.00 710415 1 1  8.0 690415
KNIGHT     44 40F D 3 4PHYS MATH  3  7  35000.00 711213 2 1  9.5 641210
```

```
LARSON     84 38F M 2 1ACAD TYP   1 8     3.75 710930 2 1  6.5 701230
LASKER     95 46M M 2 3SOC        3 4  9700.00      0 0 0   .0 720625
LEE        45 37M M 6 4MBA  PHYS  3 2 19500.00 710514 2 1  6.8 641218
LEMBERG    71 32M S 0 4MATH       3 6 11000.00 711015 3 1  6.3 700506
LEWIS      76 32M M 2 2ELEC COMM  3 3     6.75 711107 2 0  5.5 701105
LIGHT      50 31M M 3 3HIST POLI  2 6 10400.00 710209 3 1  5.0 650727
LINDEN     90 28F S 0 3ENGL EDUC  3 4  7500.00      0 0 0   .0 720315
MANDEL     91 42M M 2 2STEN MATH  1 8     5.00      0 0 0   .0 720315
MARIN      94 21M S 0 4MBA  POLI  4 2  9600.00      0 0 0   .0 720612
MARTIN      7 39M M 3 4ENGR MATH  1 1 25400.00 711213 1 1 10.0 600601
MONROE     32 31F M 0 3MATH ENGL  4 6 15000.00 720412 1 1 10.0 620627
MOREHEAD   86 39M S 0 4ENGR PHYS  5 5 13000.00 720315 2 1 10.0 710315
O'CONNORS  35 52M M 2 4ENGR BIOL  5 1 24000.00 710615 1 1 10.0 630515
OAKDALE    81 41M W 2 3GEOL MATH  3 6 10000.00 710609 3 1  8.5 701203
PARKER      1 48M M 4 4ENGR PHYS  1 1 38000.00 720101 1 1 10.0 600525
PENTEL     68 24M S 0 5PHYS MATH  3 1 29000.00 710915 1 1 10.2 700101
QUINLAN    55 29F S 0 4ENGR MATH  5 5 14500.00 711215 2 1  8.5 680515
RILEY      39 52M M 5 4ENGR PHYS  1 5 22000.00 720412 1 1 10.0 640408
RINSLER    75 48M M 4 4MATH PHIL  5 6 16800.00 710315 2 1  8.5 700915
ROTH       54 55M M 1 2PLMB ACAD  5 3     4.50 720430 2 0 10.0 680430
RUDMAN     93 20F S 0 3MATH PHYS  3 6  8500.00      0 0 0   .0 720506
SANDERS    60 45M M 3 2ELEC COMM  3 3     5.50 720215 3 0  5.0 690215
SHAMES     29 49M M 2 3EDUC       2 4 16000.00 720201 2 0  8.5 610901
SIMON      74 25M S 0 5CHEM BIOL  5 6 21000.00 710715 2 1 12.0 700715
STARK      53 30F M 1 3PHYS       2 7 13000.00 710502 3 1  5.5 671113
SUMNER     20 48M M 2 2ELEC COMM  3 3     8.50 710615 2 0  6.5 601001
VOGELSANG  65 28M M 0 4ENGL       2 4 12000.00 711108 1 0 10.0 691110
WAGNER     89 22F S 0 2ACAD       4 8     3.00      0 0 0   .0 720315
WARNER     34 42M D 1 3ENGL       3 4 14000.00 720209 2 0  8.5 630210
WEISS      56 30M M 2 4LAW  GOVT  1 2 18500.00 711215 2 1  8.5 680524
WEST       26 46M M 3 3MATH       1 6 23600.00 720316 1 1 12.5 610115
YORK       85 34M M 0 4ARCH ART   3 8     5.00 711101 2 0 10.0 710301
ZIMMERMAN  61 39M M 3 5CHEM MATH  2 6 19500.00 720315 2 0 10.0 690301

-
```

Note that the dates are written in the form

YYMMDD

where the year, month, and day are represented by two-digit integers. For example, 610115 specifies January 15, 1961, 690301 specifies March 1, 1969, and so on.

SORT

SORT orders the records in a file by ascending numerical or alphabetical order according to the information in the key fields. To order the records in descending numerical order or reverse alphabetical order, the user precedes the key field name with a minus sign (–).

The user calls SORT by typing the SORT command directly in the EXECUTIVE. SORT prompts the user for any missing information; after accepting the complete command, SORT prints

OK.

performs the requested rearrangement of all records, prints

SORT FINISHED.

and returns control to the EXECUTIVE.

The complete form of the SORT command is:

```
        unsorted   description        output      key
-SORT   file     : file           TO  file    BY  field
        name       name               name        list
```

For example:

```
-SORT PERSONNEL:PDESC TO STDPERSONNEL BY EMP.NO
STDPERSONNEL.. NEW FILE

OK.


SORT FINISHED.

-
```

SORT prompts for missing components when it reads an incomplete command. For example, a user wants to maintain his master personnel file in numerical order by employee number; the field in each record which contains employee number information is a key field. Thus, if a user wishes to sort the records in PERSONNEL by EMP.NO, he may type any of the following commands

```
-SORT
-SORT PERSONNEL
-SORT PERSONNEL:PDESC TO STDPERSONNEL
-SORT PERSONNEL TO STDPERSONNEL BY EMP.NO
```

and SORT prompts for missing information.

SORT permits the user to specify as many as 20 key fields for a single sorting procedure. SORT performs the sorting according to the order in which the field names appear in the key field list. The field mentioned first determines the overall arrangement of records. When several records contain identical data in the first key field, SORT arranges these records by data in the second key field; when more than one record contains identical information in the first two key fields, SORT arranges these records by data in the third key field; and so on. For example,

```
-SORT PERSONNEL:PDESC TO EXAMPLE BY DATE.HIRE,-PROMOTABILITY
```

instructs SORT to sort by DATE.HIRE primarily. Records in EXAMPLE are, therefore, arranged in DATE.HIRE order; when records have the same DATE.HIRE, they are arranged in descending numerical order by PROMOTABILITY for that group of equal DATE.HIRE records.

NOTE: Because of the sorting technique used, items which are identical in all key fields may not appear in the same order as they occurred in the unsorted file. If the original order is essential, RECNO should be used as the final item in the key field list.

Normally, the user specifies an output file on which to write the sorted data. For example, the previous SORT command instructs SORT to write the sorted records on the file EXAMPLE and not to change the information saved on PERSONNEL. This procedure is suggested for the initial sorting of a data file to ensure a backup file. An error in defining fields in the description file, or writing a data file in EDITOR with multiple blank compression, would instruct SORT to perform a meaningless rearrangement of the information in the file.

SORT allows the user to omit an output file, thereby instructing the system to write the results of the command on the original file. For example,

```
–SORT PERSONNEL:PDESC BY EMP.NO↵
```

instructs SORT to write the sorted PERSONNEL file on the original PERSONNEL file, thereby destroying the unsorted data file.

If the user enters an incomplete SORT command, SORT prompts for an output file name to encourage the user to preserve the original data file. A carriage return in response to the output file request, however, instructs SORT to write the sorted file on the data file named, destroying its original contents.

In the example below, the system prompts for each component in the SORT command.

```
-SORT↵
UNSORTED DATA FROM: PERSONNEL↵
DESCRIPTION FROM: PDESC↵
SORTED DATA TO: SORTEDPERSONNEL↵
SORTEDPERSONNEL.. NEW FILE↵
NAMES OF KEY FIELDS: EMP.NO↵

OK.


SORT FINISHED.

-
```

The user may type only a carriage return here if a RETRIEVE structure file corresponding to PERSONNEL exists in the user's directory.

Note that the system prints the output file name specified, followed by two periods and a NEW FILE or OLD FILE message. The user confirms the specified file name with a carriage return or enters a different output file name after typing an alt mode/escape. For example, the user specifies a different key field, DATE.HIRE, for the sort. The output file, PERSONNELSENIORITY, contains all the records from PERSONNEL, arranged in chronological order by date of hire. The file SORTEDPERSONNEL contains the same records, arranged in numerical order by employee number.

```
-SORT PERSONNEL:PDESC TO SORTEDPERSONNEL↵
SORTEDPERSONNEL.. OLD FILE⊕
SORTED DATA TO: PERSONNELSENIORITY↵
PERSONNELSENIORITY.. NEW FILE↵
NAMES OF KEY FIELDS: DATE.HIRE↵

OK.


SORT FINISHED.

-
```

The user mistakenly enters SORTEDPERSONNEL for the output file name and types an alt mode/escape to enter a different output file name.

The following examples summarize the various forms and capabilities of the SORT program.

–SORT ABC TO DEF BY FLD9↵

ABC is a RETRIEVE data base; SORT automatically looks for a corresponding RETRIEVE structure file to describe the data in ABC. The result of the sort is written on DEF; ABC remains unchanged. The sorted file must be specified as DEF:ABC'STR.E' in subsequent commands.

–SORT ABC BY FLD1↵

The sorted data is written on the file named ABC and can be described automatically in subsequent commands by the corresponding RETRIEVE structure file.

–SORT X:DESCX TO Y BY A,B↵

More than one key field is used in the sorting task. The file is sorted by A; when A is the same for several records, these records are ordered by the data in B.

–SORT ACCOUNTS TO OVERDUE BY –BALANCE↵

The file OVERDUE contains the records arranged in descending order by BALANCE; the record with the largest balance appears first.

MERGE

The MERGE program combines two sorted files to a single file sorted in the same order. The form of the MERGE command is:

```
        sorted   description                             sorted
–MERGE  file   : file         {IF conditions }   WITH  file     {IF conditions }
        name1    name         {FOR conditions}         name2    {FOR conditions}
_________________________________________________________________________________

    merged      key
TO  file    BY  field ↵
    name        list
_______________________
```

Note that the two sorted input files must be described by the same description file whose name follows the first sorted file name, and both sorted input files must contain records sorted on the key fields specified in the command. With one exception, the items in the key field list are treated identically in MERGE and SORT: If all items in the key field list are identical for two records, one in each file, the record in sorted file$_1$ is written first on the merged file.

The following example illustrates the use of the MERGE command.

```
-MERGE↵
MERGE FILE 1: INDEX1↵
FILE DESCRIPTION FROM: DX↵
CRITERIA FOR FILE 1: SUBJECT STARTS WITH "T"↵
MERGE FILE 2: INDEX2↵
CRITERIA FOR FILE 2: SUBJECT STARTS WITH "T"↵
```

If the user does not want to enter a conditional clause, he may respond with a carriage return.

```
OUTPUT TO: INDEXT↵
INDEXT.. NEW FILE↵
NAMES OF KEY FIELDS: SUBJECT↵

OK.

64 RECORDS MERGED OF 1810

-
```

As with all other IML commands, MERGE does *not* prompt for the optional conditional clauses when another command component appears after the file name. For example:

```
-MERGE INDEX1 WITH INDEX2 TO NEWINDEX↵
NEWINDEX.. NEW FILE↵
NAMES OF KEY FIELDS: SUBJECT↵
DATA DESCRIPTION FILE: DX↵

OK.

1810 RECORDS MERGED OF 1810

-
```

MERGE prompts only for the required components that are missing from the command.

SELECT AND PURGE

The SELECT command offers an easy-to-use procedure for retrieving information in a data file. The complementary PURGE command permits deletion of specific records from a data file. Both SELECT and PURGE have identical command forms and capabilities, the only difference being that SELECT retrieves specified records, whereas PURGE deletes specified records in a data file.

This discussion details the SELECT and PURGE commands in two ways: using only a data file and using both a data and activity file. The following pages provide complete documentation of the commands, including options and examples.

Both SELECT and PURGE include an important optional feature for reformatting the records in the output file. The user may request that output records contain only specific fields by following the output file name with a colon (:) and a list of field names separated by commas. For example, the user specifies three fields of interest in the selected records:

```
-SELECT FROM PERSONNEL:PDESC IF AREA HAS "CHEM" TO T:NAME,2B,DEPT,
2B,RATING

OK.

 NAME         DEPT  RATING
DURBAN          4     2
GOOD            2     1
SIMON           5     2
ZIMMERMAN       2     2


4 RECORDS SELECTED FROM 55

-
```

Similarly, a PURGE command, for example,

```
-PURGE FROM PERSONNEL:PDESC IF DEPT NOT=1 TO DEPT1PERSONNEL:
NAME,2B,EMP.NO,2B,AREA,2B,AREA2,2B,SALARY,CR
```

produces an output file named DEPT1PERSONNEL, which contains records consisting of NAME, EMP.NO, AREA, AREA2, and SALARY.

For reformatting output records, any of the items listed below may appear in the field list which follows the output file name.

Item	Meaning
field name	Prints specified field information.
"text"	Prints text enclosed in single or double quote marks.
nB	Prints n blanks.
n"text"	Prints text enclosed in single or double quote marks n times.
RECNO	Prints original sequence number of data record.
LENGTH	Prints original length of data record.
CR	Terminates logical line.
LF	Continues logical line onto next physical line.
utility date field	Prints requested date or time information.

NOTE: An automatic carriage return is printed at the end of each output record when the output file is the terminal and reformatting is specified. When reformatting and writing the output on a file, however, the user must specify CR in all cases to include a carriage return at the end of a record. See page 151 for further discussion of the reformatting capability in IML.

Using a Single Data File

When the user wants to select or purge records from a file according to field information in the record itself, he may type a command of the form:

```
                   data   description                         output
 -{SELECT} FROM   file  :  file        {IF conditions }  TO   file    ↵
  {PURGE }                             {FOR conditions}
                   name    name                               name
```

Required Components

Only data file, description file, and output file names are required. An IF clause is needed, however, to qualify which records in the data file are to be selected or deleted. For example, in our personnel application, the user enters an IF clause which specifies the employee numbers of the persons who no longer work for the company; to delete all such records in PERSONNEL, the user types

```
-PURGE FROM PERSONNEL:PDESC IF EMP.NO=27 OR 99 OR 117 ¬
TO UPDPERSONNEL ↵
```

in the EXECUTIVE. The output file, UPDPERSONNEL, contains all the records of the original PERSONNEL file except those which have 27, 99, or 117 for EMP.NO.

When the user omits an IF clause, all records qualify for the operation. For example:

```
-SELECT FROM PERSONNEL:PDESC TO DUPPERSONNEL ↵
DUPPERSONNEL.. NEW FILE↵

OK.


55 RECORDS SELECTED FROM 55

-
```

When an IF clause appears in the SELECT command, however, only the records which qualify are selected. The example below shows the same command as above executed with an IF clause specifying several conditions for selection.[1]

1 – The capabilities and construction of IF clauses are explained in detail on page 20.

```
-SELECT FROM PERSONNEL:PDESC IF AREA HAS "PHYS" AND (RATING=1 OR 2) ┐
 AND PROMOTABILITY=1↵

OUTPUT TO: T↵      The system prompts for the required output file name.

OK.
                        AREA                            RATING   PROMOTABILITY
                         ↓                                   ↓ ↓
CUMMINGS   80 29M M 2 5PHYS MATH  2  1  20000.00 711115 1 1 15.0 701112
GOLDEN     70 29M D 2 5PHYS MATH  5  7  18000.00 720301 1 1 12.2 700301
KNIGHT     44 40F D 3 4PHYS MATH  3  7  35000.00 711213 2 1  9.5 641210
PENTEL     68 24M S 0 5PHYS MATH  3  1  29000.00 710915 1 1 10.2 700101


4 RECORDS SELECTED FROM 55

-
```

Optional Components

The complete form of the SELECT or PURGE command for a single file, including optional components, is:

```
 |SELECT|                   data     description  |IF conditions |      output
-|PURGE | [PRIME]  FROM  file   :  file          |FOR conditions|  TO  file   : field list
                            name     name                                name
________________________________________________________________________________________

      key               complementary
BY   field  OTHERS  TO  output
     list               file            ↵
                        name
_______________________________________
```

The user may enter a BY clause after the output file name to request a simultaneous sequence check of the records in the data file. For example,

```
-SELECT FROM PERSONNEL:PDESC TO T BY EMP.NO↵
```

checks that each successive record is in numerical order by EMP.NO. The command aborts when a record is out of sequence and the output file, in this example, the terminal, would show the results of SELECT to that point. Similarly, the user may include a BY clause with a PURGE command. For example,

```
-PURGE FROM PERSONNEL:PDESC IF EMP.NO=27 OR 99 OR 117 ┐
TO UPDPERSONNEL BY EMP.NO↵
```

instructs PURGE to check that records are in EMP.NO order in the PERSONNEL file while it performs the specified deletions.

An OTHERS clause may appear at the end of the SELECT command to write the records which are not selected on a second output file. For example,

```
-SELECT FROM PERSONNEL:PDESC IF DATE.HIRE<700101 ┐
TO SENIORPERSONNEL OTHERS TO JUNIORPERSONNEL↵
```

simultaneously creates two output files: SENIORPERSONNEL contains the selected records which indicate a date of hire prior to January 1, 1970, and JUNIORPERSONNEL contains all other records from PERSONNEL. Similarly, an OTHERS clause may appear at the end of the PURGE command to write the deleted records on a separate file. For example,

```
-PURGE FROM PERSONNEL:PDESC IF EMP.NO=27 OR 99 OR 117
TO UPDPERSONNEL OTHERS TO TERMINATEDPERSONNEL
```

creates two output files: UPDPERSONNEL is the updated personnel file, and TERMINATEDPERSONNEL contains the records of employees just removed from the data file.

NOTE: If the user reformats the output records, the complementary output file is also reformatted. For example:

```
-SELECT FROM PERSONNEL:PDESC IF DATE.HIRE<700101
TO SENIORPERSONNEL:EMP.NO,NAME,DEPT,CR OTHERS TO JUNIORPERSONNEL
```

Both output files, SENIORPERSONNEL and JUNIORPERSONNEL, are reformatted to contain only three fields and a carriage return per record.

The PRIME option is available to select or purge only the *first* of several qualifying records which contain identical key field data. The BY clause must also appear in the command to specify the fields to check for duplicate information. For example, the command below uses the PRIME option to select one qualifying record for each area of specialization. The qualifying condition is that RATING must equal 1.

```
-SELECT [PRIME] FROM PERSONNELSTD:PDESC IF RATING=1 TO T BY AREA

OK.
                          AREA                                   RATING
                            ↓                                       ↓
CLOUTIER     9 60F D 2 1ACAD ART    3  8       5.25 720415 1 0   8.3 600823
GOOD        88 26F S 0 4CHEM ACCT   2  6   14675.00 711202 1 1   7.0 711119
HANSON      87 35F M 0 3EDUC ACAD   5  4   10000.00 720415 1 1   8.5 710415
HUGHES      42 29M D 2 2ELEC ACAD   3  3       6.95 720517 1 0   8.5 640523
VOGELSANG   65 28M M 0 4ENGL        2  4   12000.00 711108 1 0  10.0 691110
RILEY       39 52M M 5 4ENGR PHYS   1  5   22000.00 720412 1 1  10.0 640408
WEST        26 46M M 3 3MATH        1  6   23600.00 720316 1 1  12.5 610115
FLOYD       77 27F S 0 4MBA  ACCT   2  2   15000.00 711018 1 1  14.0 701112
PENTEL      68 24M S 0 5PHYS MATH   3  1   29000.00 710915 1 1  10.2 700101


9 RECORDS SELECTED FROM 55

-
```

The PRIME option with the PURGE command is very useful for locating records with duplicate information in a specific field or fields. For example,

```
-PURGE [PRIME] FROM PERSONNEL:PDESC TO DUPLICATES BY EMP.NO
```

creates a file, DUPLICATES, which contains any records with employee numbers that appear more than once in a data file. If PERSONNEL does not contain duplicate EMP.NO information in the records, the output file, DUPLICATES is blank; that is, all records were deleted.

NOTE: When using the PRIME option, the user first must sort the data file on the fields to be specified in the BY clause of the SELECT or PURGE command. If the original record arrangement in the data file is relevant, the user should include RECNO as the last field for the sort.

Using a Data and Activity File

SELECT and PURGE allow the user to work with an activity file in the selecting or purging procedure. The activity file contains records which specify the data file records to be selected or purged. For example, a file named LIST contains employee numbers of specific employees whose records the user wishes to examine.

```
-TYPE LIST ↵

13       Note that this employee number does not exist in the PERSONNEL file.
35
46
55
73
77
85
86
94

-
```

The LIST file is sorted by EMP.NO. Since the records of the data and activity files must be arranged in the same order, the user first sorts the PERSONNEL file in EMP.NO order. Then, the user types a SELECT command which uses LIST as an activity file to locate the desired records in the data file.

```
-SORT PERSONNEL:PDESC BY EMP.NO ↵

OK.

SORT FINISHED.

-SELECT LIST:LDESC FROM PERSONNEL:PDESC TO T BY EMP.NO ↵

OK.

O'CONNORS  35 52M M 2 4ENGR BIOL  5  1  24000.00 710615 1 1 10.0 630515
FULMER     46 29F S 0 5MATH MBA   4  1  22000.00 720118 1 1 15.0 650122
QUINLAN    55 29F S 0 4ENGR MATH  5  5  14500.00 711215 2 1  8.5 680515
BOSUNG     73 39M D 3 4PHYS MATH  5  7  13000.00 720101 3 0  5.5 700615
FLOYD      77 27F S 0 4MBA  ACCT  2  2  15000.00 711018 1 1 14.0 701112
YORK       85 34M M 0 4ARCH ART   3  8      5.00 711101 2 0 10.0 710301
MOREHEAD   86 39M S 0 4ENGR PHYS  5  5  13000.00 720315 2 1 10.0 710315
MARIN      94 21M S 0 4MBA  POLI  4  2   9600.00      0 0 0   .0 720612
```

```
8 RECORDS SELECTED FROM 55
1 RECORDS OF LIST FILE NOT FOUND IN PERSONNEL
-
```

This message refers to employee number 13.

The SELECT and PURGE commands may operate on binary and symbolic files at the same time. Numeric-type key fields need not have identical data types in the description files. The user may specify a binary data file and a symbolic activity file, a symbolic data file and a binary activity file, a symbolic data file and a symbolic activity file, or a binary data file and a binary activity file in a SELECT or PURGE command. The resulting output file is *always* the same file type as the data file.

Specifying an identical data type in the data and activity description files is not needed when working with a numeric-type key field in a BY clause. An I-, R-, D-, or N-type field in one file can be matched to an I-, R-, D-, or N-type field in the other file. A C-type field in one file, however, must be matched to a C-type field in the other file. In all cases, a key field must have the identical name in both description files.

Required Components

When working with an activity file, the BY clause is an essential component. In addition to specifying the record sequence for both files, the BY clause states the key field or fields that appear in both the data and activity records to determine a match for selecting or deleting records.

The general form of the SELECT or PURGE command is:

```
 {SELECT} activity    description          data   description       output      key
-{PURGE } file      : file          FROM   file : file          TO  file    BY  field ⊃
          name        name                 name   name              name        list
```

For example, the user has an activity file named TERMINATIONS, which contains a record for each terminated employee, specifying the name and employee number.

```
-TYPE TERMINATIONS ⊃

LIGHT       50
OAKDALE     81
CLIFF       83

-
```

The user wants to update the PERSONNEL data file by deleting the records of terminated employees. He uses TERMINATIONS as an activity file in the PURGE command, specifying that a data and activity record must match on EMP.NO for the data record to be deleted.

```
-PURGE TERMINATIONS:TDESC FROM PERSONNEL:PDESC TO REVPERSONNEL ¬
BY EMP.NO↵
REVPERSONNEL.. NEW FILE↵

OK.


3 RECORDS PURGED FROM 55

-
```

The output file, REVPERSONNEL, contains all the records in PERSONNEL minus the deleted records specified in TERMINATIONS.

Optional Components

The complete form of the SELECT or PURGE command is:

```
                     activity   description
  ⎧SELECT⎫           file     : file         ⎧IF  conditions ⎫
 -⎩PURGE ⎭ [SINGLE]  name       name         ⎩FOR conditions ⎭

       data     description                       output
FROM   file   : file         ⎧IF  conditions ⎫ TO file   : field list ⎧IF  interfile conditions ⎫
       name     name         ⎩FOR conditions ⎭    name                ⎩FOR interfile conditions ⎭

                          complementary
     key                  output
BY   field  OTHERS TO     file            ↵
     list                 name
```

The user may include IF clauses at three places in the command.[1] Records in either file can be screened by placing an IF clause after the file specification. Also, the output records may be qualified by comparing fields from the data and activity records in an interfile IF clause. Then the output file contains only records which meet all criteria.

For example, assume an activity file named REVIEWS contains, in each record, EMP.NO, REV.DATE, and NEWRATING. Assume RDESC is the name of its description file. The user wants to select records from PERSONNEL for those employees who had recent reviews, later than May 15, 1972 (720515), and whose new rating differs from the current rating in the data file by two or more points. He makes the desired selection by specifying two IF clauses, as follows:

```
–SELECT REVIEWS:RDESC IF REV.DATE>720515 FROM PERSONNEL:PDESC ¬
TO T IF NEWRATING:A–RATING:D>1 OR RATING:D–NEWRATING:A>1 ¬
BY EMP.NO↵
```

The first IF clause instructs SELECT to use only the activity records whose REV.DATE indicates a date after May 15, 1972. The last IF clause illustrates interfile conditions. The user wants to select the employee's record when there is a discrepancy of more than one point between the new and old ratings. Therefore, SELECT prints on the terminal (T is the output file specified)

1 – IF clauses are explained and illustrated in complete detail on page 20.

PERSONNEL records for employees who had recent reviews and receiving ratings of two or more points higher or lower than their last rating.

An OTHERS clause may appear at the end of the command to write the records from the data file which do not appear in the output file. The user requests the complementary output file in the same manner as described on page 90 for SELECT and PURGE with a single data file.

The SINGLE option in a SELECT or PURGE command matches an activity record with a single data record; normally, an activity record can match several data records.[1] For example, an activity file, SKILLS, contains records specifying areas of specialization and maximum acceptable ratings (1 is the highest).

```
-TYPE SKILLS

CHEM 3
MATH 1
MBA  3
PHYS 2

-
```

The user wants to select one record from the PERSONNEL data file for each activity record. First, he sorts the PERSONNEL file by AREA and within AREA by RATING.

```
-SORT PERSONNEL:PDESC TO SKPERSONNEL BY AREA,RATING
SKPERSONNEL.. NEW FILE

OK.


SORT FINISHED.

-
```

Now the user enters his SELECT command, specifying two IF clauses to qualify operations: First, data records with ratings of 0 are excluded from the command procedure, second, the rating in a selected data record must be equal to or less than the maximum rating in the matching activity record.

```
-SELECT [SINGLE] SKILLS:SDESC FROM SKPERSONNEL:PDESC IF RATING#0
TO T:NAME,3B,AREA,2B,RATING IF RATING:D <= RATING:A BY AREA

OK.


GOOD         CHEM    1
FULMER       MATH    1
FLOYD        MBA     1
CUMMINGS     PHYS    1

4 RECORDS SELECTED FROM 55

-
```

1 – See page 17 for an explanation of the data and activity record matching procedure.

Note the different output when the SINGLE option is omitted.

```
-SELECT SKILLS:SDESC FROM SKPERSONNEL:PDESC IF RATING#0 ↴
TO T:NAME,3B,AREA,2B,RATING IF RATING:D <= RATING:A BY AREA ↩

OK.

GOOD         CHEM    1
SIMON        CHEM    2
ZIMMERMAN    CHEM    2
DURBAN       CHEM    2
FULMER       MATH    1
MONROE       MATH    1
WEST         MATH    1
FLOYD        MBA     1
ELBERT       MBA     2
LEE          MBA     2
AVERY        MBA     2
CUMMINGS     PHYS    1
GOLDEN       PHYS    1
PENTEL       PHYS    1
KNIGHT       PHYS    2


15 RECORDS SELECTED FROM 55

-
```

The data file, SKPERSONNEL, is sorted primarily by AREA, because the data and activity files must be arranged by the key field; it is sorted by RATING within AREA so that the matching data record with the best rating appears first. Therefore, when the SINGLE option is used, the data record with the best rating is selected.

Two conditional clauses which are available for use with the SELECT and PURGE commands, and any other IML commands, are the FINISHING and the ABORTING clauses. The following table summarizes the effect of each of these conditional clauses when the given clause is true.

Conditional Clause	When Condition(s) True for Current Record
FINISHING $\begin{Bmatrix}\text{IF}\\\text{FOR}\end{Bmatrix}$ conditions	Command procedure terminates. The output file contains the results to this point.
ABORTING $\begin{Bmatrix}\text{IF}\\\text{FOR}\end{Bmatrix}$ conditions	Command procedure halts immediately.

It is suggested that the user specify an IF or FOR clause last when more than one conditional clause is desired; otherwise, an ABORTING or FINISHING clause may be ignored when true.

NOTE: When the first conditional clause is true, any other conditional clauses for the same file are ignored.

REPLACE

The REPLACE command offers a simple procedure for replacing one or more fields in a record and for replacing entire records in a data file. The REPLACE command always uses an activity file; it contains the information to substitute as well as the key field information to locate the relevant records in the data file.

Substituting Entire Records

The general form of the REPLACE command used for substituting entire records is:

```
          data    description          activity   description          output       key
-REPLACE  file  : file          WITH   file     : file            TO   file    BY   field↵
          name    name                 name       name                 name         list
```

Both the data and activity files should be described by the same description file.

In the example below, a file named CORRECTIONS is used as an activity file; it contains new records for certain employees in the PERSONNEL data file.

```
-TYPE CORRECTIONS↵

GOLDEN      70 29M D 2 5ENGR PHYS 10   3   18000.00 720301 1 1 12.2 700301
MOREHEAD    86 39M M 1 4ENGR PHYS 12   8   13000.00 720315 2 1 10.0 710315

-REPLACE PERSONNEL:PDESC WITH CORRECTIONS:PDESC TO CORRPERSONNEL
BY EMP.NO↵
CORRPERSONNEL.. NEW FILE↵

OK.

2 RECORDS REPLACED OF 55

-
```

The output file, CORRPERSONNEL, contains all the records from PERSONNEL and reflects the substitutions.

Substituting Specific Fields

The user may type a different form of the REPLACE command to replace specific fields within a record, rather than replace the entire record.

The general form of the REPLACE command used for substituting specific fields in the data file records is:

```
           data     description          activity  description
-REPLACE   file   : file          WITH   file    : file
           name     name                 name      name
```

```
   output  data                             data                                key
TO file  : field  WITH description  ,  field  WITH description ,...  BY  field⤶
   name    name                             name                                list
```

The word description appearing after WITH may be any of the following:

- One or more characters
- A number
- A utility field name
- An activity field name

The output file name is followed by a colon, and the data record fields to be replaced by "description" fields are specified. This information appears only when substituting fields within the records.

NOTE: The data and activity fields need not contain the same type of data; that is, the fields may be of different data types (binary or symbolic). Output files created with the REPLACE command are always *the same file type as the data file. Similarly, specifying identical data types in the data and activity description files is not necessary when working with a numeric-type key field in a BY clause. A C-type field in one file, however, must be matched to a C-type field in the other file. In all cases, a key field must have the identical name in both description files.*

For example, a file named REVIEWS contains records for particular employees, which specify current information on salary reviews.

```
-TYPE REVIEWS⤶

  35 720515  27000.00 1 1
  45 720515  23500.00 1 1
  53 720515  15000.00 2 1
  62 720515  12000.00 1 0
  63 720525  16500.00 2 1
  72 720515  16500.00 1 1
  75 720515  20000.00 1 1

-
```

The file RDESC contains the description of the data stored on REVIEWS.

```
-TYPE RDESC⤶

$$$
***BIN OR SYM:
S
***FIX OR VAR:
V
***WHICH TYPE:
L
***NUMBER OF LINES PER RECORD IS:
1
```

```
***  NAME, TYPE, START, LENGTH, DECIMAL PLACES
EMP.NO,N,1,4
REV.DATE,N,6,6
NEWSALARY,N,.,10,2
RATING,N,.,2
PROMOTABILITY,N,.,2


-
```

Each record contains an employee number and four fields related to an employee's review. To update the PERSONNEL file with the data in REVIEWS, the user types:

```
-REPLACE PERSONNEL:PDESC WITH REVIEWS:RDESC TO CURRENT:LAST.REV WITH ↴
REV.DATE, SALARY WITH NEWSALARY, RATING WITH RATING, PROMOTABILITY ↴
WITH PROMOTABILITY BY EMP.NO↵
CURRENT.. NEW FILE↵

OK.

7 RECORDS REPLACED OF 55

-
```

Optional Components

The complete general form of the REPLACE command is:

```
                           data   description
-REPLACE [SINGLE]  file  :  file        {IF conditions  }
                           name   name        {FOR conditions}
_____________________________________________________________

     activity  description                          output     data
WITH file    :  file        {IF conditions  }  TO   file    :  field
     name       name        {FOR conditions}        name       name
_____________________________________________________________________

                    data
WITH description ,  field  WITH description ,...
                    name
________________________________________________

                                     key
{IF interfile conditions  }  BY  field↵
{FOR interfile conditions}        list
______________________________________
```

The user may type an IF clause at three places in the command.[1] An IF clause can qualify the data records to be replaced, the activity records to be substituted, or it can qualify the execution of the REPLACE command by comparing a field in the data record with a field in the matching activity record.

1 – Refer to page 20 for a discussion of IF clauses.

The following examples illustrate the use of IF clauses for these three purposes. The command

```
–REPLACE PERSONNEL:PDESC IF JOB>10 WITH NEWJOBCODES:NDESC ↴
TO NEWPERSONNEL:JOB WITH NEWCODE BY EMP.NO↵
```

concerns only data records for which JOB>10 is true. Other data records are not altered by the command and appear unchanged in the output file. Similarly, the command

```
–REPLACE PERSONNEL:PDESC WITH REV:RDESC IF REV.DATE<730101 ↴
TO REVPERSONNEL BY EMP.NO↵
```

uses only the activity records which specify a review date before January 1, 1973.

The interfile IF clause appears after the output file specification. For example,

```
–REPLACE PERSONNEL:PDESC WITH NEWJOBCODES:NDESC ↴
TO NEWPERSONNEL:JOB WITH NEWCODE IF JOB:D>NEWCODE:A BY EMP.NO↵
```

compares JOB in the data record with NEWCODE in the matching activity record. When JOB is not greater than NEWCODE, the replacement is not performed and the output record is the unchanged data record. When the condition is true, the replacement takes place in the data record.

The SINGLE option allows the user to update several data records containing identical key fields, each with a single matching activity record. Normally, all such data records are updated with only the first matching activity record. See page 17 for further explanation of the SINGLE option.

UPDATE

The UPDATE command offers an important capability in the Information Management Library. It allows the user to manipulate information by specifying arithmetic and replacement operations, as desired. The user accesses UPDATE directly from the EXECUTIVE by typing the UPDATE command followed by a carriage return.

The discussion below explains the use of UPDATE in several parts. First, a sample problem illustrates the use of UPDATE to handle the various calculations required in computing the weekly wages for hourly and salaried employees. The built-in flexibility and power of UPDATE is demonstrated with this simplified example. Next, an explanation of UPDATE with an activity file is presented, detailing the required and optional command components. Using UPDATE with only a data file is discussed next. Finally, the structure of an UPDATE rules file is presented.

It is suggested that users who are unfamiliar with IML rules files refer to pages 27 through 41 before proceeding with the UPDATE discussion.

Sample Problem

The sample problem uses UPDATE to perform a simple computation of weekly wages for the employees in a small company.

The data file, WEEKLYHOURS, contains the following information for each employee:

Field Name	Field Contents
EMP.NO	Employee number.
RATE	H indicates hourly; A indicates annual.
HOURS	Hours worked during previous week.
WAGES	Weekly wages. This field is blank before the UPDATE command is given.

In our example, the data file contains records for 15 employees. Each record contains four fields related to the given employee.

```
-TYPE WEEKLYHOURS

   1 A 40     .00
   2 A 40     .00
   9 H 40     .00
  39 A 40     .00
  42 H 45     .00
  49 A 40     .00
  53 A 40     .00
  54 H 35     .00
  56 A 40     .00
  63 A 40     .00
  65 R 40     .00
  68 A 40     .00
  76 H 40     .00
  80 A 40     .00
  84 H 40     .00

-
```

Note the invalid rate code of R in this record.

A description file, WDESC, describes the characteristics of the data file.

```
-TYPE WDESC

$$$
***BIN OR SYM:
S
***FIX OR VAR:
V
***WHICH TYPE:
L
***NUMBER OF LINES PER RECORD IS:
1
***  NAME, TYPE, START, LENGTH, DECIMAL PLACES
EMP.NO,N,1,4
RATE,C,6,1
HOURS,N,8,2
WAGES,N,11,6,2

-
```

PERSONNEL1 is the company's personnel data file; it contains all employee data, including salary.[1] The description file named PDESC describes the data stored in PERSONNEL1. It is listed on page 81.

```
-TYPE PERSONNEL1↩

              EMP.NO                                     SALARY
                ↓                                           ↓
PARKER          1 48M M 4 4ENGR PHYS   1  1  38000.00 720101 1 1 10.0 600525
HARTMAN         2 43F D 1 3ENGL STEN   1  4  15000.00 711012 1 0 10.0 600525
CLOUTIER        9 60F D 2 1ACAD ART    3  8      5.25 720415 1 0  8.3 600823
RILEY          39 52M M 5 4ENGR PHYS   1  5  22000.00 720412 1 1 10.0 640408
HUGHES         42 29M D 2 2ELEC ACAD   3  3      6.95 720517 1 0  8.5 640523
CHEVES         49 39M D 3 4ENGR MATH   1  5  17000.00 720306 2 0 10.0 650306
STARK          53 30F M 1 3PHYS        2  7  13000.00 710502 3 1  5.5 671113
ROTH           54 55M M 1 2PLMB ACAD   5  3      4.50 720430 2 0 10.0 680430
WEISS          56 30M M 2 4LAW  GOVT   1  2  18500.00 711215 2 1  8.5 680524
ELBERT         63 51M D 3 4MBA  ACCT   3  2  13000.00 710303 2 1  7.6 690429
VOGELSANG      65 28M M 0 4ENGL        2  4  12000.00 711108 1 0 10.0 691110
PENTEL         68 24M S 0 5PHYS MATH   3  1  29000.00 710915 1 1 10.2 700101
LEWIS          76 32M M 2 2ELEC COMM   3  3      6.75 711107 2 0  5.5 701105
CUMMINGS       80 29M M 2 5PHYS MATH   2  1  20000.00 711115 1 1 15.0 701112
LARSON         84 38F M 2 1ACAD TYP    1  8      3.75 710930 2 1  6.5 701230

-
```

In our example, PERSONNEL1 is used as an activity file, because the specific data from its SALARY field is needed to compute the wages for the WEEKLYHOURS data file.

The actual UPDATE command is typed directly in the EXECUTIVE. It specifies the components required for the updating procedure: the data file, WEEKLYHOURS; its description file, WDESC; the activity file, PERSONNEL1; its description file, PDESC; an output file, WEEKLYWAGES, to contain the updated version of the data file; a key field, EMP.NO, to match an activity record with the relevant data record; and the name of the rules file, WAGERULES. The user may specify the required information with a single command; otherwise, UPDATE prompts for the needed information. The output on the following pages shows the execution and results of a single UPDATE command.

```
-UPDATE WEEKLYHOURS:WDESC WITH PERSONNEL1:PDESC TO WEEKLYWAGES ↴
BY EMP.NO AS PER WAGERULES↩
: LIST↩            The rules are listed for the purpose of the example.
10  DECLARE
20     TOTAL.WAGES,R,9,2
30  INITIAL        This section is executed before any records are processed.
40     TYPE "BE PREPARED TO REENTER RATE CODE INFORMATION"
50     TYPE "IF AN ERROR IS ENCOUNTERED IN THE UPDATING PROCEDURE.",CR
60  BEFORE         Before seeking a matching activity record, the user wants to check the rate code in the data record.
70     IF RATE NOT = "H" AND "A"
75        TYPE "RATE CODE MUST BE 'H' OR 'A'"
80        TYPE "PLEASE REENTER RATE CODE FOR EMPLOYEE #",EMP.NO,": ",NCR
85        INPUT RATE
```

1 – For this example, the PERSONNEL data file is shortened from 55 records and named PERSONNEL1.

```
88        TYPE CR,CR,CR
90   MATCHED
100    IF RATE="H"
110       IF.1 HOURS>40; 40*SALARY:A + (HOURS-40)*SALARY:A*1.5
                              TO WAGES     The wage is computed for those who worked overtime.
120       ELSE.1 SALARY:A*HOURS TO WAGES     The wage is computed for the
130    ORIF RATE="A"                         rest of the hourly employees.
140       SALARY:A/52 TO WAGES      The wage is computed for salaried employees.
150    ELSE        The following rules are executed if the rate code is not H or A.
160       TYPE "RATE CODE STILL INCORRECT"
170       TYPE "WAGES NOT COMPUTED FOR EMPLOYEE#",EMP.NO,CR,CR
180  AFTER         After the matching procedure, the cumulative wage total is updated.
190    WAGES + TOTAL.WAGES TO TOTAL.WAGES
200  FINAL         This section is executed after all records are processed.
210    TYPE "TOTAL WAGES COMPUTED ON ", @DATE, ":", TOTAL.WAGES
: RUN↵             The user executes the updating procedure.
WEEKLYWAGES.. NEW FILE↵

OK.


BE PREPARED TO REENTER RATE CODE INFORMATION
IF AN ERROR IS ENCOUNTERED IN THE UPDATING PROCEDURE.

RATE CODE MUST BE 'H' OR 'A'
PLEASE REENTER RATE CODE FOR EMPLOYEE #  65: A↵



TOTAL WAGES COMPUTED ON 10/29/72:  4915.71


15 RECORDS UPDATED OF 15

-
```

The output file, WEEKLYWAGES, has the identical format of the data file, WEEKLYHOURS. It contains all the records in the data file and reflects the updating performed to compute weekly wages.

```
-TYPE WEEKLYWAGES↵

   1 A 40 730.77
   2 A 40 288.46
   9 H 40 210.00
  39 A 40 423.08
  42 H 45 330.13
  49 A 40 326.92
  53 A 40 250.00
  54 H 35 157.50
```

```
56 A 40 355.77
63 A 40 250.00
65 A 40 230.77
68 A 40 557.69
76 H 40 270.00
80 A 40 384.62
84 H 40 150.00

-
```

Note that WEEKLYHOURS and PERSONNEL1 are not deleted and still contain the original information.

Using a Data and Activity File

The following discussion concerns UPDATE using an optional activity file in the updating process. The shortened command form for updating with a single file is presented on page 105.

The complete general form of the UPDATE command is:

```
                             data     description
-UPDATE  {[SINGLE]   }       file  :  file          {IF conditions }
         {[MULTIPLE] }       name     name          {FOR conditions}
_____________________________________________________________________

      activity    description
WITH  file     :  file          {IF conditions }
      name        name          {FOR conditions}
_____________________________________________________

    output                                      key                 rules
TO  file    {IF interfile conditions }     BY   field  AS  PER      file  ⮐
    name    {FOR interfile conditions}          list                name
_________________________________________________________________________
```

The command instructs UPDATE to update a data file, using information in an activity file. UPDATE creates a new data file which contains an updated record for each record in the original data file. Many of the command components specify information required for any updating; other components are optional. The required and optional components are discussed below.

NOTE: The data and activity files need not contain the same type of data; that is, the files may be of different data types (binary or symbolic). Output files created with the UPDATE command are always *the same file type as the data file. Similarly, specifying identical data types in the data and activity description files is not necessary when working with a numeric-type key field in a BY clause. A C-type field in one file, however, must be matched to a C-type field in the other file. In all cases, a key field must have the identical name in both description files.*

Required Components

The general form of the UPDATE command, including all required components, is:

```
         data    description          activity    description          output        key                rules
-UPDATE  file  : file           WITH  file     :  file            TO   file     BY   field  AS  PER     file ⮐
         name    name                 name        name                 name          list               name
______________________________________________________________________________________________________________
```

The user specifies the rules file name as T (for terminal) to enter the rules file statements initially.[1] The user may omit a description file name if a RETRIEVE structure file corresponding to the specified data or activity file exists in the user's directory.

Optional Components

The SINGLE and MULTIPLE options and the IF clauses are the optional components in the UPDATE command. See page 104 for the complete general command form which shows the location of all optional components. The purpose of each optional component is explained below.

Using IF clauses, the user may specify conditions to screen the records in a file. Records that do not meet the conditions are excluded from command operation. For example,

```
–UPDATE WEEKLYHOURS:WDESC IF RATE="H" OR "A" WITH PERSONNEL1:PDESC ┐
TO WAGES BY EMP.NO AS PER WAGERULES⮐
```

updates only those records which contain an A or H in the RATE field; records without an A or H for RATE appear unchanged in the output file.

Three IF clauses may appear in the UPDATE command, enabling the user to screen data, activity, and output records from the command procedure. For complete instructions and capabilities of the IF clauses, refer to page 20.

The SINGLE option makes it possible to update several data records containing identical key field information, each with a matching activity record. Only one matching activity record updates a data record; then the next matching activity record updates the next data record. Normally, all matching activity records update the first matching data record, and any subsequent data records with the same key field data are not updated with an activity record. See page 18 for an illustration of the matching procedure with the SINGLE option.

The MULTIPLE option provides an alternative method (to the SINGLE option) for matching data and activity records in the UPDATE procedure. Normally all activity records which specify the same key field data are used to update the first matching data record. When the MULTIPLE option is specified, a single activity record is used to update all data records that specify the same key field data. Refer to page 19 for an illustration of the matching procedure with the MULTIPLE option.

Using a Single Data File

UPDATE allows the user to update information in a file without using an activity file. In this situation, the form of the command is:

```
         data   description                        output            rules
–UPDATE  file  : file      ⎧IF conditions ⎫  TO  file    AS PER  file ⮐
                           ⎩FOR conditions⎭
         name    name                              name              name
```

For example, to update all the records in a file named INVENTORY according to the rules on SUBRULES, the command is:

```
–UPDATE INVENTORY:DESC TO UPDATED AS PER SUBRULES⮐
```

The IF clause is optional. When used, it qualifies the records for the command procedure. For example, to perform the same task as above, but only for records which have a code of 1, the user types:

1 – Complete documentation of rules files begins on page 27.

```
-UPDATE INVENTORY:DESC IF CODE=1 TO UPDATED AS PER SUBRULES
```

A BY clause may be included in the command after the output file name to request that UPDATE check the sequence of the records on a key field or fields. For example, to perform the same updating task as above, but also to check that the records are in PARTNO order, the user types:

```
-UPDATE INVENTORY:DESC IF CODE=1 TO SEQUPD BY PARTNO AS PER SUBRULES
```

Note that if the qualified records are not in PARTNO order, UPDATE so informs the user and aborts the command.

Structure of an UPDATE Rules File

The statements in an UPDATE rules file may be divided into as many as six sections, depending on the user's requirements and whether an activity file is used.[1] The table below presents the six sections available; each section is discussed following the table.

Section	Purpose
DECLARE	Creates working storage.
INITIAL	Assigns starting values to declared fields. If necessary, prints introductory messages.
BEFORE	Used with an activity file. Performs specified rules on data record before seeking a matching activity record.
MATCHED *or* DETAILS	Performs specified rules on data record, using matching activity record when activity file in use, or performs specified rules on data record when no activity file in use.
AFTER	Used with an activity file. Performs specified rules on data record after activity file procedure whether or not a matching activity record was found.
FINAL	After all records are processed, performs and prints final calculations of declared fields and prints final messages.

NOTE: The rules file may contain all or some of the sections; however, the sections must appear in the file in the order shown above. The BEFORE and AFTER sections are used only with an activity file. When no section name appears at the start of the rules file, UPDATE assumes it is the MATCHED or DETAILS section. The words MATCHED and DETAILS may be used interchangeably.

The DECLARE Section

The DECLARE section allows creation of working storage. It specifies new fields which do not exist in the input records, but which can be used to save data from records or calculations. For example,

1 – Rules file statements are presented on page 32.

```
10 DECLARE
20 TOTAL.WAGES,R,9,2
30 BASE,I,5
```

creates two new fields for use during the updating procedure.

The user may declare as many fields as desired, the only limitation being that a maximum of 100 lines is permitted in a rules file.

The form used to specify new fields is:

field name,data type,field length,decimal places

Note that only one field declaration may appear on a line.

The field name may contain as many as 31 characters; must begin with a letter; and may be any combination of letters, the digits 0 through 9, the period (.), and the character @. See page 47 for a list of reserved field names.

The data type may be C, N, I, R, or D. The data type includes the contents of the field, as follows:

Data Type	Meaning
C	Character data
N	Numeric data
I	Integer number
R	Real number
D	Double precision number

NOTE: With declared fields, the data types I, R, and D do not indicate binary data, but rather specify the most efficient storage for an integer, real, or double precision number, respectively.

The field length specifies the maximum number of characters needed to write the data in symbolic form.

The decimal places specification is necessary only for those fields that are to contain numbers with decimal points; it specifies the maximum number of digits to the right of the decimal point.

NOTE: Declared numeric fields are automatically initialized to zero; declared character fields are initialized to blanks.

Refer to page 141 for a description of the ROUNDIN and ROUNDOUT statements which may be used to round numeric and/or arithmetic data to conform to the field descriptions.

The INITIAL Section

The INITIAL section is executed at the beginning of the updating procedure, before any records are processed. Data and activity fields, therefore, may not be specified in this section. The user may assign starting values for declared fields in this section. For example,

```
40 INITIAL
50 100 TO BASE
```

assigns an initial value of 100 to a declared field named BASE. It is also possible to assign initial values to declared fields directly at the terminal, using an INPUT statement.[1] For example,

```
50 INITIAL
60 TYPE "PLEASE ENTER THE BASE VALUE"
70 INPUT BASE
```

assigns the value entered at the terminal to BASE.

The INITIAL section may also be used for printing introductory remarks. For example:

```
30 INITIAL
40 TYPE "BE PREPARED TO REENTER RATE CODE INFORMATION"
```

The BEFORE Section

The user may include the BEFORE section only when an activity file is used. The section contains rules to be executed for the data record being processed before a matching activity record is sought. Activity fields, therefore, may not appear in the BEFORE section.

The BEFORE section may specify any valid control and instruction statements to check and/or perform calculations on the data record fields. For example, the user wants to check the contents of the RATE field before using an activity record (in the MATCHED or DETAILS section, discussed below) to perform calculations based on the RATE data.

```
60  BEFORE
70    IF RATE NOT = "H" AND "A"
75       TYPE "RATE CODE MUST BE 'H' OR 'A'"
80       TYPE "PLEASE REENTER RATE CODE FOR EMPLOYEE #",EMP.NO,": ",NCR
85       INPUT RATE
88       TYPE CR,CR,CR
```

The MATCHED or DETAILS Section

The MATCHED or DETAILS section usually appears in all UPDATE rules files. If no activity file is used, the MATCHED or DETAILS section specifies rules to be performed on all records being processed; if an activity file is used, the section specifies rules to be performed only when a matching activity record is located.

The DETAILS or MATCHED section may contain any valid control and instruction statements to update the data records, specifying activity record fields followed by :A if an activity file is used. For example, the user wants to compute gross wages based on the hours and rate code in the data record and the salary in the matching activity record.

```
90  MATCHED        The user may type DETAILS or MATCHED.
100   IF RATE="H"
110      IF.1 HOURS>40; 40*SALARY:A + (HOURS-40)*SALARY:A*1.5
                        TO WAGES
```

1 – The INPUT statement is detailed on page 34.

```
120        ELSE.1 SALARY:A*HOURS TO WAGES
130     ORIF RATE="A"
140        SALARY:A/52 TO WAGES
150     ELSE
160        TYPE "RATE CODE STILL INCORRECT"
170        TYPE "WAGES NOT COMPUTED FOR EMPLOYEE#",EMP.NO,CR,CR
```

The AFTER Section

The user may include the AFTER section only when an activity file is used. The section contains rules to be executed for a data record after processing its matching activity records. Activity fields, therefore, may not appear in the AFTER section. The AFTER section is executed for each data record being processed whether or not a matching activity record was found.

The AFTER section may contain any valid control and instruction statements to specify the final updating of the data records, perform calculations with declared fields, and print information at the terminal. For example, the user wants to determine the total wages computed for all records processed.

```
180 AFTER
190 WAGES + TOTAL.WAGES TO TOTAL.WAGES
```

After the wage is computed for a given data record (in the MATCHED section), the wage is added to the current wage total. When all records are processed, TOTAL.WAGES contains the sum of the wages computed for all records.

Another example AFTER section is shown below. Assume the user wants to calculate gross wages, deductions, and net wages, and the MATCHED section uses the activity file to correct the hourly rate and number of dependents specified in certain data file records. After the activity file corrects the relevant data file records, the user wants to compute gross wage, deductions, and net wage for all records.

```
65   AFTER
70     80*HR.RATE TO GROSS
75     IF YTD.FICA < 468.00; GROSS*.052 TO FICA
80        IF.1 FICA + YTD.FICA > 468.00; 468.00 - YTD.FICA TO FICA
85     DO
90     GROSS*.15 - (DEPENDENTS*5.00) TO FIT
95     GROSS - FIT - FICA TO NETPAY
98     YTD.FICA + FICA TO YTD.FICA
99     YTD.FIT + FIT TO YTD.FIT
```

The UNAPPLIED Section

The UNAPPLIED rules section provides the user access to the activity records that are not matched with any data record during the UPDATE procedure. The UNAPPLIED section is similar to the BEFORE or AFTER section for data records. It may contain any of the IML instruction and control statements; the statements in the UNAPPLIED section may specify activity fields and declared fields but not data fields.

NOTE: Although it is permitted, the user is not required to type :A after activity field names in the UNAPPLIED section.

The following example uses the UNAPPLIED rules section.

The sample problem shows the use of the activity file PERSONNEL1, which contains pay information, to update the data file TIMECRD, which contains time card information. The output file produced specifies each employee's weekly pay information. The rules file computes the pay based on (1) the rate code (A or H) and the hours specified in the data file and (2) the pay rate specified in the activity file. When an employee name appears in PERSONNEL1 but not in TIMECRD, the UNAPPLIED rules section produces a message that alerts the user and provides him with the employee name and number.

The TIMECRD data file contains four fields in each record: EMP.NO, RATE, HOURS, and WAGES.

```
-TYPE TIMECRD

    1 A 40      .00
    2 A 40      .00
    9 H 40      .00
   39 A 40      .00
   42 H 45      .00
   49 A 40      .00
   53 A 40      .00
   54 H 35      .00
   56 A 40      .00
   63 A 40      .00
   65 A 40      .00
   68 A 40      .00
   76 H 40      .00
   80 A 40      .00
   84 H 40      .00


-
```

The PERSONNEL1 activity file contains 17 fields that specify all employee data. The fields in each record that are relevant to this application—EMP.NO, SALARY, and NAME—are shown below using the SELECT command.

```
-SELECT FROM PERSONNEL1:PDESC TO T:EMP.NO,5B,SALARY,5B,NAME

OK.

    1       38000.00       PARKER
    2       15000.00       HARTMAN
    9           5.25       CLOUTIER
   20           8.50       SUMNER
   39       22000.00       RILEY
   42           6.95       HUGHES
   49       17000.00       CHEVES
   53       13000.00       STARK
   54           4.50       ROTH
   56       18500.00       WEISS
   63        13000.00      ELBERT
   65        12000.00      VOGELSANG
   68        29000.00      PENTEL
   76            6.75      LEWIS
   80        20000.00      CUMMINGS
   84            3.75      LARSON


16 RECORDS SELECTED FROM 16


-
```

Note that the data file TIMECRD contains only 15 records, whereas the activity file PERSONNEL1 contains 16 records.

The actual UPDATE procedure is presented below.

```
-UPDATE TIMECRD:TDESC WITH PERSONNEL1:PDESC TO PAY
BY EMP.NO AS PER WAGERULES
: LIST
10 DECLARE
20   TOTAL.WAGES,R,9,2
30 MATCHED
40   IF RATE="H"
50     IF.1 HOURS>40;40*SALARY:A+(HOURS-40)*SALARY:A*1.5 TO WAGES
60     ELSE.1 SALARY:A*HOURS TO WAGES
70   ORIF RATE="A";SALARY:A/52 TO WAGES
80   ELSE TYPE "RATE CODE ERROR IN REC ",RECNO,"EMP ",EMP.NO,CR
90 AFTER
100  WAGES+TOTAL.WAGES TO TOTAL.WAGES
110 UNAPPLIED
120  TYPE NAME:A,"(EMPLOYEE # ",EMP.NO:A,") - NO TIME CARD.",CR,CR
180 FINAL
190  TYPE "TOTAL WAGES COMPUTED ON ",@DATE,":",TOTAL.WAGES
: RUN
PAY.. NEW FILE

OK.


SUMNER    (EMPLOYEE #    20) - NO TIME CARD.


TOTAL WAGES COMPUTED ON 04/27/74:  4915.71


15 RECORDS UPDATED OF 15
1 RECORDS OF PERSONNEL1 FILE NOT FOUND IN TIMECRD


-
```

The unapplied activity record informs the user that this employee is not included in the TIMECRD data file.

This message always appears when an activity record does not match any data record.

To complete the example, the resulting output file, PAY, is shown below.

```
-TYPE PAY↵

 1 A 40 730.77
 2 A 40 288.46
 9 H 40 210.00
39 A 40 423.08
42 H 45 330.13
49 A 40 326.92
53 A 40 250.00
54 H 35 157.50
56 A 40 355.77
63 A 40 250.00
65 A 40 230.77
68 A 40 557.69
76 H 40 270.00
80 A 40 384.62
84 H 40 150.00

-
```

The WAGES field now contains the appropriate weekly pay for each employee in the original TIMECRD data file.

The FINAL Section

In the FINAL section, the user may specify final calculations on declared fields and print information at the terminal. The FINAL section is executed after all records are processed. Data file fields, therefore, may not appear in this section.

Any valid control and instruction statements may appear in the FINAL section. For example, at the end of the updating procedure, the user wants to print the total wages computed and the current date.[1]

```
200 FINAL
210 TYPE "TOTAL WAGES COMPUTED ON ", @DATE, TOTAL.WAGES
```

CONVERT

The CONVERT command copies an existing data file to another form specified by the user. The command can, for example, convert a symbolic file to a binary form, a binary file to a symbolic form, a fixed length record file to a variable length record file, or a variable length record file to a fixed length record file with any number of fields rearranged, eliminated, or modified.

The user enters the CONVERT command directly in the EXECUTIVE. For example, the command

```
-CONVERT FILEX:XDESC TO FILEY:YDESC AS PER CONVRULES↵
```

1 – The current date is stored in an IML-declared field. See page 24 for details on utility fields.

instructs CONVERT to change the form of the file FILEX, which is described by XDESC, in writing the data on FILEY. The description file YDESC specifies the desired form for FILEY, and the CONVRULES file details how the fields as defined in XDESC are to be converted to the YDESC form.

Conversion of Binary and Symbolic Files

The discussion of CONVERT is organized in several parts. First, sample problems are presented, which illustrate the two possible methods of using CONVERT: with and without a rules file. The CONVERT command form and its components are discussed next. Finally, the use of a CONVERT rules file is detailed.

Refer to page 141 for a description of the ROUNDIN and ROUNDOUT statements which may be used with the CONVERT command to round numeric and/or arithmetic results to conform to field descriptions.

Sample Problems

The first example shows the use of CONVERT without a rules file. The user simply wants to convert a symbolic file to a binary file of fixed length records.

First, he creates a description file, specifying the desired form of the output data file. He uses the same field names in this description file as in the original description file. The original description file, PSDESC, and the new description file, PDESC, are shown below.

```
-TYPE PSDESC

$$$                                  Original form of data file.
***BIN OR SYM:
S
***FIX OR VAR:
V
***WHICH TYPE:
L
***NUMBER OF LINES PER RECORD IS:
1
***   NAME, TYPE, START, LENGTH, DECIMAL PLACES
NAME,C,1,9
EMP.NO,N,.,4
AGE,N,.,3
SEX,C,.,2
MAR.STAT,C,.,1
CHILDREN,N,.,2
LAST.DEG,N,.,2
AREA,C,.,5
AREA2,C,.,4
DEPT,N,.,3
JOB,N,.,3
SALARY,N,.,10,2
LAST.REV,N,.,7
```

```
RATING,N,.,2
PROMOTABILITY,N,.,2
PCT.RAISE,N,.,5,1
DATE.HIRE,N,.,7

-TYPE PDESC

$$$
***BIN OR SYM:
BIN
***FIX OR VAR:
FIX
***LENGTH:
63
***  NAME, TYPE, START, LENGTH, DECIMAL PLACES
NAME,C,1,9
EMP.NO,I,.,4
AGE,I,.,3
SEX,C,.,2
MAR.STAT,C,.,1
CHILDREN,I,.,2
LAST.DEG,I,.,2
AREA,C,.,5
AREA2,C,.,4
DEPT,I,.,3
JOB,I,.,3
SALARY,R,.,10,2
LAST.REV,I,.,7
RATING,I,.,2
PROMOTABILITY,I,.,2
PCT.RAISE,R,.,5,1
DATE.HIRE,I,.,7


-
```

Form of output data file.

Now the user enters the CONVERT command.

```
-CONVERT PERSONNEL:PSDESC TO BINPERSONNEL:PDESC BY NAMES
BINPERSONNEL.. NEW FILE

OK.


55 RECORDS CONVERTED OF 55

-
```

The BY NAMES clause directs CONVERT to perform the conversion automatically; all fields in PSDESC whose names also appear in PDESC are written on BINPERSONNEL in the specified form.

The second example shows the use of the CONVERT rules file. The user wants to convert the PERSONNEL data file to a fixed length record file. At the same time, he changes certain field names, eliminates several fields, and modifies the data in the DEPT field. The new description file, PFDESC, is shown below. The original description file, PSDESC, appears on page 113.

```
-TYPE PFDESC

$$$
***BIN OR SYM:
SYM
***FIX OR VAR:
FIX
***LENGTH:
25
***  NAME, TYPE, START, LENGTH, DECIMAL PLACES
EMPLOYEE,C,1,9
NUMBER,N,.,3
AGE,N,.,2
SEX,N,.,1
STATUS,C,.,1
DEGREE,N,.,1
DEPT,N,.,1
JOB,N,.,1
HIRED,N,.,6


-
```

The user enters the CONVERT command to perform the operation and includes the name of a rules file which specifies the method of copying and modifying the fields.

```
-CONVERT PERSONNEL:PSDESC TO PERSFILE:PFDESC AS PER RULES2
: LIST                          The rules file is listed for the purpose of this example.
10 NAME TO EMPLOYEE
20 EMP.NO TO NUMBER
30 AGE TO AGE
40 SEX TO SEX
50 MAR.STAT TO STATUS
60 LAST.DEG TO DEGREE
65 DEPT TO DEPT
70 4 TO DEPT IF DEPT=5
80 JOB TO JOB
90 DATE.HIRE TO HIRED
: RUN
PERSFILE.. NEW FILE

OK.


55 RECORDS CONVERTED OF 55


-
```

Command Components

The complete general form of the CONVERT command is:

```
           original  original                                  1
           data      description  ⎧IF conditions            ⎫
–CONVERT   file    : file         ⎨FINISHING IF conditions  ⎬
           name      name         ⎩ABORTING IF conditions   ⎭
_____________________________________________________________

   output   output
TO data   : description  ⎧BY NAMES             ⎫
   file     file         ⎩AS PER rules file name⎭↵
   name     name
_______________________________________________________
```

The command directs CONVERT to copy the information in the original data file to the output data file from the form specified on the original description file to the form specified on the output description file.

The BY NAMES clause is included when the user wants automatic copying of the fields whose names appear in both description files. The AS PER clause is included when one or more fields to be copied do not have the identical name in both description files, or when the user wants to modify any field information before writing it on the output data file. One of the two clauses is required in a CONVERT command. See page 118 for a complete discussion of the CONVERT rules file. The BY NAMES clause is discussed further on page 117.

The conditional clauses are optional. The user may include as many as three conditional clauses in the CONVERT command. When included, the specified condition(s) is tested in the current record of the original data file.

Specifying the Output Form

The output description file details the form of the output data file to be written by CONVERT. The output description file specifies whether the file is to be symbolic or binary with fixed length or variable length records and the name, location, length, and data type of all output fields.

The following paragraphs discuss various points to consider in creating the output description file.

If the user does not want to use a rules file, the fields to be converted must have the same field names in both description files. This situation is detailed on the following page.

When converting a symbolic file to binary form, the output description file must specify all numeric fields with the I, R, and D data types, as appropriate. For example, if a field is defined in the original description file as

KEY,N,.,3

it could be defined in the output description file as:

KEY,I,.,3

Similarly, a field defined in the original description file as

COST,N,.,10,2

1 – Note that for this component, braces do not indicate a choice of only one item.

could be defined in the output description file as:

COST,R,.,10,2

When converting a binary file to symbolic form, the output description file must specify numeric fields with the N data type. For example, if three fields are defined in the original description file as

KEY,I,.,3
COST,R,.,10,2
RATIO,D,.,16,7

they could be defined in the output description file as:

KEY,N,.,3
COST,N,.,10,2
RATIO,N,.,16,7

If the user wants the output data file to contain fixed length records, he must specify the record length in the output description file. If the file is to be symbolic, the record length is simply the sum of the individual field lengths plus any blanks. If the file is to be binary, the record length is determined by the type of data stored in the fields.

Performing Automatic Conversion

The BY NAMES clause performs an automatic conversion without using a rules file. It requires that all fields to be converted have the same field names in the output description file as in the original description file. The BY NAMES clause instructs CONVERT to write on the output data file, in the specified form, all fields in the original records whose names also appear in the output description file.

For example, assume that the original data and description files are PERSONNEL and PSDESC, respectively. PSDESC is shown on page 113. If the user wants to convert PERSONNEL to a binary file of variable length records containing the first five fields of PERSONNEL, he uses the output description file shown below.

```
-TYPE PBVDESC

$$$
***BIN OR SYM:
BIN
***FIX OR VAR:
VAR
***  NAME, TYPE, START, LENGTH, DECIMAL PLACES
NAME,C,1,9
EMP.NO,I,.,4
AGE,I,.,3
SEX,C,.,2
MAR.STAT,C,.,1

-
```

The user specifies the same field names as used in the original description file.

To perform the automatic conversion, the user enters the CONVERT command with the BY NAMES clause, specifying the original data file PERSONNEL, its description file PSDESC, a name for the output data file, and the output description file PBVDESC.

```
-CONVERT PERSONNEL:PSDESC TO BVPERSONNEL:PBVDESC BY NAMES↵
BVPERSONNEL.. NEW FILE↵


OK.



55 RECORDS CONVERTED OF 55



-
```

NOTE: When a field name in the output description file does not appear in the original description file, that field is blank in all output records.

Rules File

The CONVERT rules file is used to specify which fields of the original data file are to be written in which fields of the output data file, and how, if at all, the data is to be modified in the process. The CONVERT rules file is similar to other IML rules files, except there are no control statements (IF, ORIF, ELSE, and DO), and there are no sections.

Available Rules File Statements

The table below summarizes the rules file statements which may appear in a CONVERT rules file. Further discussion of the TYPE, @TYPE, INPUT, @INPUT, and DONE statements is presented below; the DELETE, FINISH, and ABORT statements are detailed on pp. 188–89.

Statement	Function
replacement	Specifies a field in the original data file or an expression to be written in a field in the output data file.
TYPE	Prints comments and/or data from the original file at the terminal or to a TOUT file.[1]
@TYPE	Prints comments and/or data from the original file at the terminal.
INPUT	Accepts data for a field in the output file from the terminal or from a command file.
@INPUT	Accepts data for a field in the output file from the terminal.
DONE	Instructs CONVERT to ignore the remaining statements in processing the current record.
DELETE	Erases the current record from the output file and begins processing the next record.

1 – TOUT files are explained in the *Tymshare TYMCOM-IX EXECUTIVE Reference Manual.*

Statement	Function
FINISH	Terminates record processing and ends the CONVERT procedure.
ABORT	Halts the CONVERT procedure and immediately returns control to the EXECUTIVE.

An IF or FOR modifier may appear at the end of any statement. Unless the user wants a statement performed for each record processed, he includes an IF or FOR modifier. For example, assume that a CONVERT rules file contains the statements:

```
10 NAME TO NAME
20 CODE TO KEY
30 TYPE "REENTER CODE FOR ",NAME IF CODE=0
40 INPUT KEY IF CODE=0
```

Lines 10 and 20 are executed for each record processed. Lines 30 and 40 are executed only when CODE equals zero.

NOTE: The IF or FOR modifier always specifies conditions to be tested in the original record.

Clarifications of Statement Functions

This discussion clarifies the use of replacement, TYPE, @TYPE, INPUT, and @INPUT statements in a CONVERT rules file. In particular, it specifies when a statement refers to a field in the original data file and when a statement refers to a field in the output data file.

A replacement statement specifies the information to be written in a particular field of the output records. Ordinarily, the form used is

field name$_1$ TO field name$_2$

where field name$_1$ is a field in the original file, and field name$_2$ is a field in the output file. For example, in the replacement statements

```
MAR.STAT TO STATUS
DATE.HIRE TO HIRED
```

MAR.STAT and DATE.HIRE are fields in the original file; STATUS and HIRED are fields in the output file.

If the user wants to modify the data, the left side of the replacement statement contains an arithmetic expression using fields from the original file or simply a number. For example, the following are valid replacement statements:

```
EMP.NO*10 TO EMP.NO
10 TO STATUS
```

NOTE: When the user wants to modify field information for certain conditions only, the replacement statement is followed by an IF or FOR modifier. For example:

10 TO STATUS IF MAR.STAT=0

The modified replacement statement should be preceded by the general replacement statement, such as:

MAR.STAT TO STATUS
10 TO STATUS IF MAR.STAT=0

If these statements were reversed, the conditional modification would be destroyed by the general replacement statement.

Field names in the TYPE and @TYPE statements always refer to the original data file. For example, when the TYPE statement

TYPE "REENTER CODE. CURRENT VALUE IS ",CODE FOR CODE>4

is executed and CODE is greater than 4, the system prints the comments in quote marks followed by the value of CODE in the current record of the original file.

The first field name in INPUT and @INPUT statements refers to the output data file. For example, when the INPUT statement

INPUT CODE FOR CODE>4

is executed and CODE is greater than 4 in the original record, a value is accepted for the CODE field of the output record. In this example, CODE is a field name in the original and output records. The first CODE field refers to the output record; the second CODE field refers to the original record, since an IF or FOR modifier always refers to the original record.

Sample Problem

The user wants to convert a symbolic file named INV to a binary file of variable length records, and in the process he wants to add an additional field and make several modifications.

The original file contains five fields per record: CODE, COST, QUANTITY, PART.NO, and DESC. In this application, the only meaningful CODE values are 1 and 2. The output file is to contain six fields per record: CODE, COST, QUANTITY, TOT.DOLLARS, PART.NO, and DESC.

The additional field TOT.DOLLARS is to contain the product of COST and QUANTITY. The modifications apply to several fields. First, the user wants to enter a new CODE value at the terminal if the original CODE value is not equal to 1 or 2. Next, because of an inventory change, part number 9 is now the same as part number 7. The user, therefore, wants to change part number 9 to part number 7 and change the descriptions of part numbers 7 and 9 to the same general description, WIDGET.

The original description file, DESC1, and the output description file, DESC2, are shown below.

```
-TYPE DESC1↵

$$$
***BIN OR SYM:
SYM
***FIX OR VAR:
VAR
***WHICH TYPE:
L
***NUMBER OF LINES PER RECORD IS:
1
```

```
***  NAME, TYPE, START, LENGTH, DECIMAL PLACES
CODE,N,1,1
COST,N,.,6,2
QUANTITY,N,.,4
PART.NO,N,.,2
DESC,C,.,15


-TYPE DESC2↵

$$$
***BIN OR SYM:
BIN
***FIX OR VAR:
VAR
***  NAME, TYPE, START, LENGTH, DECIMAL PLACES
CODE,I,4,1
COST,R,.,6,2
QUANTITY,I,.,4
TOT.DOLLARS,R,.,9,2
PART.NO,I,.,2
DESC,C,.,15


-
```

The user enters the CONVERT command to perform the operation.

```
-CONVERT INV:DESC1 TO NEWINV:DESC2 AS PER RULES7↵
: LIST↵                          The rules are listed for the purpose of this example.
10 CODE TO CODE
20 COST TO COST
30 QUANTITY TO QUANTITY
40 PART.NO TO PART.NO
50 DESC TO DESC
60 COST*QUANTITY TO TOT.DOLLARS
70 TYPE "REENTER CODE FOR PART.NO #",PART.NO  IF CODE NOT=1 AND 2
80 INPUT CODE IF CODE NOT=1 AND 2
90 7 TO PART.NO FOR PART.NO=9
99 "WIDGET" TO DESC FOR PART.NO=7 OR 9
: RUN↵                           The user executes the CONVERT command.
NEWINV.. NEW FILE↵

OK.


REENTER CODE FOR PART.NO # 4
1↵


641 RECORDS CONVERTED OF 641


-
```

SECTION 6
REPORT GENERATION

The REPORT command offers a comprehensive reporting facility suitable for generating reports of any description, including those on preprinted forms. Constructed in the same form as all other IML commands, REPORT permits the user to enter one simple command in the EXECUTIVE to initiate the report procedure.

Discussion of the REPORT command is organized in several parts. The first discussion presents a sample problem introducing the user to REPORT's convenient features and ease of use. The next discussion presents the general information about REPORT: the required and optional components of the command. Last, the actual REPORT rules file is detailed. The REPORT rules file contains six sections which allow the user to organize his report easily and handle all aspects of report generation. The additional rules statements and built-in functions, unique to REPORT, are presented in this discussion.

It is suggested that users who are unfamiliar with IML rules files read pages 27 through 41 before proceeding with the discussion of REPORT.

SAMPLE PROBLEM

The sample problem produces a report on the personnel data file. Because the user requests subtotals for each department, he first sorts the file by department. The data file SPERSONNEL is shown below. Its description file PDESC appears on page 81.

```
-TYPE SPERSONNEL↲                         DEPT
                                            ↓
CHEVES     49 39M D 3 4ENGR MATH  1  5  17000.00 720306 2 0 10.0 650306
HARTMAN     2 43F D 1 3ENGL STEN  1  4  15000.00 711012 1 0 10.0 600525
KINNEY     62 28F M 1 3ENGL EDUC  1  4  10500.00 710415 1 1  8.0 690415
LARSON     84 38F M 2 1ACAD TYP   1  8      3.75 710930 2 1  6.5 701230
MANDEL     91 42M M 2 2STEN MATH  1  8      5.00      0 0 0   .0 720315
MARTIN      7 39M M 3 4ENGR MATH  1  1  25400.00 711213 1 1 10.0 600601
PARKER      1 48M M 4 4ENGR PHYS  1  1  38000.00 720101 1 1 10.0 600525
RILEY      39 52M M 5 4ENGR PHYS  1  5  22000.00 720412 1 1 10.0 640408
WEISS      56 30M M 2 4LAW  GOVT  1  2  18500.00 711215 2 1  8.5 680524
WEST       26 46M M 3 3MATH       1  6  23600.00 720316 1 1 12.5 610115
AVERY      36 37M M 0 4MBA  HIST  2  2  16800.00 720217 2 1  8.5 640212
CUMMINGS   80 29M M 2 5PHYS MATH  2  1  20000.00 711115 1 1 15.0 701112
```

```
FLINKER     58 28F M 0 4EDUC ECON  2  4  11000.00 711215 1 1 10.0 680715
FLOYD       77 27F S 0 4MBA  ACCT  2  2  15000.00 711018 1 1 14.0 701112
GOOD        88 26F S 0 4CHEM ACCT  2  6  14675.00 711202 1 1  7.0 711119
LIGHT       50 31M M 3 3HIST POLI  2  6  10400.00 710209 3 1  5.0 650727
SHAMES      29 49M M 2 3EDUC       2  4  16000.00 720201 2 0  8.5 610901
STARK       53 30F M 1 3PHYS       2  7  13000.00 710502 3 1  5.5 671113
VOGELSANG   65 28M M 0 4ENGL       2  4  12000.00 711108 1 0 10.0 691110
ZIMMERMAN   61 39M M 3 5CHEM MATH  2  6  19500.00 720315 2 0 10.0 690301
BORNEMAN    72 29M S 0 5PHYS MATH  3  7  12800.00 710515 3 0  5.5 700515
CLIFF       83 31M M 4 4LAW  ENGR  3  2  14000.00 710212 4 0  4.0 701217
CLOUTIER     9 60F D 2 1ACAD ART   3  8      5.25 720415 1 0  8.3 600823
ELBERT      63 51M D 3 4MBA  ACCT  3  2  13000.00 710303 2 1  7.6 690429
HUGHES      42 29M D 2 2ELEC ACAD  3  3      6.95 720517 1 0  8.5 640523
KNIGHT      44 40F D 3 4PHYS MATH  3  7  35000.00 711213 2 1  9.5 641210
LASKER      95 46M M 2 3SOC        3  4   9700.00      0 0 0   .0 720625
LEE         45 37M M 6 4MBA  PHYS  3  2  19500.00 710514 2 1  6.8 641218
LEMBERG     71 32M S 0 4MATH       3  6  11000.00 711015 3 1  6.3 700506
LEWIS       76 32M M 2 2ELEC COMM  3  3      6.75 711107 2 0  5.5 701105
LINDEN      90 28F S 0 3ENGL EDUC  3  4   7500.00      0 0 0   .0 720315
OAKDALE     81 41M W 2 3GEOL MATH  3  6  10000.00 710609 3 1  8.5 701203
PENTEL      68 24M S 0 5PHYS MATH  3  1  29000.00 710915 1 1 10.2 700101
RUDMAN      93 20F S 0 3MATH PHYS  3  6   8500.00      0 0 0   .0 720506
SANDERS     60 45M M 3 2ELEC COMM  3  3      5.50 720215 3 0  5.0 690215
SUMNER      20 48M M 2 2ELEC COMM  3  3      8.50 710615 2 0  6.5 601001
WARNER      34 42M D 1 3ENGL       3  4  14000.00 720209 2 0  8.5 630210
YORK        85 34M M 0 4ARCH ART   3  8      5.00 711101 2 0 10.0 710301
DURBAN      52 27M S 0 4CHEM MATH  4  6  16500.00 720312 2 0  6.5 660310
FULMER      46 29F S 0 5MATH MBA   4  1  22000.00 720118 1 1 15.0 650122
MARIN       94 21M S 0 4MBA  POLI  4  2   9600.00      0 0 0   .0 720612
MONROE      32 31F M 0 3MATH ENGL  4  6  15000.00 720412 1 1 10.0 620627
WAGNER      89 22F S 0 2ACAD       4  8      3.00      0 0 0   .0 720315
ABNER       92 38M M 0 4ELEC PHYS  5  3  12000.00      0 0 0   .0 720415
BOSUNG      73 39M D 3 4PHYS MATH  5  7  13000.00 720101 3 0  5.5 700615
DUNCAN      64 29M S 0 3MATH       5  6  13500.00 711215 2 0  8.0 690630
GOLDEN      70 29M D 2 5PHYS MATH  5  7  18000.00 720301 1 1 12.2 700301
HANSON      87 35F M 0 3EDUC ACAD  5  4  10000.00 720415 1 1  8.5 710415
JOHNSON      4 62M M 4 4ENGR MATH  5  5  16800.00 720115 2 0 10.0 600525
MOREHEAD    86 39M S 0 4ENGR PHYS  5  5  13000.00 720315 2 1 10.0 710315
O'CONNORS   35 52M M 2 4ENGR BIOL  5  1  24000.00 710615 1 1 10.0 630515
QUINLAN     55 29F S 0 4ENGR MATH  5  5  14500.00 711215 2 1  8.5 680515
RINSLER     75 48M M 4 4MATH PHIL  5  6  16800.00 710315 2 1  8.5 700915
ROTH        54 55M M 1 2PLMB ACAD  5  3      4.50 720430 2 0 10.0 680430
SIMON       74 25M S 0 5CHEM BIOL  5  6  21000.00 710715 2 1 12.0 700715

-
```

The user requests his report by typing a REPORT command directly in the EXECUTIVE. He specifies the data file name and its description file, an output file name for the actual report, and the name of the file which contains the report rules.

```
-REPORT SPERSONNEL:PDESC TO PERSONNELREPORT AS PER PERSONNELRPTDESC↵
: LIST↵
```

The user requests a listing of the REPORT rules.

```
10  DECLARE
20      CHEM.COUNT,I,3
30      ENGR.COUNT,I,3
40      SR.EMP,I,3
50      JR.EMP,I,3
60      YR.WAGE,R,9,2
```

Five temporary fields are created within the report procedure.

```
70      FORMAT 1: $$,$$$,$$$.DD
```

The user declares a picture format to be used in printing certain fields below.

```
80  INITIAL
86      5 TO MARGIN
90      SKIP TO 3
100     PRINT 15B,"PERSONNEL SUMMARY REPORT",CR,15B,@WEEKDAY,4B,
         @CDATE,CR
130 HEADINGS
```

This section specifies the headings for each page of the report.

```
131     IF PAGE NOT=1
135     PRINT PAGE,CR,"PERSONNEL SUMMARY",CR,@DATE,CR
140     SKIP 6
145     ELSE SKIP 3
148     DO
150     PRINT "  NAME     NUMBER  DEPT  JOB     SALARY    RATING",
          "  PROMOTABILITY",CR,CR
160 DETAILS
```

This section specifies instructions to be executed for each record in the file.

```
170     IF "CHEM" IN AREA OR AREA2
175        CHEM.COUNT +1 TO CHEM.COUNT
180     IF "ENGR" IN AREA OR AREA2
185        ENGR.COUNT +1 TO ENGR.COUNT
```

The IF statements check for certain conditions and increment the counter when appropriate.

```
190     IF DATE.HIRE<700101
195        SR.EMP +1 TO SR.EMP
200     ELSE JR.EMP +1 TO JR.EMP
210     IF SALARY<25.
```

Hourly wages are converted to an annual figure.

```
215        SALARY*2080 TO YR.WAGE
220     ELSE SALARY TO YR.WAGE
230     DO
240     PRINT NAME,EMP.NO,3B,DEPT,3B,JOB,4B,YR.WAGE,5B,RATING,
         6B,PROMOTABILITY,CR
250 TOTALS
```

In this section, the user requests subtotals desired.

```
260     ON DEPT
265     SKIP 1
268     PRINT "-----------------------------------------------
---------",CR
270     PRINT 8B,"*TOTAL SALARY*",5B,SUM YR.WAGE(1)
280     PRINT 3B,"AVERAGE ",2B,AVG YR.WAGE(1),CR,CR,CR
285     IF LINE>55
287        SKIP TO TOP
```

SUM and AVG are built-in functions in REPORT.

```
290 FINAL
```

This section specifies desired information after all records are processed.

```
300     PRINT CR,CR
301     PRINT "FINAL TOTALS",CR,"------------------------------------
------------------------",CR,CR
310     PRINT 8B,"*TOTAL PAYROLL*",4B,SUM YR.WAGE(1),CR
312     PRINT CR,"EMPLOYEE CHARACTERISTICS:",CR,CR
320     PRINT "SENIOR EMPLOYEES ",SR.EMP,CR
330     PRINT "JUNIOR EMPLOYEES ",JR.EMP,CR
```

Picture format 1 is used to print the sum of YR.WAGE.

```
340     PRINT "EMPLOYEES WITH CHEMISTRY BACKGROUND ",2B,CHEM.COUNT,CR
350     PRINT "EMPLOYEES WITH ENGINEERING BACKGROUND ",ENGR.COUNT,CR
355     SKIP TO 61
: RUN↵
PERSONNELREPORT.. NEW FILE↵

OK.


REPORT FINISHED.


-
```

The user executes the report procedure. He may make changes to the REPORT rules before executing the command.

REPORT writes the requested report on the output file and returns control to the EXECUTIVE.

PERSONNELREPORT is now in the user's directory. The file is listed on the following pages.

```
                 PERSONNEL SUMMARY REPORT
                 MONDAY       OCT.30,1972


  NAME      NUMBER  DEPT  JOB    SALARY    RATING  PROMOTABILITY

CHEVES        49     1     5    17000.00      2        0
HARTMAN        2     1     4    15000.00      1        0
KINNEY        62     1     4    10500.00      1        1
LARSON        84     1     8     7800.00      2        1
MANDEL        91     1     8    10400.00      0        0
MARTIN         7     1     1    25400.00      1        1
PARKER         1     1     1    38000.00      1        1
RILEY         39     1     5    22000.00      1        1
WEISS         56     1     2    18500.00      2        1
WEST          26     1     6    23600.00      1        1

-------------------------------------------------------------
        *TOTAL SALARY*      $188,200.00   AVERAGE    $18,820.00


AVERY         36     2     2    16800.00      2        1
CUMMINGS      80     2     1    20000.00      1        1
FLINKER       58     2     4    11000.00      1        1
FLOYD         77     2     2    15000.00      1        1
GOOD          88     2     6    14675.00      1        1
LIGHT         50     2     6    10400.00      3        1
SHAMES        29     2     4    16000.00      2        0
STARK         53     2     7    13000.00      3        1
VOGELSANG     65     2     4    12000.00      1        0
ZIMMERMAN     61     2     6    19500.00      2        0

-------------------------------------------------------------
        *TOTAL SALARY*      $148,375.00   AVERAGE    $14,837.50


BORNEMAN      72     3     7    12800.00      3        0
CLIFF         83     3     2    14000.00      4        0
CLOUTIER       9     3     8    10920.00      1        0
ELBERT        63     3     2    13000.00      2        1
HUGHES        42     3     3    14456.00      1        0
KNIGHT        44     3     7    35000.00      2        1
LASKER        95     3     4     9700.00      0        0
LEE           45     3     2    19500.00      2        1
LEMBERG       71     3     6    11000.00      3        1
LEWIS         76     3     3    14040.00      2        0
LINDEN        90     3     4     7500.00      0        0
OAKDALE       81     3     6    10000.00      3        1
PENTEL        68     3     1    29000.00      1        1
RUDMAN        93     3     6     8500.00      0        0
SANDERS       60     3     3    11440.00      3        0
SUMNER        20     3     3    17680.00      2        0
WARNER        34     3     4    14000.00      2        0
YORK          85     3     8    10400.00      2        0

-------------------------------------------------------------
        *TOTAL SALARY*      $262,936.00   AVERAGE    $14,607.56
  2
PERSONNEL SUMMARY
10/30/72



  NAME      NUMBER  DEPT   JOB     SALARY    RATING  PROMOTABILITY


DURBAN       52      4      6     16500.00      2       0
FULMER       46      4      1     22000.00      1       1
MARIN        94      4      2      9600.00      0       0
MONROE       32      4      6     15000.00      1       1
WAGNER       89      4      8      6240.00      0       0

--------------------------------------------------------------
        *TOTAL SALARY*        $69,340.00    AVERAGE    $13,868.00


ABNER        92      5      3     12000.00      0       0
BOSUNG       73      5      7     13000.00      3       0
DUNCAN       64      5      6     13500.00      2       0
GOLDEN       70      5      7     18000.00      1       1
HANSON       87      5      4     10000.00      1       1
JOHNSON       4      5      5     16800.00      2       0
MOREHEAD     86      5      5     13000.00      2       1
O'CONNORS    35      5      1     24000.00      1       1
QUINLAN      55      5      5     14500.00      2       1
RINSLER      75      5      6     16800.00      2       1
ROTH         54      5      3      9360.00      2       0
SIMON        74      5      6     21000.00      2       1

--------------------------------------------------------------
        *TOTAL SALARY*       $181,960.00    AVERAGE    $15,163.33



FINAL TOTALS
--------------------------------------------------------------

        *TOTAL PAYROLL*     $850,811.00

EMPLOYEE CHARACTERISTICS:

SENIOR EMPLOYEES   31
JUNIOR EMPLOYEES   24
EMPLOYEES WITH CHEMISTRY BACKGROUND      4
EMPLOYEES WITH ENGINEERING BACKGROUND    9
```

GENERAL INFORMATION

The REPORT command assists the user in preparing a report based on the information in a data file.

The user calls REPORT directly in the EXECUTIVE by typing all or part of a REPORT command. The form of the command showing only required information is:

```
        data    description       output          rules
−REPORT file  : file          TO  file    AS PER  file ↵
        name    name              name            name
```

The report is prepared from the data file as it is defined by the description file, and written on the output file according to the layout and calculations in the REPORT rules file. For example,

```
−REPORT INVENTORY:INVDES TO INVFINAL AS PER INVSPEC↵
```

creates a report based on the data in INVENTORY which is described by INVDES, and writes the report on INVFINAL in accordance with the instructions on INVSPEC.

The user may specify any data file, including a RETRIEVE file, for the report. The description file is the same as used with all IML commands; it describes the type of data file and the location of the fields within a record. The user may create a description file with the DEFINE command, use an existing description file, or enter T as the description file name. When the user enters T as the description file name, REPORT prompts for the needed data file description.[1] When working with RETRIEVE files, the user may omit the data description file name. REPORT automatically seeks a corresponding RETRIEVE structure file in the user's directory. The output file name specifies the file on which the finished report is written; the output file may be T to print the report directly at the terminal. The REPORT rules file specifies the desired layout of the report, the calculations, and the information to be printed.

The user creates a REPORT rules file most easily by specifying the REPORT rules file name as T, allowing the system to prompt for and check each line as it is entered. In this situation, REPORT requests a file name on which to save the REPORT rules. For example,

```
−REPORT INVENTORY:INVDES TO INVFINAL AS PER T↵
SAVE REPORT RULES ON: INVRPT↵
```

allows the user to enter a new REPORT rules file and instructs the system to write the rules in INVRPT. See page 27 for an explanation of creating a rules file.

An optional IF clause may appear in the REPORT command to qualify which records from the data file may be used in producing the report. For example, the command

```
−REPORT PERSONNEL:PDESC IF DEPT=5 TO DEPT5RPT AS PER RPTDESC↵
```

requests a report based on all records in PERSONNEL for which the DEPT is 5. The system writes the report on DEPT5RPT, using the RPTDESC REPORT rules file.

THE REPORT RULES FILE

The REPORT rules file contains the user's report specifications, which define the report layout, such as the title and page headings; specify optional picture formats for printing individual fields, such as with dollar signs and commas; and request calculations, subtotals, and final totals. The REPORT rules file is structured to handle simple reports with a minimum of instructions, yet offers extensive capabilities to make any report easy to design and produce.

1 – See page 43 for a description of DEFINE and entering the file description.

Discussion of the rules file is organized in several parts. An overview is presented first, introducing the design and various features of the REPORT rules file. Next, the numerous built-in functions are described. Picture formatting, including an explanation of all available format characters and examples of their use, is presented in the following discussion. The additional statements that may appear in a REPORT rules file are detailed next.[1] Finally, the sections in which REPORT rules statements are organized are described, presenting the purpose, special features, and examples.

Overview

The REPORT rules file is organized conveniently in six sections: DECLARE, INITIAL, HEADINGS, DETAILS, OTHERS, TOTAL, and FINAL. Each section has a special function and is executed when appropriate. For example, the HEADINGS section specifies page headings for each page of the report, and is executed every time the report continues or skips to a new page. The FINAL section, on the other hand, specifies final totals and calculations to be printed, and is executed only once after all records in the file are processed. Each REPORT section is explained in detail below; the purpose and special features are outlined for each section. Some of the examples given also appear in the sample problem on page 123.

REPORT contains numerous built-in functions which make it convenient for the user to describe his specific and general report requirements. REPORT offers several descriptive information functions, such as @DATE, which contains the current date, or LINE, which contains the current line number of the report page being printed. These functions are available without requiring initial data from the user. In addition, several functions are available to control the layout of the report; for example, LINES or MARGIN controls the number of lines per page or the number of blank lines at the bottom of each page, respectively. REPORT presets some of these functions, for example, LINES as 66 per page and MARGIN as 0; the user may change the values easily. For example,

```
5 TO MARGIN
```

specifies that five blank lines are to be printed at the bottom of each report page.

REPORT permits the user to specify a format to print field information if he wants to alter the format as defined in the data description file. Various picture format characters are available for this purpose; they appear on page 133. For example, suppose a field named SALARY is defined in the data description file as:

```
SALARY,N,.,8,2
```

If the user wants to print the value of SALARY preceded by a dollar sign ($) and with commas in the number, he may enter a PRINT instruction, such as:

```
PRINT SALARY($$$,$$$.DD)
```

Also, suppose he wishes to print the sum of SALARY in all records.[2] The statement

```
PRINT SUM SALARY
```

may cause an error, since the sum may require more than the eight-character field length used in the field description for SALARY. However, using a picture format, such as

```
PRINT SUM SALARY(DDDDDDD.DD)
```

he avoids the problem.

1 – The rules file statements common to all rules file are presented on page 32.
2 – The SUM function is discussed on page 147.

Finally, REPORT includes four basic arithmetic functions to make it easy for the user to determine the maximum, minimum, average, or sum of the values in a specified field. These arithmetic functions are valid only in the TOTALS and FINAL sections, and are described on page 147.

It is suggested that the user read this section in its entirety before he designs a REPORT rules file. In this way, he may note the numerous conveniences built into the IML reporting capability, as well as the organization of the REPORT sections and their separate functions.

Built-in Utility Functions

Several field names and clauses are automatically defined by REPORT to help the user describe his report and to control paging and line spacing. These functions may appear in any section of the REPORT rules file. The tables below list all the available utility functions with their meanings. In addition, the tables specify the type of data (character or numeric) and length, as appropriate.

Utility Fields for Descriptive Information

Utility Field Name	Prints	Data Type and Length	Example
@DATE	Current date as MM/DD/YY.	Character, 8	11/02/73
@CDATE	Current date as MMM.DD,YYYY.	Character, 11	MAR.15,1975
@TIME	Current time as HH:MM.	Character, 5	14:30
@CTIME	Current time as HH:MM AM or PM.	Character, 8	2:30 PM
@WEEKDAY	Name of current day of week.	Character, 9	TUESDAY
@CALMONTH	Name of current month as MMM.	Character, 3	APR
@DAY	Current day as DD.	Numeric, 2	15
@MONTH	Current month as MM.	Numeric, 2	10
@YEAR	Current year as YYYY.	Numeric, 4	1974
@NDATE	Current date as YYMMDD.	Numeric, 6	731231
RECNO	Number of record being processed.	Numeric, 5	78
LENGTH	Length of record being processed.	Numeric, 5	120
PAGE	Current page number of report.	Numeric, 3	6
LINE	Current line number of report page.	Numeric, 2	52
HPOS	Current print position on line being written. The first print position on a line is 1. Whenever a carriage return, line feed, or new page occurs, HPOS is reset to 1.	Numeric, 3	42

(Table continues)

Utility Field Name	Prints	Data Type and Length	Example
nB	Specified number of blanks, where n may range from 1 to 80.	Character, n	16B
"text"	Text enclosed in single or double quote marks.	Character, number of characters in quote marks	"MESSAGE"
n"text"	Text enclosed in single or double quote marks n times. The number of characters in quote marks times n may not exceed 80.	Character, number of characters in quote marks times n	70"–"
CR	Carriage return, that is, spaces up one line and returns to left margin. CR terminates a logical line.[1]	Character, 1	
LF	Line feed, as above, but continues a logical line.[1]	Character, 1	

Utility Statements for Paging and Line Spacing

Utility Statement and Meaning	Example
n TO MARGIN Specifies the number of blank lines at the bottom of a page. MARGIN is preset to 0 but may be set to n, where n is a positive number or expression, if an automatic bottom margin is desired on each page. The statement may be followed by a FOR modifier.	7 TO MARGIN
n TO LINES Specifies the number of lines, including MARGIN, printed per page, where n is a number or an expression. It is preset to 66 but may be changed at any time. The statement may be followed by a FOR modifier.	55 TO LINES FOR RECNO>100
n PAGE DIVIDER IS item list Instructs REPORT to print a page divider at the top of every report page. This feature must be specified in the DECLARE section. The item list may be: CR — carriage return LF — line feed "characters" *or* 'characters' — Any characters to be used as a page divider. When a page divider is declared, REPORT prints the specified page divider at the top of every report page, whether or not an OFFHEADINGS statement is in effect or a HEADINGS section appears in the rules file.	20 PAGE DIVIDER IS "–",CR,CR,CR,CR 20 PAGE DIVIDER IS CR,".",LF,CR 20 PAGE DIVIDER IS '–',LF,'–',CR

1 – A logical line consists of a series of characters terminated by a carriage return. A line feed continues a logical line, permitting a logical line to occupy more than one physical line.

Utility Statement and Meaning	Example
NOTE: At least one CR or LF should be specified at the end of the item list unless the next line of the report is to be printed on the same line as the page divider. The page divider lines are counted in the number of lines per page.	
'c' TO TOF *or* n TO TOF Top of Form character or number. When TOF appears in a PRINT statement, it executes the TOF as well as printing the specified page headings. The character or number specified is represented by c or n respectively. The n character instructs REPORT to use the ASCII character equivalent to the internal decimal code which n represents. *NOTE: Control L is TOF unless reset with this utility.*	200 92 TO TOF 210 PRINT TOF 200 '¢' TO TOF 210 PRINT TOF
SKIP TO TOP Skips to the top of the next page and automatically prints specified page headings. The statement may be followed by a FOR modifier.	SKIP TO TOP FOR LINE>52
SKIP TO n Skips to the nth line on the report page, where n may be a number or an expression. The statement may be followed by a FOR modifier.	SKIP TO 20
SKIP n Skips n lines, where n may be a number or an expression. The statement may be followed by a FOR modifier.	SKIP 4
SPACE n Prints n blanks, where n may be a number or an expression. The statement may be followed by a FOR modifier.	SPACE 10 FOR CODE='Z'
TAB n Moves the print head backward or forward to position n on the line, where n may be a number or an expression. The statement may be followed by a FOR modifier.	TAB 3

Picture Formats

When describing a line to be printed, it is possible to instruct REPORT to print the data in a form different from that specified in the data description file. To do so, the user simply follows the field name with a picture enclosed in parentheses. For example, assume a field named SALES is defined in the data description file as:

SALES,N,.,7,2

If the value of SALES in the current record is 8732.79, the statement

```
PRINT SALES
```

prints the data as

```
8732.79
```

whereas the statement

```
PRINT SALES($$,$$$.DD)
```

or

```
PRINT SALES(2$,3$.2D)
```

instructs REPORT to print the data in SALES as:

```
$8,732.79
```

When the data exists in the correct and desired format to be printed, it is not necessary to specify a picture format.

The table below lists each picture character and a summary of its function. The discussions which follow detail each of the picture format characters with examples.

Picture Character	Prints
X	Single character, used with all character data.
D	One digit.
.	Decimal point.
V	No character is printed. V indicates the location of the decimal point.
Z	A blank for a leading zero.
*	Asterisk for a leading zero.
+	Plus sign or minus sign, as appropriate.
–	Blank or minus sign, as appropriate.
N	Blanks surrounding a positive number. Parentheses surrounding a negative number.
$	Dollar sign.
,	Commas. Blank when preceded by a leading zero or nondigit.

NOTE: Only the X format may be used for character data. The format for numeric data may be specified using the remaining picture characters.

The user may instruct the system to choose an appropriate format by following a field name with an empty picture format. For example, in conjunction with the SUM function in REPORT,[1] the user prints the sum of all values in the ACRES field with an empty picture format:

```
PRINT SUM ACRES( )
```

Regardless of the declared field length for ACRES, the system prints the data in free format, and the user avoids a possible error, such as asterisks printed instead of the number. The empty picture format is a convenient feature when data no longer fits in a field as it is declared in the data description. When used, the system chooses a legal format which prints numeric fields in free format; that is, it uses only as many characters as needed. Similarly, trailing blanks are truncated for character fields.

REPORT contains an impressive assortment of picture format characters to format and use symbols with the data to be printed. These features are discussed below.

The X Format

The X format is used for all character fields. Beginning with the left-most character in the field, one character is printed for each X in the picture format. For example, suppose the data in NAME contains HARTMAN. The statement

```
PRINT NAME(XXXXX)
```

prints the data as HARTM. Similarly,

```
PRINT NAME(XXXXXXXXX)
```

prints the data as HARTMAN and two trailing blanks.

The D Format

The D format must be used for all digits to the right of the decimal point; it may also be used to indicate any digit in the number. The D format prints one digit for each D in the format. For example,

```
PRINT 1.23(D.DDD)
```

or

```
PRINT 1.23(1D.3D)
```

prints the data as 1.230, whereas

```
PRINT 1.23(DDD.D)
```

or

```
PRINT 1.23(3D.1D)
```

prints the data as 001.2.

NOTE: When using a picture format to print a numeric field, digits to the right of the decimal point must always be indicated with the D format. The D format prints a negative number only if a sign is included in the picture format.

1 – The SUM function is discussed on page 147.

The . and V Formats

A single . or V indicates the location of the decimal point. The . prints the decimal point; the V indicates the location but suppresses the printing of the decimal point. For example,

```
PRINT 1.23(2D.3D)
```

prints the data as 01.230. On the other hand, the statement

```
PRINT 1.23(DDVDDD)
```

prints the data as 01230. Note that the decimal point functions but is not printed.

Assume a field, PAY, is to be written on a preprinted form containing a box for PAY with a line separating dollars and cents. In this situation, the user, not wanting the decimal point to be printed, types:

```
PRINT PAY(DDDVDD)
```

If, for example, PAY equals 179.50, REPORT prints the data as:

$\boxed{179|50}$

The Z and * Formats

The Z or * character, used repetitively in a format, indicates that blanks or asterisks, respectively, are to be printed instead of leading zeros. The Z format does not require the user to specify a sign as part of the format. The * format prints a negative number only if the sign is included in the picture format. For example, to print the value –11.66 with the * format, any of the following formats is valid:

```
–***.DD
+***.DD
```

The following examples further clarify the Z and * formats:

```
PRINT 1.23(***.DD)
```

or

```
PRINT 1.23(3*.2D)
```
(The 3*.2D specification is used to indicate three asterisks and two decimal places.)

prints the data as **1.23, and

```
PRINT 1.23(ZZZ.DD)
```

or

```
PRINT 1.23(3Z.2D)
```

prints the data as 1.23 with two leading blanks.

A Z format for a negative number must be large enough to accommodate the sign of the number. For example, the value –123.66 cannot be printed with the format 3Z.2D, because the field width of three characters to the left of the decimal point is insufficient for printing three digits and a leading minus sign. The correct format for this example is 4Z.2D.

The +, –, and N Formats

The +, –, and N characters instruct REPORT to indicate a negative or positive value of the field being printed. The + format causes a plus sign or a minus sign to be printed; the – format prints a blank or a minus sign; and the N format prints a positive number with one blank on either side and a negative number enclosed in parentheses. These characters are never used in combination with one another. For example, the following table shows legal formats; blanks are indicated with the letter b.

Format	Prints +1.23 As	Prints –1.23 As	Comments
ZZZ.DD+	bb1.23+	bb1.23–	A single + or – may be the first or last character of a format which also uses other characters to the left of the decimal point. When a + or – is the first character of a format, the format which follows must include a position for the sign to the left of the decimal point.
–ZZZ.DD	bbb1.23	bb–1.23	
++++.DD	bb+1.23	bb–1.23	The + or – may be used repetitively to print the sign just to the left of the first nonzero character.
– – – – .DD	bbb1.23	bb–1.23	
ZZZ.DDN	bbb1.23b	(bb1.23)	A single N must be used as the last character of the format.

NOTE: When the plus sign or the minus sign is used repetitively, it must appear in all positions to the left of the decimal point, except for dollar signs and commas in the format. See the $ and , formats below.

The $ Format

One or more dollar signs in a format print a single dollar sign with the value. The $ character may be used repetitively to place the dollar sign to the left of the first nonzero digit. For example,

PRINT 12.34($$$$.DD)

or

PRINT 12.34(4$.2D)

prints the value as $12.34. If not used repetitively, only one dollar sign may appear in the format; the dollar sign must be the first character of the format unless a sign is included. For example,

PRINT 12.34($ZZZ.DD)

or

PRINT 12.34($3Z.2D)

prints the data as $ 12.34.

A plus sign may be included with the dollar sign. The $ format character prints a negative number only if a sign is included in the picture format. For example, to print the value –11.66, any of the following formats is valid:

```
+$$$.DD
-$$$.DD
```

The , Format

The user may indicate commas to be printed with the data by including appropriately placed commas in the picture format. For example,

```
$ZZ,ZZZ.DDN
-$$,$$$,$$$.DD
+++,+++.DD
--,---.DD
```

are all legal formats. When a comma would be preceded by a leading zero or nondigit, the comma is suppressed. For example,

```
PRINT 123.45($$,$$$.DD)
```

prints the data as $123.45, whereas

```
PRINT 1234.56($$,$$$.DD)
```

prints the data as $1,234.56.

Additional Rules File Statements

In addition to the rules file statements presented on pages 32 through 41, REPORT provides three more statements: PRINT, ON, and general picture format declarations.

The PRINT Statement

The PRINT statement writes information on the REPORT output file. The user should enter PRINT statements for *all* information he wants to appear in the report produced.

The form of the PRINT statement is

PRINT item list

where any of the items on page 35 (except NCR) may appear in the item list, separated by commas. The item list may not be blank.

The PRINT statement does not include an automatic carriage return at the end of the item list. The user must, therefore, specify CR as the last item if a carriage return is desired. For example, the statements

```
10 PRINT "REPORT TITLE",3B
20 PRINT @DATE,CR
```

write on the REPORT output file the text REPORT TITLE, three blanks, and the current date on one line, and then return the carriage.

There are three basic differences between PRINT and TYPE statements.[1] The PRINT statement writes information on the REPORT output file; the TYPE statement prints information directly at the terminal. Lines that are written by a PRINT statement are included in the number of lines per report page; lines printed by a TYPE statement are *not* included in the number of lines per page. The PRINT statement does not generate an automatic carriage return; the TYPE statement generates an automatic carriage return at the end of the information printed.

The ON Statement

One or more ON statements control breaks in printing details from individual records. The ON statement may appear only in the TOTALS section of the rules file. ON statements are useful for specifying when subtotal information is to be printed.

The ON statement is described in detail with the TOTALS section on page 147.

The OFFHEADINGS and ONHEADINGS Statements

The OFFHEADINGS statement suppresses normal execution of the HEADINGS section at the top of the report page. The ONHEADINGS statement resumes normal execution of the HEADINGS section when the OFFHEADINGS statement was previously used.

Picture Format Declarations

An important feature of the DECLARE section allows the user to specify general picture formats for use throughout the rules file. Since many fields may require the same special format, the user can specify the format once in the DECLARE section and later refer to it by number.

This feature is detailed with the DECLARE section on page 140.

Structure of a REPORT Rules File

The REPORT rules file consists of six sections, providing the user with an efficient organization in which to detail his particular report requirements. A REPORT rules file may contain all or some of the sections, but the sections must appear in the following order:

DECLARE
INITIAL
HEADINGS
DETAILS
OTHERS
TOTALS
FINAL

The DECLARE section makes it possible to specify new fields to save data from records or intermediate calculations, and to specify general formats. The INITIAL section contains the report title, which is printed once at the beginning of the report. The INITIAL section may also assign initial values to declared fields or accept values from the terminal for specific fields.

1 – The TYPE statement is presented on page 34.

The HEADINGS section describes the page headings to be printed at the top of every page in the report. The DETAILS section specifies calculations to be performed on each record and the information to be printed from each record in the data file. In the TOTALS section, the user requests the various totals and subtotals desired. The FINAL section allows the user to print totals or other accumulated data after all records are processed.

The discussion which follows treats each REPORT section separately, specifying available capabilities, legal statements, and examples.

The DECLARE Section

The DECLARE section allows creation of working storage. It specifies new fields which do not exist in the data file but which can be used to save data from a record or calculation. For example,

```
10  DECLARE                        Section name must appear alone on a line.
20      CHEM.COUNT,I,3
30      ENGR.COUNT,I,3
40      SR.EMP,I,3
50      JR.EMP,I,3
60      YR.WAGE,R,9,2
```

creates temporary storage for five fields which do not appear in the input records. The user may declare as many fields as desired, the only limitation being that a maximum of 100 lines is permitted in a rules file.

The form used to specify a new field is:

field name, data type, field length, decimal places

The user may specify several field declarations on a single line by separating the field declarations with a semicolon (;). The following example illustrates declaring five working storage fields while entering only two lines in the DECLARE section.

```
10 DECLARE
20 TEMP1,N,5,2;TEMP2,N,5,2
30 MCOUNT,I,4;FCOUNT,I,4;TCOUNT,I,5
```

The field name may contain as many as 31 characters; must begin with a letter; and may be any combination of letters, the digits 0 through 9, the period (.), and the character @. See page 47 for a list of reserved field names.

The data type may be C, N, I, R, or D, indicating the contents of the field, as follows:

Data Type	Meaning
C	Character data
N	Numeric data
I	Integer number
R	Real number
D	Double precision number

NOTE: With declared fields, the data types I, R, and D do not indicate binary data, but rather specify the most efficient storage for an integer, real, and double precision number, respectively.

The field length specifies the maximum number of characters needed to write the data in symbolic form. The specification of the number of decimal places is necessary only for those fields that are to contain numbers with decimal points; it specifies the maximum number of digits to the right of the decimal point.

NOTE: Declared numeric fields are automatically initialized to zero; declared character fields are automatically initialized to blanks. Once a declared field is assigned a value, however, that value is used with subsequent records unless the value is specifically changed; if so, the new value is used with subsequent records, and so on.

The user may also declare as many as 20 general picture formats. With this feature, the user can specify the format once in the DECLARE section and later refer to it by number. For example, the user specifies three general formats:

```
10 DECLARE
20 FORMAT 1: $$$,$$$.DD
30 FORMAT 2: ----.DD
40 FORMAT 3: ZZZZZ
```

At a later point in the report description, he requests a special format by number:

```
170 PRINT NAME,S.S.NO,GROSS(1),DEDUCT(2),DOLLAR(3),NET(1)
```

The form for specifying general picture formats is

FORMAT n:picture format

where n may range from 1 to 20, inclusive.

See page 125 for an example of a DECLARE section specifying temporary storage fields and general formats in a complete rules file.

The ROUNDIN and ROUNDOUT statements may be used in the DECLARE section of the UPDATE, VERIFY, or REPORT rules files to round numeric data and/or arithmetic results to conform to the field descriptions.

The ROUNDIN statement rounds the values in all numeric fields to their specific field descriptions before the values are used in an arithmetic calculation.

The ROUNDOUT statement rounds numeric values and arithmetic results to conform to the receiving field descriptions. To preserve accuracy in calculations, intermediate results of an expression are not rounded. The initial values are rounded when the ROUNDIN statement appears; the final values are rounded when the ROUNDOUT statement appears.

For example, assume the following field descriptions for a binary file:

```
FLD1,R,6,2
FLD2,R,6,2
FLD3,R,6,2
```

Suppose the statement

```
FLD1 + FLD2 TO FLD3
```

appears in the rules file, and the stored values for FLD1 and FLD2 are 212.35499999 and 201.35499999. If only a ROUNDIN statement appears in the rules file, FLD1 and FLD2 are rounded, and the value of FLD3 is 413.70. If only a ROUNDOUT statement appears in the rules file, FLD1 and FLD2 are added, and the value of FLD3 is rounded to 413.71. If both ROUNDIN and ROUNDOUT statements appear in the rules file, FLD1 and FLD2 are rounded and summed, and the value of FLD3 is rounded to 413.70.

IML provides rounding to as many as five decimal places. Values from or to fields specifying more than five decimal places are not rounded. Users who want more accuracy can obtain it, despite any rounding statements, by specifying more than five decimal places in the appropriate field descriptions. The function used by IML to perform the rounding can be described by the equation

$$\mathrm{IP}(\text{value} * 10^N + .5)/10^N$$

where IP means integer part, and N is the number of decimal places specified.

The INITIAL Section

The INITIAL section is executed once at the beginning of the report procedure before any records are processed. Here, the user specifies the title for the entire report, as well as the initial values for declared fields and built-in functions. For example:

```
80   INITIAL
86      5 TO MARGIN
90      SKIP TO 3
100     PRINT 15B,"PERSONNEL SUMMARY REPORT",CR,15B,@WEEKDAY,4B,
         @CDATE,CR
```

Line 86 sets the bottom margin to five blank lines; line 90 skips to the third line of the report page; and line 100 specifies the report title.

The INPUT statement is very useful in the INITIAL section for entering a control value for a single field directly at the terminal. For example:

```
50 INITIAL
60 TYPE "PLEASE ENTER A REPORT CODE OF 1,2, OR 3"
70 INPUT RPT.CODE
```

Line 60 prints the comments in quote marks directly at the terminal, and line 70 accepts the entered value for RPT.CODE. RPT.CODE must be a new field specified in the DECLARE section.

All the valid control and instruction statements may appear in this section. Data field names, however, may not be specified, since the INITIAL section is executed before any data records are processed.

See page 125 for an example of an INITIAL section in a complete rules file.

The HEADINGS Section

The HEADINGS section is executed whenever REPORT begins a page of the report. A SKIP TO TOP statement, PRINT TOF statement, or page completion, therefore, causes the execution of the entire HEADINGS section. The HEADINGS section specifies the information to be printed at the top of every page in the report. For example:

```
130 HEADINGS
135     PRINT PAGE,CR,"PERSONNEL SUMMARY",CR,@DATE,CR
140     SKIP 6
150     PRINT " NAME    NUMBER DEPT  JOB    SALARY    RATING",
         "  PROMOTABILITY",CR,CR
```

Line 135 specifies a page title, including the data and page number; line 150 specifies the column headings for the details which follow.

In addition, the user may specify control statements and assign values to declared fields in the HEADINGS section. Data record fields, however, may not appear in the HEADINGS section.

See page 125 for an example of a HEADINGS section used in a complete rules file.

The DETAILS Section

The DETAILS section is executed for each record in the file. The user describes any calculations required and the information to be printed for each record. Information from a record to be used for a later calculation may be saved in a temporary storage field created in the DECLARE section. For example:

```
160 DETAILS
170     IF "CHEM" IN AREA OR AREA2
175        CHEM.COUNT +1 TO CHEM.COUNT
180     IF "ENGR" IN AREA OR AREA2
185        ENGR.COUNT +1 TO ENGR.COUNT
190     IF DATE.HIRE<700101
195        SR.EMP +1 TO SR.EMP
200     ELSE JR.EMP +1 TO JR.EMP
210     IF SALARY<25.
215        SALARY*2080 TO YR.WAGE
220     ELSE SALARY TO YR.WAGE
230     DO
240     PRINT NAME,EMP.NO,3B,DEPT,3B,JOB,4B,YR.WAGE,5B,RATING,
         6B,PROMOTABILITY,CR
```

Lines 170 through 200 cause a continuous calculation of the number of employees with a background in chemistry, the number of employees with a background in engineering, those employed before January 1, 1970, and those employed since January 1, 1970, respectively. The counts are stored in four temporary fields created in the DECLARE section. Lines 210 through 220 compute the yearly salary. If the salary in the current record is less than 25, it is at an hourly rate (no one earns more than $25 per hour) and is converted to an annual rate; otherwise, the given salary is already an annual one. Line 240 specifies the details to be printed from each record.

When a function (SUM, AVG, MIN, and MAX), described below, applies to a data field, the field values in the record are always used in the evaluation; that is, any calculations specified in the DETAILS section do not affect the function result. If the function evaluation is to be affected by the rules in the DETAILS section, declared fields must be used.

If the user wants a declared field to assume an initial value in each record processed by the DETAILS section, he must specifically assign it a value at the beginning of the DETAILS section.

See page 125 for an example of a DETAILS section used in a complete rules file.

The OTHERS Section

The OTHERS rules section permits the user to perform operations with records which were excluded from the report by an IF or FOR clause in the REPORT command.

The OTHERS section is similar to the DETAILS section; however, the OTHERS section processes each record that is excluded by the IF or FOR clause specified in the REPORT command, whereas the DETAILS section processes each record that is qualified by the IF or FOR clause. For example, assume that the user enters the REPORT command shown below:

```
–REPORT PROD:PRDESC IF DAY NOT = 0 TO T AS PER PRULES↵
```

Each record of PROD which does not specify DAY as zero is processed by the DETAILS section; each record of PROD which does specify DAY as zero is processed by the OTHERS section.

NOTE: Records which are processed by the OTHERS section do not affect the cumulative functions SUM, MIN, MAX, and AVG.

The example below illustrates the use of the OTHERS section in generating a report on the production of parts. The user wants the report to show, for each part, the quantity made per day and the percentage that quantity is of the week's production of that part.

The data file PROD contains records with four fields: PARTNO, WEEK, DAY, and QTY. The records specifying DAY as zero contain the total quantity of that part produced for the given week, whereas records specifying DAY as 1, 2, 3, 4, or 5 contain the quantity of that part produced for the day. Before entering the REPORT command, the user sorts the data file by PARTNO, WEEK, and DAY.

The sorted data file PROD and its description file, PRDESC, are shown below.

```
-TYPE PROD↵

 10   3   0   100
 10   3   1    10
 10   3   2    15
```

```
10  3  3   25
10  3  4   40
10  3  5   10
20  3  0  150
20  3  1   20
20  3  2   25
20  3  3   30
20  3  4   35
20  3  5   40

-TYPE PRDESC⤸

$$$
***BIN OR SYM:
S
***FIX OR VAR:
V
***WHICH TYPE:
L
***NUMBER OF LINES PER RECORD IS:
1
***  NAME, TYPE, START, LENGTH, DECIMAL PLACES
PARTNO,N,1,3
WEEK,N,.,3
DAY,N,.,3
QTY,N,.,5

-
```

The REPORT command procedure is shown below.

```
-REPORT PROD:PRDESC IF DAY NOT = 0 TO PRODRPT AS PER PRULES⤸
: LIST⤸                        The rules are listed for the purpose of this example.
100 DECLARE
120     TTL.QTY,N,5            This temporary field is to contain the value of QTY when DAY is zero.
130     PERCENT,N,5,2
170 INITIAL
193     SKIP 5
195     PRINT "SUMMARY REPORT, ",@WEEKDAY(),", ",@CDATE,2B,@CTIME
200     PRINT CR,CR,CR
210 DETAILS                    When the value of DAY is not zero, the record is processed in this section.
230     IF DAY = 1
240         PRINT "PRODUCTION OF PART ",PARTNO()," FOR WEEK ",WEEK()
245         PRINT CR,CR
250         PRINT 4B,"DAY",3B,"UNITS",3B,"PERCENT",CR
270     DO 100*QTY/TTL.QTY TO PERCENT
280     PRINT 4B,DAY,3B,QTY,4B,PERCENT,"%",CR
330 OTHERS                     When the value of DAY is zero, the record is processed in this section.
```

```
350     QTY TO TTL.QTY
370 TOTALS
390     ON PARTNO
395           PRINT 12B,13"-",CR
400           PRINT 4B,"TOTAL:",SUM QTY,3B,SUM PERCENT(ZZZ.DD),"%"
410           PRINT CR,CR,CR
: RUN
PRODRPT.. NEW FILE


OK.





REPORT FINISHED.


-
```

The report produced is shown below.

```
-TYPE PRODRPT

SUMMARY REPORT, TUESDAY, FEB. 6,1974   4:55 PM


PRODUCTION OF PART 10 FOR WEEK 3

      DAY    UNITS    PERCENT
        1       10     10.00%
        2       15     15.00%
        3       25     25.00%
        4       40     40.00%
        5       10     10.00%
                -------------
     TOTAL:    100    100.00%

PRODUCTION OF PART 20 FOR WEEK 3


      DAY    UNITS    PERCENT
        1       20     13.33%
        2       25     16.67%
        3       30     20.00%
        4       35     23.33%
        5       40     26.67%
                -------------
     TOTAL     150    100.00%


-
```

The TOTALS Section

In the TOTALS section, the user requests the various subtotals desired. The user indicates when the subtotal information is to be printed by including one or more ON statements of the form

ON field list

in the TOTALS section. *NOTE: At least one ON statement must be used if the TOTALS section appears in a rules file.* Only the data record fields may appear in an ON statement. REPORT prints the subtotal information based on a change in the value of a specified field. For example, the statement

ON STORE,SALESMAN

indicates that subtotals are to be printed whenever STORE or SALESMAN changes in the next record.

A PRINT statement may appear on a subsequent line specifying the information to be printed. Four functions are available in the TOTALS section to assist the user in describing the subtotal information desired:

Subtotal Function	**Computes**
SUM	Sum of all values in a given field.
AVG	Average value of a given field; average value is computed by counting the number of records, adding the values, and dividing the sum by the count.
MIN	Minimum value in a given field.
MAX	Maximum value in a given field.

For example, the user wishes to print subtotal information whenever the value in DEPT changes. For each department, he wishes to print the sum of the salaries and the average salary. He accomplishes this as follows:

```
250 TOTALS
260    ON DEPT
265    SKIP 1
270    PRINT 8B,"*TOTAL SALARY*",5B,SUM YR.WAGE(1)
280    PRINT 3B,"AVERAGE ",2B,AVG YR.WAGE(1),CR,CR,CR
285    IF LINE>55
287       SKIP TO TOP
```

Line 260 specifies when to print the information, that is, when the value of DEPT changes. Lines 265 through 280 specify what to print; that is, REPORT prints a blank line, then prints the sum and average salary on the same line. Lines 285 and 287 control the paging after a subtotal is printed.

The user may include several ON statements, each with a corresponding PRINT statement. The maximum number of fields allowed in *all* ON statements, however, is 20 fields. Whenever a field specified in an ON statement changes value, *all preceding* ON statements and corresponding instructions are executed in numerical order. For example:

```
500 TOTALS
510     ON STORE,SALESMAN
520       SKIP 1
530       PRINT '***',14B,SUM AMT,SUM UNITS,CR
540     ON DIST
550       PRINT '**',15B,SUM AMT,SUM UNITS,CR
560     ON DIV
570       PRINT '*',16B,SUM AMT,SUM UNITS,CR
580       SKIP TO TOP
```

When STORE or SALESMAN changes, lines 510 through 530 are executed; when DIST changes, lines 510 through 550 are executed; and when DIV changes, lines 510 through 580 are executed. Note that when DIV changes, all subtotals are printed and the next DIV begins on a new page, as per line 580.

ON statements may control breaks in printing individual records for executing any instruction statements. That is, it is not necessary to specify a PRINT statement after an ON statement. For example:

```
100 TOTALS
110 ON CLASS
120 SKIP TO TOP
```

The TOTALS section above instructs REPORT to skip to a new page for each class.

In addition, the user may specify control statements, arithmetic operations, and/or declared fields following an ON statement. For example:

```
100 TOTALS
110 ON CLASS
120 SUM QTY*.175 TO BASE
130 PRINT "QUANTITY",QTY,"BASE",BASE
140 SKIP TO TOP FOR LINE>55
```

A few rules are appropriate at this point to conclude the discussion of ON statements.

The user may specify only fields in the data records; he may not specify a temporary storage field name in an ON statement.

The user may precede a field name in an ON statement with a minus sign (–) to indicate reverse order. For example,

```
ON –DIV
```

indicates that the divisions appear in the data base in reverse order, such as 3 first, then 2, then 1. With character fields, the minus sign indicates reverse alphabetical order.

REPORT performs a sequence check to determine that the records in the data file are sorted on the fields named in the ON statements of the TOTALS section. The user must, therefore, sort his data file before entering the REPORT command.

The last field list specified in all ON statements must appear first in the SORT command, the next to last field list must appear next in the SORT command, and so on.[1] Note that in a single ON statement field list, those field names must appear in the same order within the SORT command. For example, to sort a file in preparation for the TOTALS section (lines 500 through 580) on page 148, the appropriate SORT command is:

```
–SORT SALESDATA:SDESC TO RPTSALES BY DIV,DIST,STORE,SALESMAN↵
```

NOTE: If a minus sign precedes a field name in an ON statement, it must also precede that field name in the SORT command.

The following statements are used to illustrate another SORT command in preparation for the TOTALS section.

```
500 TOTALS
510 ON A,B,C
 .
 .
 .
550 ON D
 .
 .
 .
590 ON E,–F,G
 .
 .
 .
```

The appropriate SORT command is:

```
–SORT FILEX:DESC TO FILEY BY E,–F,G,D,A,B,C↵
```

See page 125 for an example of a TOTALS section used in a complete rules file.

The FINAL Section

After all records are processed, REPORT executes the FINAL section. It is here that the user specifies totals or other accumulated calculations to be printed. All the functions available in the TOTALS section are also valid in the FINAL section. These functions, SUM, AVG, MIN, and MAX, may be used with data fields or temporary fields created in the DECLARE section. For example:

```
290 FINAL
300    PRINT CR,CR
301    PRINT "FINAL TOTALS",CR,"-----------------------------------
-------------------------",CR,CR
310    PRINT 8B,"*TOTAL PAYROLL*",4B,SUM YR.WAGE(1),CR
312    PRINT CR,"EMPLOYEE CHARACTERISTICS:",CR,CR
320    PRINT "SENIOR EMPLOYEES ",SR.EMP,CR
330    PRINT "JUNIOR EMPLOYEES ",JR.EMP,CR
340    PRINT "EMPLOYEES WITH CHEMISTRY BACKGROUND ",2B,CHEM.COUNT,CR
350    PRINT "EMPLOYEES WITH ENGINEERING BACKGROUND ",ENGR.COUNT,CR
355    SKIP TO 61
```

Line 310 prints the sum of all values in YR.WAGE; lines 320 through 350 print the accumulated data for the counts on senior employees, junior employees, employees with chemistry backgrounds, and those with engineering backgrounds, respectively.

See page 125 for an example of a FINAL section used in a complete rules file.

1 – See page 83 for documentation of sorting procedures.

SECTION 7
ADVANCED APPLICATIONS

This section presents a variety of advanced capabilities in the Information Management Library. Some of the discussions introduce additional options and commands in IML; others detail the use of the EXECUTIVE or language subsystems in combination with IML.

This section by no means exhausts the advanced or indirect capabilities present in the Information Management Library. The section discusses the more popular applications and features not documented elsewhere in the manual. It is hoped that the information in this section serves not as an encyclopedia but rather as an indication of the potential applications handled with IML.

The first discussion details the important reformatting facility of the SELECT command, allowing the user to change the structure of a data file, that is, to rearrange or eliminate fields in the records of a data file. Special attention is given to reformatting RETRIEVE files, although this capability is equally applicable to IML data files.

Next, a new command, APPEND, for appending a file or data entered at the terminal to another file, is introduced. Regardless of the structure of the files, APPEND allows the user to append a description file, data file, or any other file to an existing or new file.

The next discussion presents an important concept for conserving storage of description files and rules files: the multiple description file. The purpose, creation, and use of multiple description files are detailed, and a host of additional options in DEFINE are presented, allowing the user to list, rename, delete and edit the individual data file descriptions and rules sets on a single multiple description file.

The last discussion explores the use of command files with IML. It includes the creation of simple command files as well as complicated command files which originate from a language subsystem. Also presented are additional rules statements for command file control and the use of CREATE in a command file with data entry from the terminal.

REFORMATTING A RETRIEVE DATA BASE

The SELECT command provides RETRIEVE users with an important capability to reformat the information in symbolic or binary data bases.[1] Using SELECT's optional output file specification, the user may change the structure of records in a RETRIEVE file to allow additional fields, to eliminate fields, and to rearrange the order of fields. Furthermore, the user may include an optional IF clause in the same command to delete entire records from the data base being reformatted.

1 – See pages 87 through 96 for complete documentation of SELECT.

The basic form of the SELECT command used to reformat records in a RETRIEVE data base is

```
                                   output
-SELECT  FROM  data  base  TO  file      :  field list↵
                                   name
```

where the field list specifies the desired fields and their arrangement in the records to be written. The user may include blanks, carriage returns, line feeds, and literals in specifying the output file format.

NOTE: When reformatting a RETRIEVE binary file, the user should remember that character fields must be a multiple of 3. When adding literals and blanks, therefore, the text in quote marks and/or blanks must be a multiple of 3.

To specify blanks, the user simply types the number of blanks desired, followed by B, at the appropriate position in the field list. For example, the field list

FLD1,9B,FLD2

specifies that nine blanks are to be written after the field FLD1.

To specify a carriage return or a line feed, the user types CR or LF, respectively, at the appropriate position in the field list. For example, the field list

FLD1,9B,FLD2,LF,FLD3,FLD4,CR

specifies that a line feed is to be written after FLD2, and a carriage return is to appear at the end of each record.

NOTE: SELECT does not include an automatic carriage return at the end of each record when writing the output on a file. The user must, therefore, specify CR at the end of the field list when he wants a carriage return to be written.

To specify literals, that is, text to appear in each record, the user enters the text enclosed in single or double quote marks at the appropriate position in the field list. For example, the format

FLD1,"TEXT",FLD2

specifies that the characters TEXT are to immediately follow FLD1 in each record.

NOTE: The user may specify as many as 80 characters of text enclosed in quote marks and as many as 80 blanks for any position in the field list.

To select as well as reformat records, the user includes an IF clause in the SELECT command.[1] The form used is:

```
                                                          output
-SELECT  FROM  data  base  { IF  conditions  }  TO  file      :  field list↵
                           { FOR conditions  }       name
```

NOTE: The user must create a new description or structure file by calling DEFINE or RETRIEVE when he wants to use the reformatted data base in a subsequent command.

The sample applications reformat a RETRIEVE data base named INVEN. The data base and its structure file are shown on the following page.[2]

1 – See page 20 for a discussion of IF clauses and page 89 for the use of IF clauses with the SELECT command.

2 – Note that RECNO is not a field in the records, but is printed automatically with listings in RETRIEVE.

```
-RETRIEVE

.BASE INVEN

24 RECORDS(38)

.LIST

 RECNO        NAME                     QTY  DAT CLASS
     NUMBER

     1    28 CAL MANUAL               5400 6906 IX
     2    31 COGO MANUAL               675 6901 IX
     3    32 CSMP MANUAL             12420 7205 IX
     4    34 CUC MANUAL               2295 7108 IX
     5   100 DEBUG MANUAL             4500 7201 X
     6    35 EASYPLOT MANUAL          5292 7007 IX
     7     3 EDITOR MANUAL            2610 6907 IX
     8     1 EXECUTIVE MANUAL         5850 7101 IX
     9    38 FORTUNE 500 MANUAL      10800 7005 IX
    10     7 IML MANUAL               6993 7109 IX
    11    24 LAPLACE MANUAL           4212 7002 IX
    12    40 LNED MANUAL              2097 7202 IX
    13    41 LOGSIM MANUAL           18387 7002 IX
    14   106 MINMAX MANUAL            9639 7111 X
    15    25 NASAP MANUAL             3402 7006 IX
    16    12 RETRIEVE MANUAL         23859 7108 IX
    17    48 STATPAK MANUAL           5400 7103 IX
    18   108 STATPAK MANUAL           5904 7202 X
    19     8 SUPER BASIC MANUAL      21600 7105 IX
    20     6 SUPER FORTRAN MANUAL    12798 7004 IX
    21   111 TYMEX MANUAL            15147 7203 X
    22    13 TYMTAB MANUAL           45000 7206 IX
    23    49 TYMTRAC MANUAL              0 7110 IX
    24    50 TYMUSE MANUAL             270 7003 IX

24 RECORDS

.STRUCTURE


FIELD   TYPE  WIDTH  NAME

    1   I       4    NUMBER
    2   C      21    NAME
    3   I       6    QTY
    4   I       4    DAT
    5   C       2    CLASS
```

The examples below illustrate the use of SELECT to reformat the INVEN data base in three different ways. The first example shows the elimination and rearrangement of fields in each record of INVEN. Example 2 reformats INVEN by adding one field and leaving space for another field. The final example illustrates the selection and simultaneous reformat of records in INVEN.

Example 1

The user wants to omit the QTY and DAT fields, and also wants the records to contain CLASS first, then NAME, and last, NUMBER.

```
-SELECT FROM INVEN TO IDMANUALS:CLASS,NAME,NUMBER,CR
IDMANUALS.. NEW FILE
OK.

24 RECORDS SELECTED FROM 24

-TYPE IDMANUALS

IXCAL MANUAL               28
IXCOGO MANUAL              31
IXCSMP MANUAL              32
IXCUC MANUAL               34
X DEBUG MANUAL            100
IXEASYPLOT MANUAL          35
IXEDITOR MANUAL             3
IXEXECUTIVE MANUAL          1
IXFORTUNE 500 MANUAL       38
IXIML MANUAL                7
IXLAPLACE MANUAL           24
IXLNED MANUAL              40
IXLOGSIM MANUAL            41
X MINMAX MANUAL           106
IXNASAP MANUAL             25
IXRETRIEVE MANUAL          12
IXSTATPAK MANUAL           48
X STATPAK MANUAL          108
IXSUPER BASIC MANUAL        8
IXSUPER FORTRAN MANUAL      6
X TYMEX MANUAL            111
IXTYMTAB MANUAL            13
IXTYMTRAC MANUAL           49
IXTYMUSE MANUAL            50

-
```

The user specifies a carriage return (CR) at the end of each record.

Note that the user may easily insert spaces between the fields when he does not want the fields to be contiguous. For example,

```
-SELECT FROM INVEN TO IDMANUALS:CLASS,2B,NAME,2B,NUMBER,CR ↵
```

specifies that two blanks are written after CLASS and NAME. This capability is also illustrated in the next example.

Example 2

The user wants to add the characters TYMCOM– before the CLASS field, and also wants to leave space for a five-character field, which is to contain data from a later operation, between the QTY and DAT fields.

```
-SELECT FROM INVEN TO NEWINV:NUMBER,NAME,QTY,5B,DAT,"TYMCOM-",CLASS,CR ↵
NEWINV.. NEW FILE↵

OK.


24 RECORDS SELECTED FROM 24

-TYPE NEWINV↵

  28CAL MANUAL            5400     6906TYMCOM-IX
  31COGO MANUAL            675     6901TYMCOM-IX
  32CSMP MANUAL          12420     7205TYMCOM-IX
  34CUC MANUAL            2295     7108TYMCOM-IX
 100DEBUG MANUAL          4500     7201TYMCOM-X
  35EASYPLOT MANUAL       5292     7007TYMCOM-IX
   3EDITOR MANUAL         2610     6907TYMCOM-IX
   1EXECUTIVE MANUAL      5850     7101TYMCOM-IX
  38FORTUNE 500 MANUAL   10800     7005TYMCOM-IX
   7IML MANUAL            6993     7109TYMCOM-IX
  24LAPLACE MANUAL        4212     7002TYMCOM-IX
  40LNED MANUAL           2097     7202TYMCOM-IX
  41LOGSIM MANUAL        18387     7002TYMCOM-IX
 106MINMAX MANUAL         9639     7111TYMCOM-X
  25NASAP MANUAL          3402     7006TYMCOM-IX
  12RETRIEVE MANUAL      23859     7108TYMCOM-IX
  48STATPAK MANUAL        5400     7103TYMCOM-IX
 108STATPAK MANUAL        5904     7202TYMCOM-X
   8SUPER BASIC MANUAL   21600     7105TYMCOM-IX
   6SUPER FORTRAN MANUAL 12798     7004TYMCOM-IX
 111TYMEX MANUAL         15147     7203TYMCOM-X
  13TYMTAB MANUAL        45000     7206TYMCOM-IX
  49TYMTRAC MANUAL           0     7110TYMCOM-IX
  50TYMUSE MANUAL          270     7003TYMCOM-IX

-
```

Example 3

The user wants to select records for which CLASS equals X, and also wants to reformat the records to contain the NAME field first, then the NUMBER field preceded by the character #, then the QTY field, three blanks, and the DAT field.

```
-SELECT FROM INVEN IF CLASS="X" TO INVTYMCOMX:NAME,"#",NUMBER,¬
QTY,3B,DAT,CR ↵
INVTYMCOMX.. NEW FILE↵

OK.


4 RECORDS SELECTED FROM 24

-TYPE INVTYMCOMX↵

DEBUG MANUAL           # 100  4500   7201
MINMAX MANUAL          # 106  9639   7111
STATPAK MANUAL         # 108  5904   7202
TYMEX MANUAL           # 111 15147   7203

-
```

SAVING A COMPILED RULES FILE PROCEDURE

Once the user is satisfied that no further changes are to be made to a rules file procedure, he may use the SAVE PRODUCTION command to direct IML to compile the rules file, the description file(s), the IF or FOR clause(s), the key field list, and the file names used with the IML command. SAVE PRODUCTION compiles and saves a more efficient form of the entire procedure on a user-specified file for future execution with the PERFORM command.

The user enters the SAVE PRODUCTION command at rules file command level instead of entering the RUN command. The form of the SAVE PRODUCTION command is:

```
                             production
: SAVE {PRODUCTION}   ON     file        ↵
       {PRO       }          name
```

It is suggested that the user save a copy of the current rules file before using the SAVE PRODUCTION command. A listing of the rules cannot be reconstructed from the production file. The user is asked for the name of the rules file to ensure that any changes are incorporated in the file before the production file is created. The following example illustrates the creation of a production file with the SAVE PRODUCTION command.

```
-REPORT SPERSONNEL:PDESC IF @NDATE-LAST.REV>=9900 TO REVDUE ¬
AS PER DPTRPT↵
: SAVE PRODUCTION ON PRDFIL↵
REVDUE.. NEW FILE↵
PRDFIL.. NEW FILE↵

PRODUCTION SAVED.

-
```

The user enters the IML command procedure to be saved.

Instead of entering the RUN command, he uses the SAVE PRODUCTION command.

He confirms the report output file name and the production file name.

Control returns to the EXECUTIVE after the production file is written.

Executing a Compiled Rules File Procedure

The PERFORM command executes a compiled, saved production rules file procedure directly in the EXECUTIVE. In addition to two options which print the file names and the version of the IML command associated with the production file, several alternative forms of the PERFORM command allow the user to respecify one or more file names of the production file.

The basic form of the PERFORM command is:

```
         ([FILES]  )  production
–PERFORM {         }  file       ↩
         ([VERSION])  name
```

This form of the PERFORM command allows the user to display the command name and the file name(s) established in the production file or the name and version number associated with the production file. The user must specify the production file name used with the SAVE PRODUCTION command when using the PERFORM command.

For example, suppose the user wants to execute the REPORT command procedure shown on page 156. The compiled form of this procedure is saved on the file PRDFIL. The user simply types

```
–PERFORM PRDFIL↩
REVDUE.. OLD FILE↩
```

PERFORM lets the user confirm or change the established output file name.

which is considerably more efficient than the equivalent, longer procedure shown below:

```
-REPORT SPERSONNEL:PDESC IF @NDATE-LAST.REV>=9900 TO REVDUE
AS PER DPTRPT↩
: RUN↩
REVDUE.. OLD FILE↩

OK.

•

•

•
```

The FILES option prints the command name and the file name(s) established in a production file.

```
-PERFORM [FILES] PRDFIL↩
REPORT SPERSONNEL TO REVDUE


-PERFORM [FILES] PRDUPD↩
UPDATE MASTER WITH JULACT TO CURRENT


-
```

The words TO and WITH are included in the response to identify the data, activity, and output files.

To print the name and version number of the IML command associated with a production file, the user types:

```
-PERFORM [VERSION] PRDFIL
REPORT EO6.00

-PERFORM [VERSION] PRDUPD
UPDATE EO6.00

-
```

CHANGING FILE NAMES

Several alternative forms of the PERFORM command allow the user to change the data, activity, and output file names that were previously established in the production file.[1] Most of the forms documented below require that the user respecify the same number and type (binary or symbolic) of files as were used when the production file was created.[2] The particular form chosen by the user depends on the IML command (REPORT, CONVERT, VERIFY, or UPDATE) and the number of files established in the production file.

If the production file uses the REPORT command, the CONVERT command, the VERIFY command with an output file, or the UPDATE command without an activity file, the user respecifies file names and executes the compiled procedure with the command:

```
          production  data       output
-PERFORM  file        file   TO  file
          name        name       name
```

For example, if the user wishes to execute the production file PRDFIL and respecify the file names used in this procedure, he types:

```
-PERFORM PRDFIL NEWDATA TO NEWRPT
NEWRPT.. NEW FILE
```

The established description file, IF clause, and rules file are always used in the production execution. The above PERFORM command is equivalent to and more efficient than the following procedure:

```
-REPORT NEWDATA:PDESC IF @NDATE-LAST.REV>=9900 TO NEWRPT
AS PER DPTRPT
: RUN
NEWPRT.. NEW FILE

OK.

.

.

.
```

If the production file uses the VERIFY command without an output file, the user may respecify the data file name and execute the compiled procedure with the command:

```
          production  data
-PERFORM  file        file
          name        name
```

1 – If the user wants to change only the output file name, a special form of the PERFORM command is not necessary; he may simply type an alt mode/escape in response to the NEW/OLD FILE message and respecify the output file name at that point.

2 – The exception to respecifying the same number of files is noted on page 159.

If the production file uses the UPDATE command with an activity file, the user respecifies file names and executes the compiled procedure with the command:

```
          production  data         activity    output
-PERFORM  file        file   WITH  file    TO  file   ↵
          name        name         name        name
```

For example, if the production file PRDUPD was created after the user entered the command

```
-UPDATE MASTER:MDESC WITH JULACT:ADESC TO CURRENT ¬
IF INVOICE:A=INVOICE:D BY ACCOUNT AS PER UPDRUL↵
```

then the command

```
-PERFORM PRDUPD CURRENT WITH AUGACT TO NEWMASTER↵
NEW MASTER.. NEW FILE↵
```

is equivalent to

```
-UPDATE CURRENT:MDESC WITH AUGACT:ADESC TO NEWMASTER ¬
IF INVOICE:A=INVOICE:D BY ACCOUNT AS PER UPDRUL↵
: RUN↵
NEWMASTER.. NEW FILE↵

OK.
.

.

.
```

If the user simply types

```
-PERFORM↵
```

PERFORM only prompts for the production file name. The user may, however, also respecify file names on the same line with the requested production file name. For example:

```
-PERFORM↵
PRODUCTION NAME: PRDUPD CURRENT WITH AUGACT TO NEWMASTER↵
```

NOTE: If the user respecifies file names but does not enter a name for every file, the file name(s) established in the production file is used for the unspecified file name(s).

APPENDING A FILE TO ANOTHER FILE

The APPEND command permits the user to append one file to another file in an efficient manner, requiring only that both files be the same type, either binary or symbolic. The form of the APPEND command is

-APPEND file name$_1$ TO file name$_2$↵

where only file name$_1$ may be T (for terminal); file name$_2$ may be a new file or an existing file; and neither file name may be NOTHING.

NOTE: Neither file need be an IML file.

The user may type simply

```
-APPEND↵
```

and the system prompts:

```
FILE TO BE APPENDED:
APPEND TO FILE:
```

For example, the command

```
-APPEND FILE1 TO FILE2↵
```

adds the contents of FILE1 to the end of FILE2. After the command is executed, FILE2 contains the original contents of FILE2 plus the contents of FILE1.

The user may append data from the terminal to a specified file by entering an APPEND command of the form

```
-APPEND T TO file name↵
```

where the file named may be a new file or an existing file. For example:

```
-APPEND T TO SDATA↵
```

When appending data from the terminal, the full set of editing control characters may be used, as in the APPEND mode of EDITOR.[1] The user terminates the APPEND command with a Control D as the first character of a line when entering data from the terminal.

When Control Q deletes the line being typed, it echoes as an up arrow (↑). When Control Q deletes the preceding complete line, it echoes as an up arrow and a crosshatch (↑#). Control Q may be used repeatedly to delete several lines. Alternatively, the user may actually type an up arrow (Shift N), followed by a carriage return at the beginning of a line, to delete the preceding line. The system prompts with the number of the line to be entered next and returns the carriage. An up arrow followed by a carriage return may be used repeatedly to delete several preceding lines. For example:

```
-APPEND T TO CREDITS↵
CREDITS.. NEW FILE↵
OK.
A-S-48↵
R-C-62↵
D-O-76↵
E-T-12↵
N-T-74↵
↑↵              The user deletes the preceding line.
5:              The system prompts for the next line, namely, the fifth line.
↑↵              The user deletes another line.
4:              The system prompts for the next line, which is now the fourth line.
T-F-87↵
H-L-22↵
A-O-18↵
D-R-83↵
D-Y-56↵
E-A-09↵
```

1 – See the *Tymshare EDITOR Reference Manual* for complete documentation of control characters and their functions.

```
U-N-41
S- -22
↑
11:
FINISHED.
```
The user types Control D at the beginning of the line, terminating the APPEND procedure.

```
-
```

USING MULTIPLE DESCRIPTION FILES

The DEFINE program incorporates several additional capabilities which allow the user to reduce significantly the storage required for description files and rules files. The user may store more than one data file description and set of rules on a single file, calling DEFINE to list, edit, rename, or delete specific information on a multiple description file.

The discussion of multiple description files is organized in several parts. First, the technique for creating a multiple description file is discussed, including data file descriptions and rules sets. Next, the numerous DEFINE options for facilitating the use of multiple description files are presented. The discussion includes the [DIR], [LIST], [DELETE], [RENAME], and [EDIT] options, together with their various forms and purposes.

Creating a Multiple Description File

This discussion presents the method of creating a data file description as part of a multiple description file and the method of creating a rules file to be included in a multiple description file.

Data File Descriptions

The user creates a data file description as part of a multiple description file, using a special form of the DEFINE command. The form used is

```
        multiple
-DEFINE description(title)
        file
        name
```

where the specified title must conform to the rules for naming files.[1] For example, the command

```
-DEFINE MULT(INVDES)
```

creates a data file description titled INVDES on a file named MULT. After the user enters the appropriate information, the system writes the data file description on MULT with the heading:

$$$FIL(INVDES)

The following example illustrates the creation of two data file descriptions to appear in the same multiple description file.

1 – See the *Tymshare TYMCOM-IX EXECUTIVE Reference Manual* for details on file naming conventions.

```
-DEFINE MULT(WAGE)↵
MULT.. NEW FILE↵          The user creates a new multiple description file.
SHORT PROMPTS? YES↵
BIN OR SYM: SYM↵
FIX OR VAR: VAR↵
VAR TYPE(C,L,S): WHICH TYPE: L↵
NUMBER OF LINES PER RECORD IS:1↵
  NAME, TYPE, START, LENGTH, DECIMAL PLACES
: EMP.NO,N,1,4↵
: RATE,C,6,1↵
: HOURS,N,8,2↵
: WAGES,N,11,6,2↵
: ↵
DESCRIPTION FILE WRITTEN.

-DEFINE MULT(TERMIN)↵
MULT.. OLD FILE↵          The description is to be appended to MULT.
SHORT PROMPTS? YES↵
BIN OR SYM: SYM↵
FIX OR VAR: FIX↵
LENGTH: 13↵
  NAME, TYPE, START, LENGTH, DECIMAL PLACES
: NAME,C,1,9↵
: EMP.NO,N,.,4↵
: ↵
DESCRIPTION FILE WRITTEN.

-
```

The multiple description file named MULT is shown below.

```
-TYPE MULT↵

$$$FIL(WAGE)
***BIN OR SYM:
SYM
***FIX OR VAR:
VAR
***WHICH TYPE:
L
***NUMBER OF LINES PER RECORD IS:
1
***  NAME, TYPE, START, LENGTH, DECIMAL PLACES
EMP.NO,N,1,4
RATE,C,6,1
HOURS,N,8,2
WAGES,N,11,6,2

$$$FIL(TERMIN)
***BIN OR SYM:
SYM
```

```
***FIX OR VAR:
FIX
***LENGTH:
13
***  NAME, TYPE, START, LENGTH, DECIMAL PLACES
NAME,C,1,9
EMP.NO,N,.,4


-
```

Note that each data file description is preceded by an appropriate heading of the form:

$$$FIL(title)

When the user wants to specify the data file description in an IML command, he types the multiple description file name, followed by the title in parentheses, at the normal position for the description file name. For example:

```
-SELECT FROM WDATA:MULT(WAGE) IF RATE='H' TO T↵
```

The command specifies a data file named WDATA defined by a data file description, WAGE, on a multiple description file named MULT.

NOTE: When more than one data file description has the same title in a multiple description file, IML always uses the first data file description in the file. It is suggested, therefore, that the user specify different titles for each data file description in a given multiple description file.

Rules Files

The user creates a set of rules as part of a multiple description file by entering T (for terminal) for the rules file name in an UPDATE, VERIFY, or REPORT command. When the system prompts, for example, in UPDATE,

SAVE UPDATE RULES ON:

the user enters the name of the multiple description file, followed by a title enclosed in parentheses, for the set of rules. The specified title must conform to the rules for naming files.[1] For example:

```
-UPDATE WEEKLYHOURS:MULT(WAGE) TO HOURSUPD AS PER T↵
SAVE UPDATE RULES ON: MULT(WAGE)↵
MULT.. OLD FILE↵
:
```

The user specifies a data file description on a multiple description file.

The rules are titled WAGE and written on a multiple description file named MULT.

UPDATE prompts for the rules statements.

The entered rules are appended to MULT with the heading:

$$$UPD(WAGE)

1 – See the *Tymshare TYMCOM-IX EXECUTIVE Reference Manual* for details on file naming conventions.

NOTE: The data file description and rules set need not be on the same multiple description file. For example, the command

```
-VERIFY WEEKLYHOURS:MULT(WAGE) AS PER PAYROLL(TIMECARD)↵
```

specifies a data file description on the MULT multiple description file and a set of verification rules on the PAYROLL multiple description file.

The table below specifies the various headings used for UPDATE, VERIFY, and REPORT rules in a multiple description file.

Heading	Specifies
$$$UPD(title)	UPDATE rules
$$$VER(title)	VERIFY rules
$$$REP(title)	REPORT rules

The user may specify the same title for UPDATE, VERIFY, and REPORT rules; the appropriate set of rules is located according to the IML command used. For example,

```
-VERIFY WEEKLYHOURS:WDESC AS PER MULT(WAGE)↵
```

instructs VERIFY to read the rules following the heading

$$$VER(WAGE)

in the file MULT. Similarly,

```
-REPORT HOURSUPD:WDESC TO RPT AS PER MULT(WAGE)↵
```

instructs REPORT to read the rules following the heading

$$$REP(WAGE)

in the MULT multiple description file.

NOTE: If the same title is assigned to several sets of rules for a given IML procedure (UPDATE, VERIFY, or REPORT), the specified set which appears first in the multiple description file is used. It is suggested, therefore, that the user specify a different title for each set of UPDATE rules, a different title for each set of VERIFY rules, and a different title for each set of REPORT rules.

Altering a Multiple Description File

The DEFINE command provides five options for examining and editing data file descriptions or rules sets on a multiple description file. The form of the command, including an optional data file description title or rules set title, is

```
                   multiple
-DEFINE [option]   description(title)↵
                   file
                   name
```

where a title may be included to specify a particular data file description or rules set on the multiple description file named.[1] The general purpose of each option is presented in the table below. The discussion following the table details all options, explaining the various forms of each option with examples.[2]

Option	Purpose
[DIR]	Prints titles of data file descriptions and/or rules sets on specified multiple description file.
[LIST]	Prints data file descriptions and/or rules sets on specified multiple description file.
[DELETE]	Deletes data file description or rules set on specified multiple description file.
[RENAME]	Renames data file description or rules set on specified multiple description file.
[EDIT]	Permits user to edit a data file description on a multiple description file or on a simple description file.

Determining Titles

The [DIR] option prints a directory listing of the contents of a multiple description file; that is, it prints the titles of the data file descriptions and rules sets on the file. For example, if the user wants a directory listing for a multiple description file named PAYROLL, he types:

```
-DEFINE [DIR] PAYROLL ↩

OK.

$$$FIL(TERMIN)
$$$FIL(REVIEW)
$$$FIL(WAGE)
$$$UPD(WAGE)
$$$VER(WAGE)
$$$REP(PRINTCHECK)
$$$VER(TIMECARD)

FINISHED.

-
```

The file contains three data file descriptions, one set of UPDATE rules, two sets of VERIFY rules, and one set of REPORT rules.

The user may request only data file descriptions by typing a DEFINE command of the form:

```
                   multiple
-DEFINE [DIR-F]    description ↩
                   file
                   name
```

1 – When more than one data file description or an UPDATE, VERIFY, or REPORT rules set has the same title, the first so-titled one is referred to.

2 – Each option has a form to specify only data file descriptions, UPDATE rules, VERIFY rules, or REPORT rules.

For example:

```
-DEFINE [DIR-F] PAYROLL↵

OK.


$$$FIL(TERMIN)
$$$FIL(REVIEW)
$$$FIL(WAGE)

FINISHED.


-
```

Similarly, the user may request only VERIFY, UPDATE, or REPORT rules sets by using the option [DIV–V], [DIR–U], or [DIR–R], respectively, in the DEFINE command. For example:

```
-DEFINE [DIR-U] PAYROLL↵

OK.


$$$UPD(WAGE)

FINISHED.

-DEFINE [DIR-V] PAYROLL↵

OK.


$$$VER(WAGE)
$$$VER(TIMECARD)

FINISHED.


-
```

Listing Data File Descriptions and Rules Sets

The [LIST] option lists either a particular data file description or rules set on a specified multiple description file or the entire multiple description file. To list the entire multiple description file, the user types:

```
                        multiple
-DEFINE  [LIST]  description
                        file
                        name
```

For example, to list the entire contents of the PAYROLL multiple description file, the user types:

```
-DEFINE [LIST] PAYROLL
```

The user may list a single data file description or rules set by typing a DEFINE command of the form

```
                          multiple
-DEFINE  [LIST-x]  description(title)
                          file
                          name
```

where x may be F, V, U, or R to specify a data file description or a VERIFY, UPDATE, or REPORT rules set, respectively. A title is required and specifies the desired data file description or rules set. For example, if the user wants a listing of the data file description titled WAGE on the PAYROLL multiple description file, he types:

```
-DEFINE [LIST-F] PAYROLL(WAGE)

OK.


$$$FIL(WAGE)
***BIN OR SYM:
SYM
***FIX OR VAR:
VAR
***WHICH TYPE:
L
***NUMBER OF LINES PER RECORD IS:
1
***  NAME, TYPE, START, LENGTH, DECIMAL PLACES
EMP.NO,N,1,4
XCODE,C,5,1
RATE,C,6,1
HOURS,N,8,4,1
WAGES,N,.,6,2
DATE,N,.,6


FINISHED.


-
```

Deleting a Data File Description or Rules Set

The [DELETE] option permits the user to delete a single data file description or rules set on a specified multiple description file.

To delete the *first* data file description or rules set with a given title, the user types a DEFINE command of the form

```
                        multiple
-DEFINE  [DELETE]       description(title)⤸
                        file
                        name
```

DEFINE then prints the title of the data file description or rules set to be deleted, since more than one may have the specified title, and waits for the user to respond with YES or NO, followed by a carriage return. For example, the following commands illustrate the [DELETE] option for the PAYROLL multiple description file. See the PAYROLL directory listing on page 165.

```
-DEFINE [DELETE] PAYROLL(REVIEW) ⤸

OK.


OKAY TO DELETE $$$FIL(REVIEW)? YES⤸

FINISHED.

-DEFINE [DELETE] PAYROLL(WAGE)⤸

OK.


OKAY TO DELETE $$$FIL(WAGE)? NO⤸

COMMAND ABORTED.
-
```

Note that the first of three data file descriptions and rules sets with the WAGE title is used.

The user may specify that the deletion applies to a data file description or a VERIFY, UPDATE, or REPORT rules set for the title given by using the [DELETE–F], [DELETE–V], [DELETE–U], or [DELETE–R] option, respectively, in the DEFINE command. In this situation, DEFINE does not request that the user confirm the deletion. For example, the user wants to delete the UPDATE rules set titled WAGE on the PAYROLL multiple description file:

```
-DEFINE [DELETE-U] PAYROLL(WAGE)↵

OK.


FINISHED.


-
```

Renaming a Data File Description or Rules Set

The [RENAME] option changes the title of a data file description or rules set on a specified multiple description file. A DEFINE command of the form

```
                    multiple
-DEFINE [RENAME]    description(title) AS new title↵
                    file
                    name
```

requests the renaming of the *first* data file description or rules set with the specified title to the specified new title.

The current directory listing of the PAYROLL multiple description file is shown first to clarify the examples which follow.

```
-DEFINE [DIR] PAYROLL↵

OK.


$$$FIL(TERMIN)
$$$FIL(WAGE)
$$$VER(WAGE)              Note that the WAGE title appears twice.
$$$REP(PRINTCHECK)
$$$VER(TIMECARD)

FINISHED.


-
```

The user wants to change the title of the first data file description or rules set having WAGE for a title. He types:

```
-DEFINE [RENAME] PAYROLL(WAGE) AS WKWAGE↵

OK.


FINISHED.
```

```
-DEFINE [DIR] PAYROLL↵

OK.


$$$FIL(TERMIN)
$$$VER(WAGE)
$$$REP(PRINTCHECK)
$$$VER(TIMECARD)
$$$FIL(WKWAGE)

FINISHED.


-
```

Note that the data file description or rules set renamed appears at the end of the multiple description file.

The user may specify that the renaming applies to a data file description or a VERIFY, UPDATE, or REPORT rules set for the title given by using the [RENAME–F], [RENAME–V], [RENAME–U], or [RENAME–R] option, respectively, in the DEFINE command. For example,

```
–DEFINE [RENAME–V] PAYROLL(WAGE) AS CKWAGE↵
```

specifies that the user wants to rename the VERIFY rules set with the WAGE title on the PAYROLL multiple description file; he wants the new title to be CKWAGE.

Editing a Data File Description

The [EDIT] option allows the user to alter a data file description on a multiple description file or to alter a simple description file.

To edit a data file description on a multiple description file, the form used is

```
                    multiple
–DEFINE  [EDIT]     description(title)↵
                    file
                    name
```

where a title is required; it specifies the title of the data file description to be edited. To edit a simple description file, the form used is:

```
                    description
–DEFINE  [EDIT]     file        ↵
                    name
```

Either form of the DEFINE [EDIT] command produces the same results. See page 55 for complete documentation on editing a description file.

USING COMMAND FILES

IML is designed for efficient and flexible command file operation.[1] The user may write a set of IML commands on a file and initiate the execution of these commands with as little or as much terminal interaction as desired.

IML contains all the capabilities for simple command file operation and offers extended features to permit input from the terminal, as well as terminal output, during command file operation and to specify terminal output when a TOUT file is in use.[2] Furthermore, IML reads the information to be written and automatically sends IML diagnostics and informative messages, such as

```
55 RECORDS SELECTED FROM 55
```

to the terminal, ensuring that only the actual data and user-specified comments are written on the TOUT file.

This discussion begins with an explanation of simple command file operation. Next, several additional rules file statements specifically for command file control are introduced. These rules may appear in UPDATE, VERIFY, and REPORT rules files. The CREATE program and its command file operation with input from the terminal are discussed next. Finally, the combination of IML and a language subsystem is explored. This discussion includes the creation of IML command files by a SUPER BASIC program, and the use of the EXECUTIVE DUMP command to leave and reenter the subsystem during command file execution.

Simple Command Files: Creation and Execution

The user may create a command file in EDITOR.[3] He enters each command as if he were entering it directly in the EXECUTIVE. Note that all responses normally entered at the terminal, such as a carriage return for a NEW FILE or OLD FILE message, must appear at the correct position in the command file. The last command in the command file should be

```
COMMAND T
```

which returns control to the terminal for command entry and execution.

When the user writes the commands on a file, he must use a line feed after the WRITE command. For example:

```
-EDITOR⊃
*APPEND⊃
SELECT FROM INV:IDESC IF QTY<3 TO SMALL OTHERS TO LARGE⊃
⊃          The user includes two carriage returns to confirm the names of the two output files specified.
⊃
TYPE SMALL⊃       The user may include any EXECUTIVE commands.
COMMAND T⊃        After the command file operation, the user wants to return control to the terminal.
*WRITE¬           The WRITE command must be followed by a line feed.
TO: COMSELECT⊃
 NEW FILE⊃
79 CHARACTERS
*QUIT⊃

-
```

1 – Command files are detailed in the *Tymshare TYMCOM-IX EXECUTIVE Reference Manual.*
2 – See the *Tymshare TYMCOM-IX EXECUTIVE Reference Manual* for an explanation of TOUT files.
3 – See the *Tymshare EDITOR Reference Manual* for an explanation of creating files in EDITOR.

The user begins execution of a command file by entering at the terminal a command of the form:

–COMMAND command file name↩

For example, the user executes the command file created above.

```
-COMMAND COMSELECT↩                          The user initiates execution of the command file.
8 RECORDS SELECTED FROM 12                   The SELECT command is executed.

1120544   AIR VALVE          1
1120550   NEDL VALVE         2
1120552   SHADE STOP         2
1120555   I T HSG            1               The TYPE command is executed.
1120574   FOCUS RING         2
1120576   YOKE               1
1120577   TRUNNION           1
1120579   DIAL ELV           2

-                    Control is returned to the terminal as per the COMMAND T command.
```

Rules Statements for Command File Control

When VERIFY, UPDATE, or REPORT is executed from a command file, the last rules file statement may be RUN; otherwise, RUN must appear at the appropriate position in the command file. For example, the last lines in a rules file might be:

200 FINAL
210 TYPE "END OF UPDATING PROCEDURE"
RUN *Note that RUN is not preceded by a line number.*

Otherwise, RUN must appear in the command file immediately after the UPDATE command.

Two additional rules file statements, @INPUT and @TYPE, are available for use in UPDATE, VERIFY, and REPORT rules files. The @INPUT and @TYPE statements are similar to the INPUT and TYPE statements, except they direct control to the terminal for that statement even though a command file or TOUT file is in use.[1]

The @INPUT statement is used with command files and permits input of one field from the terminal, rather than input from the command file as with INPUT. The form used is

@INPUT field name $\begin{Bmatrix}\text{IF conditions}\\\text{FOR conditions}\end{Bmatrix}$

or simply:

@INPUT field name

For example, when the statements below are executed in a command file, line 60 accepts a value for CODE from the terminal; line 70, on the other hand, takes the value for XCOUNT from the command file.

1 – The INPUT and TYPE statements are discussed on page 34.

```
10 DECLARE
20 CODE,C,1
30 XCOUNT,I,3
40 INITIAL
50 TYPE "ENTER A CODE FOR UPDATE PROCEDURE"
60 @INPUT CODE
70 INPUT XCOUNT
```

The @TYPE statement is used with TOUT files and permits the user to print information on the terminal rather than writing on the TOUT file as with TYPE. The form used is

@TYPE item list $\left\{\begin{array}{l}\textbf{IF conditions}\\ \textbf{FOR conditions}\end{array}\right\}$

or simply:

@TYPE item list

The item list may contain any item valid in a TYPE statement.[1] For example:

@TYPE "COMMENT",CR,FLD1,4B,FLD2

When the statements below are executed with a TOUT file in use, line 40 writes the specified information on the TOUT file, whereas line 50 prints the specified information at the terminal.

```
40 TYPE "RESULT IS ",FLD1,4B,"RECORD NUMBER",RECNO
50 @TYPE "PLEASE ENTER THE CODE FOR ",NAME
```

Documentation of Command File Execution

The NOTE command is incorporated in IML as an aid in documenting command file execution. NOTE provides all the calendar utility fields and permits the user to specify one or several lines of text. The information may be printed at the terminal or written on a TOUT file.[2]

The complete form of the NOTE command is:

–**NOTE [TERMINAL] item list**↩

where the items in the list are separated by commas. The TERMINAL option instructs NOTE to print the information at the terminal whether or not a TOUT file is in use. The user may abbreviate the option word to any subset of the word TERMINAL.

The following table summarizes the items that may appear in the item list.

Item	Prints	Length
@CALMONTH	Name of current month as MMM	3
@CDATE	Current date as MMM.DD,YYYY	11
@CTIME	Current time as HH:MM AM or PM	8

(Table continues)

1 – See page 35 for a complete list of valid items in an @TYPE or TYPE statement.
2 – Refer to the *Tymshare TYMCOM-IX EXECUTIVE Reference Manual* for an explanation of TOUT files.

Item	Prints	Length
@DATE	Current date as MM/DD/YY	8
@DAY	Current day as DD	2
@MONTH	Current month as MM	2
@NDATE	Current date as YYMMDD	6
@TIME	Current time as HH:MM	5
@WEEKDAY	Name of current day of week	9
@YEAR	Current year as YYYY	4
CR	Carriage return	1
LF	Line feed	1
nB	Specified number of blanks, where n may range from 1 to 80	n
“text”	Text enclosed in single or double quote marks, where text may contain as many as 80 characters	Number of characters in quote marks
n“text”	Text enclosed in single or double quote marks n times, where the number of characters in quote marks times n may not exceed 80	Number of characters in quote marks times n

For example, the following are valid NOTE commands:

```
–NOTE “UPDATING PHASE COMPLETED AT ”,@CTIME,5B,@CDATE↵
–NOTE 10“*”,@WEEKDAY,2B,“REPORT IS FINISHED”,10“*”,CR,CR↵
–NOTE “INVENTORY AS OF ”,@DATE,2B,@TIME,“ IS WRITTEN ON NEWINV”↵
```

NOTE: The NOTE command includes an automatic carriage return at the end of the item list.

When the TERMINAL option is not used, the information is printed at the terminal only if a TOUT file is not being used.

The user may type

```
–NOTE [VERSION]↵
```

to determine the version number of the current NOTE program.

CREATE and Command Files

The CREATE command may appear in command files for entering data from a file, from the terminal, or from the command file.

When the user wishes to enter data from the terminal while the command file is in use, the form used is:

```
                                      data   description                BY NAME
-CREATE  [PROMPT]    APPENDING TO     file : file         FROM @T       BY LIST        ⮐
         [character]                  name   name                       WITH field list
```

This is the same as the general command form for CREATE, except that the user specifies

FROM @T

to request data entry from the terminal during command file execution.

For example, the user wants to use a command file to create a data file with data entry from the terminal, sort the created data file, and then print the sorted file at the terminal. The command file shown below performs the desired tasks.

```
-TYPE COMCREATE ⮐

CREATE TDATA:TDES FROM @T
                     <------------------
SORT TDATA:TDES TO TDATASTD BY FLD1       Carriage returns appear on these lines to
                     <------------------  confirm the specified output file name.
TYPE TDATASTD
COMMAND T

-
```

The interaction below shows the user entering data for CREATE from the terminal while the command file is in use.

```
-COMMAND COMCREATE ⮐

OK.                                        The CREATE command is executed.


DATA MUST BE ENTERED IN THE FOLLOWING ORDER:
FLD1 C 20
FLD2 N 8 2
FLD3 N 8 2
FLD4 N 4


1: SANDERSON,145.45,3200.00,529 ⮐
2: MILLER,45.90,500.35,334 ⮐
3: MASON,250.00,1295.45,277 ⮐
4: AMES,300.65,2579.66,143 ⮐
5: ⮐                                      The user terminates data entry with a carriage return.
4 RECORDS CREATED.

SORT FINISHED.                                  Control returns to the command file.

AMES                  300.65 2579.66 143        The TYPE command is executed.
MASON                 250.00 1295.45 277
MILLER                 45.90  500.35 334
SANDERSON             145.45 3200.00 529

-
```

If the user wants to enter the data for CREATE in the command file, he uses the normal form for CREATE with terminal data entry. For example:

CREATE ABC:DESC FROM T

The CREATE command is then followed by a carriage return (for the NEW FILE or OLD FILE message), the data for each record, and a final carriage return to terminate data entry.

To enter the data for CREATE from a file, the user types a CREATE command specifying the input file name. For example, to create a file named RATES according to the RDESC description file from data on a file named RDATA, the user types in the command file:

CREATE RATES:RDESC FROM RDATA

The next line contains a carriage return confirming the NEW FILE or OLD FILE message.

Creation of an IML Command File in a Language Subsystem

The user may write a program in SUPER BASIC, SUPER FORTRAN, or BATCH FORTRAN which causes the creation and execution of an IML command file, allowing another user to call the appropriate subsystem, load the program, and answer a few simple questions to accomplish the task. The user may write a program as simple or as complex as needed to accomplish his tasks.

The user may write instructions which specify the form and content of the command file, utilizing all the language capabilities to provide the desired degree of flexibility. Depending on the user's requirements, the program may create a command file containing a series of IML and EXECUTIVE commands, use variable file names for the data and command files, and specify loops with a user-entered index.

The following examples present very simple SUPER BASIC programs illustrating the technique for creating command files without attempting to present any of the more complex capabilities available in the language.

The SUPER BASIC program below creates a command file named COMFILE to select from the INV data file the record(s) containing a particular part number.

```
-TYPE PROG1↩

10 VAR=ZERO
15 STRING P
20 PRINT "WHICH PART NUMBER DO YOU WANT TO SEE":
30 INPUT P
40 OPEN "COMFILE",OUTPUT,1
45 PRINT ON 1:"QUIT"
50 PRINT ON 1:"SELECT FROM INV:IDESC IF PARTNO HAS '"+P+"' TO T"
60 PRINT ON 1:"COMMAND T"
70 CLOSE 1
80 OPEN "COMFILE",INPUT,*
```

When the user wants to perform this task, he simply calls SUPER BASIC, loads and runs PROG1, and answers the request for the desired part number. The program, in turn, writes the necessary SUPER BASIC, IML, and EXECUTIVE commands on a command file and executes it.

```
-SBASIC

>LOAD PROG1
>RUN
WHICH PART NUMBER DO YOU WANT TO SEE? 1120579

1120579   DIAL ELV           2


1 RECORDS SELECTED FROM 12

-
```

Statement 50, in PROG1, writes the appropriate SELECT command by inserting the entered part number between the phrase

SELECT FROM INV:IDESC IF PARTNO HAS '

and the phrase

' TO T

Because PARTNO is defined in IDESC as a character field, the program must contain the entered part number surrounded with single quote marks.

Statement 80, in PROG1, opens COMFILE as a command file, indicated by an asterisk (*) instead of a file identification number.

NOTE: It is necessary to include a PAUSE or STOP statement after opening a file as a command file when additional statements appear after the OPEN statement. See the example on the following page for an illustration.

In addition to creating and initiating a command file from a language subsystem, the user may reenter the subsystem after IML or EXECUTIVE commands, and continue the program without losing the previous program status and variable values. This is accomplished by including a DUMP command,[1] specifying a file on which to save a core image of the current subsystem work, immediately after leaving the subsystem. When the user wants to reenter the subsystem as it previously existed, he enters a RUN command followed by the name of the DUMP file.

The SUPER BASIC program on the next page includes the DUMP and RUN commands in the command file to perform a task similar to the one in the previous example, but also to reenter the subsystem and continue the program.

1 – See the *Tymshare TYMCOM-IX EXECUTIVE Reference Manual* for an explanation of the DUMP command.

```
-TYPE PROG2↵

10 VAR=ZERO
15 STRING P,K
20 PRINT "WHICH PART NUMBER DO YOU WANT TO SEE":
30 INPUT P
40 OPEN "COMFILE",OUTPUT,1
41 PRINT ON 1:"QUIT"
42 PRINT ON 1:"DUMP TEMP"
43 PRINT ON 1   A carriage return only is written, confirming the file named in the DUMP command.
50 PRINT ON 1:"SELECT FROM INV:IDESC IF PARTNO HAS '"+P+"' TO T:
CR,CR,10B,DESC,QTY"
54 PRINT ON 1:"RUN TEMP"
56 PRINT ON 1:"GO TO 100"
60 PRINT ON 1:"COMMAND T"
70 CLOSE 1
80 OPEN "COMFILE",INPUT,*
90 STOP
100 CLOSE "TEMP"
102 CLOSE *
104 CLOSE "COMFILE"
106 PRINT "ANOTHER PART":
110 INPUT K
120 IF K="YES" THEN GO TO 20 ELSE PRINT "END OF RUN"

-
```

When a user wants to perform the task, he calls SUPER BASIC, loads and runs PROG2, and answers the simple questions. The program handles the creation and execution of the necessary command file, as well as reentering the subsystem to perform another similar task.

```
-SBASIC↵

>LOAD PROG2↵
>RUN↵
WHICH PART NUMBER DO YOU WANT TO SEE? 1120537↵

 NEW FILE          This is printed for the DUMP file created; the
                   response is contained in the command file.
          PRM MIRROR            4    The SELECT command requests
                                     the description and quantity
                                     for the specified part number.

1 RECORDS SELECTED FROM 12
SBASIC                  The user automatically reenters the subsystem as per the RUN command.
ANOTHER PART? YES↵
WHICH PART NUMBER DO YOU WANT TO SEE? 1120544↵

 NEW FILE               The program deletes the DUMP file after each SELECT procedure.

          AIR VALVE             1
```

```
1 RECORDS SELECTED FROM 12
SBASIC
ANOTHER PART? NO ↵
END OF RUN

>
```

The STOP statement on line 90 in the SUPER BASIC program is necessary, since several statements follow after the program opens a command file. Statements 100 and 104 delete the indicated files by specifying the actual file name in double quote marks (instead of the file number) with the CLOSE command.

APPENDIX A
COMMAND AND RULES FILE SUMMARIES

The following tables summarize the EXECUTIVE IML commands and the rules file commands and statements. The first table contains each IML command in two general forms: The first form specifies the required information only; the second specifies all possible command components. The second and third tables present the rules file commands and statements.

EXECUTIVE IML COMMANDS	
Command	**Description and General Form**
APPEND	Adds data in one file to the end of another file, regardless of the structure of the files but requiring that both files be either binary or symbolic. $-\text{APPEND} \begin{Bmatrix} \text{T} \\ \text{file name}_1 \end{Bmatrix} \text{ TO file name}_2$ ↵
CREATE	Creates records of a data file from free format data entered at the terminal or from a file, according to the specified description file. $-\text{CREATE } \begin{matrix}\text{data}\\ \text{file}\\ \text{name}\end{matrix} : \begin{matrix}\text{description}\\ \text{file}\\ \text{name}\end{matrix} \text{ FROM } \begin{Bmatrix}\text{T}\\ \text{input file name}\\ \text{@T}\end{Bmatrix}$ ↵ $-\text{CREATE } \begin{Bmatrix}\text{[PROMPT]}\\ \text{[character]}\end{Bmatrix} \text{ APPENDING TO } \begin{matrix}\text{data}\\ \text{file}\\ \text{name}\end{matrix} : \begin{matrix}\text{description}\\ \text{file}\\ \text{name}\end{matrix}$ $\text{FROM } \begin{Bmatrix}\text{T}\\ \text{input file name}\\ \text{@T}\end{Bmatrix} \begin{Bmatrix}\text{BY NAME}\\ \text{BY LIST}\\ \text{with field list}\end{Bmatrix}$ ↵ $-\text{CREATE [character] } \begin{matrix}\text{data}\\ \text{file}\\ \text{name}\end{matrix} : \begin{matrix}\text{description}\\ \text{file}\\ \text{name}\end{matrix} \text{ FROM}$ $\begin{matrix}\text{input}\\ \text{file}\\ \text{name}\end{matrix} \begin{Bmatrix}\text{BY NAME}\\ \text{BY LIST}\\ \text{WITH field list}\end{Bmatrix} \text{ ERRORS TO } \begin{matrix}\text{errors}\\ \text{file}\\ \text{name}\end{matrix}$ ↵

(Table continues)

Command	Description and General Form
DEFINE	Creates the description of a data file and writes it on the file named. –DEFINE description file name ↵ –DEFINE { [EDIT] description file name | [option][1] multiple description file name(title) } ↵
MERGE	Merges two sorted files to a single sorted file. –MERGE sorted file name$_1$: description file name WITH sorted file name$_2$ TO merged file name BY key field list ↵ –MERGE sorted file name$_1$: description file name { IF conditions | FOR conditions } WITH sorted file name$_2$ { IF conditions | FOR conditions } TO merged file name BY key field list ↵
PURGE, using a single data file	Deletes from a data file the records for which the specified conditions are true. –PURGE FROM data file name : description file name TO output file name ↵ –PURGE [PRIME] FROM data file name : description file name { IF conditions | FOR conditions } TO output file name : field list[2] BY key field list OTHERS TO complementary output file name ↵

1 – DEFINE options appear on page 165.
2 – A table of valid items that may be specified in the field list appears on page 88.

Command	Description and General Forms

PURGE, using a data file and an activity file	Deletes from a data file the records which match activity records on the key fields named.
	—PURGE activity file name : description file name FROM data file name : description file name TO output file name BY key field list ↩
	—PURGE [SINGLE] activity file name : description file name {IF conditions | FOR conditions}
	FROM data file name : description file name {IF conditions | FOR conditions}
	TO output file name : field list {IF interfile conditions | FOR interfile conditions}
	BY key field list OTHERS TO complementary output file name ↩

REPLACE	Replaces records in the data file with activity records which match on the key fields named. —REPLACE data file name : description file name WITH activity file name : description file name TO output file name BY key field list ↩ —REPLACE [SINGLE] data file name : description file name {IF conditions | FOR conditions} WITH activity file name : description file name {IF conditions | FOR conditions} TO output file name : data field name WITH description[1] data field name WITH description[1] , . . . {IF interfile conditions | FOR interfile conditions} BY key field list ↩
REPORT	Generates a report of the data file according to the statements in the rules file named. —REPORT data file name : description file name TO output file name AS PER rules file name ↩ —REPORT data file name : description file name {IF conditions | FOR conditions} TO output file name AS PER rules file name ↩

(Table continues)

1 – A list of valid items that may be specified in the description appears on page 98.

Command	Description and General Form
SELECT, using a single data file	Selects from a data file the records for which the specified conditions are true. –SELECT FROM data file name : description file name TO output file name ↵ –SELECT [PRIME] FROM data file name : description file name {IF conditions FOR conditions} TO output file name : field list[1] BY key field list OTHERS TO complementary output file name ↵
SELECT, using a data file and an activity file	Selects from a data file the records which match activity records on the key fields named. –SELECT activity file name : description file name FROM data file name : description file name TO output file name BY key field list ↵ –SELECT [SINGLE] activity file name : description file name {IF conditions FOR conditions} FROM data file name : description file name {IF conditions FOR conditions} TO output file name : field list {IF interfile conditions FOR interfile conditions} BY key field list OTHERS TO complementary output file name ↵
SORT	Sorts a data file according to its data in the key fields named. –SORT unsorted file name : description file name BY key field list ↵ –SORT unsorted file name : description file name TO output file name BY key field list ↵

1 – A table of valid items that may be specified in the field list appears on page 88.

Command	Description and General Form
UPDATE, using a single data file	Modifies the records of a data file according to the statements in the rules file named. −UPDATE data file name : description file name TO output file name AS PER rules file name ↲ −UPDATE data file name : description file name {IF conditions / FOR conditions} TO output file name BY key field list AS PER rules file name ↲
UPDATE, using a data file and an activity file	Modifies records in a data file, using activity records which match on the key fields named, according to the statements in the specified rules file. −UPDATE data file name : description file name WITH activity file name : description file name TO output file name BY key field list AS PER rules file name ↲ −UPDATE {[SINGLE] / [MULTIPLE]} data file name : description file name {IF conditions / FOR conditions} WITH activity file name : description file name {IF conditions / FOR conditions} TO output file name {IF interfile conditions / FOR interfile conditions} BY key field list AS PER rules file name ↲
VERIFY	Checks the contents of a data file according to the statements in the rules file named. −VERIFY data file name : description file name AS PER rules file name ↲ −VERIFY data file name : description file name {IF conditions / FOR conditions} TO output file name BY key field list AS PER rules file name ↲

RULES FILE COMMANDS	
Command	**Description and General Form**
LIST	Lists all, or specified, lines of current rules file statements. : $\underline{\text{LIST}}$↵ *or* : $\underline{\text{LIST} \left\{\begin{matrix} n \\ n_1{:}n_2 \end{matrix}\right\}}$↵ where $n_1{:}n_2$ specifies a range of lines from line n_1 through line n_2.
n, followed by a carriage return	Deletes line n from the current rules file. : $\underline{\text{n}}$↵
n, followed by rules file statement(s)	Adds or replaces statement(s) at line n in the current rules file. : $\underline{\text{n statement(s)}}$↵
QUIT	Writes the current rules file statements on a user-specified file and returns control to the EXECUTIVE. : $\underline{\text{QUIT}}$↵ *or* : $\underline{\text{Q}}$↵
RUN	Writes the current rules file statements on a user-specified file and begins execution of the IML command. : $\underline{\text{RUN}}$↵

RULES FILE STATEMENTS

Statement	Description and General Form
field declaration	Defines a temporary field for the duration of the command procedure. : n field name, $\left\{\begin{matrix}C\\N\\I\\R\\D\end{matrix}\right\}$,maximum field length,decimal places ↵ where n is a line number, and the specification of the number of decimal places is optional.
picture format declaration	Specifies a general picture format for use throughout the rules file.[1] : n FORMAT x:picture format ↵ where n is a line number, and x is a picture format reference number.
IF	Evaluates a relational expression, and for a true value executes the statement(s) which immediately follow. : n IF condition ↵ where n is a line number.
ORIF	Evaluates a relational expression when the preceding IF or ORIF statements are false, and for a true value executes the statement(s) which immediately follow. : n ORIF condition ↵ where n is a line number.
ELSE	Executes the statement(s) which immediately follow when the preceding IF and ORIF statements are false. : n ELSE ↵ *or* : n ELSE statement(s) ↵ where n is a line number.
DO	Executes unconditionally the statements which follow, terminating the control of the preceding IF, ORIF, or ELSE statements. : n DO ↵ *or* : n DO statement(s) ↵ where n is a line number.
replacement	Replaces specified field(s) with the expression value. : n expression TO field list ↵ where n is a line number, and the expression may be a number, one or more field names in an arithmetic expression, or text enclosed in quote marks.
INPUT	Accepts a value from the terminal or from a command file if in use; replaces the entered value in the specified field; and continues command operation. : n INPUT field name ↵ where n is a line number.

(Table continues)

1 – This statement applies only to a REPORT rules file. Picture format construction is detailed on page 133.

Statement	Description and General Form
@INPUT	Accepts a value from the terminal when a command file is in use; replaces the entered value in the specified field; and returns control to the command file. : **n @INPUT field name**↵ where n is a line number.
TYPE	Writes the specified information at the terminal or on a TOUT file if in use.[1] : **n TYPE item list**[2]↵ where n is a line number.
@TYPE	Types the specified information at the terminal regardless of a TOUT file in use. : **n @TYPE item list**↵ where n is a line number.
DONE	Terminates execution of the current rules file section and begins execution of the next section.[3] : **n DONE**↵ where n is a line number.
PRINT[4]	Writes the specified information on the REPORT output file. : **n PRINT item list**[5]↵ where n is a line number.
ON[4]	Performs indicated operations whenever a specified field changes value. : **n ON field list**↵ where n is a line number.
RETURN TO	The RETURN TO statement allows the user to transfer control to a preceding line of rules in the same rules section. : **RETURN TO n** $\left\{\begin{array}{l}\textbf{IF conditions}\\\textbf{FOR conditions}\end{array}\right\}$↵ where n is a line number, and the IF or FOR modifier is optional.
ROUNDIN	The ROUNDIN statement rounds the values in all numeric fields to their specific field descriptions before the values are used in an arithmetic calculation. : **ROUNDIN**↵
ROUNDOUT	The ROUNDOUT statement rounds numeric values and arithmetic results to conform to the receiving field descriptions. : **ROUNDOUT**↵
FINISH	The FINISH statement terminates record processing and transfers control to the FINAL section, if any, or simply ends the command procedure. : **FINISH** $\left\{\begin{array}{l}\textbf{IF conditions}\\\textbf{FOR conditions}\end{array}\right\}$↵

1 – See the *Tymshare TYMCOM-IX EXECUTIVE Reference Manual* for an explanation of the TOUT command.
2 – A table of valid items appears on page 35.
3 – Rules file sections are discussed on page 30.
4 – This statement applies only to a REPORT rules file.
5 – Any of the items which appear on page 35 may be specified except NCR.

Statement	Description and General Form
ABORT	ABORT halts the procedure immediately and returns control to the EXECUTIVE. $\underline{\text{: ABORT } \begin{Bmatrix} \text{IF conditions} \\ \text{FOR conditions} \end{Bmatrix}}$ ↩
DELETE	DELETE omits the current record from the output file and performs an automatic DONE statement; that is, the remaining rules are ignored for the current record. DELETE may not appear in a REPORT rules file. $\underline{\text{: DELETE } \begin{Bmatrix} \text{IF conditions} \\ \text{FOR conditions} \end{Bmatrix}}$ ↩

APPENDIX B
ERROR MESSAGES

Appendix B contains a list of the most common IML error messages. The messages at the beginning of the list print a particular file name, field name, group of characters, or number in front of the message. These messages are arranged in alphabetical order according to the constant part of the message. The remaining error messages are arranged in alphabetical order.

Each message is followed by a possible explanation of what may cause the error condition. When the explanation is system error, the user did not cause the error and should reenter his IML command. If the user encounters a system error again, he should contact his Tymshare representative.

file name CANNOT BE RENAMED SO file name'SORTING' CONTAINS THE SORTED DATA.

File directory overflow prevents writing the sorted data on the specified file. The results are stored on file name'SORTING' in the user's directory.

character CANNOT BE USED IN A FIELD NAME, TRY AGAIN.

An illegal character was used in the field name.

number CHRS. MAX.

The user has entered more characters for an INPUT or @INPUT statement than allowed in the field length.

field name HAS 2 DIFFERENT DESCRIPTIONS, TRY AGAIN.

The user has specified this field name twice with different descriptions.

file name ILLEGAL AS FILE NAME.

TERMINAL, TELETYPE, NOTHING, or any left subset of these names may not be used as the data or activity file name.

file name ILLEGAL AS THE OUTPUT FILE.

The output file must be a file name; TERMINAL, TELETYPE, NOTHING, or any left subset of these names is not permitted.

file name IS DELETED, CANNOT FINISH THE RENAMING SO file name'SORTING' CONTAINS THE SORTED DATA.

A copy of the specified file is in the user's directory with the file name plus 'SORTING'. The user should log out, then log in and rename file name'SORTING' as desired.

file name IS NOT LEGAL AS 'APPEND FROM' FILE.

The file name following the word APPEND may not be any left subset of the word NOTHING.

file name IS NOT LEGAL AS 'APPEND TO' FILE.

The file name following TO may not be T (TERMINAL) or N (NOTHING); a new file name or an old file that is not protected may be specified.

field name NOT DEFINED.

The user has specified a field name that does not appear in the description file.

number RECORD OUT OF SEQUENCE ON file name.

The file(s) in the command are not sorted in the same sequence as the key field list following the word BY.

1ST CHARACTER OF NAME CANNOT BE A PERIOD (.), TRY AGAIN.

A field name must begin with a letter.

1ST CHARACTER OF NAME CANNOT BE NUMERIC, TRY AGAIN.

A field name must begin with a letter.

'APPEND TO' FILE BAD.

The file specified as the APPEND TO file is protected.

BAD COMMAND FILE INPUT.

A command in the active command file has an error; or a response, such as to the NEW FILE or OLD FILE message, is entered incorrectly.

BAD DATA.

A field contains illegal data, such as nonnumeric characters in a numeric field.

BAD DESCRIPTION FILE, TRY AGAIN.

The user specified a nonexistent file or a file that is not a legal description file, such as without $$$ at the beginning or without a blank line at the end.

BAD INFO. IN DESCRIPTION FILE.

The description file contains illegal specifications or is not in the proper form. See BAD DESCRIPTION FILE, above.

BAD INPUT.

System error. Try again.

BAD INPUT FROM COMMAND FILE.

A command in the active command file has an error; or a response, such as to the NEW FILE or OLD FILE message, is entered incorrectly.

BAD INPUT ON END OF FILE AT RECORD number.

In the variable length file, the last record is not terminated properly, or extra characters are at the end.

BAD NUMBER.

The user has entered for an INPUT or @INPUT statement something other than a legal number, such as letters or nonnumeric symbols.

BAD NUMERIC LITERAL.

The user has specified something other than a legal number, or a number which exceeds the maximum size allowed.

BAD RULES.

No rules file exists, or the rules file was not properly written.

BAD STATEMENT.

The user has entered something which is not a legal command and is not a line number.

BAD TYPE FOR BINARY FILE.

The field data type must be C, I, R, or D.

BAD TYPE FOR SYMBOLIC FILE.

The field data type must be C or N.

CAN NOT PROCESS SCRAMBLED FILE.

IML handles only normal symbolic or binary files.

CANNOT COMPLETE THE APPEND.

The APPEND TO file is bad because of a system error. Contact your Tymshare representative to restore your file.

CANNOT CREATE ON AN APPEND ONLY FILE.

The file name specified is an append-only file.

CANNOT OPEN file name.

The specified file is protected, or an illegal file name was specified.

CANNOT OPEN file name FOR OUTPUT WITHOUT NAME LIST.

T (TERMINAL) was specified as the output file, but it was not followed by a colon (:) and a list of fields desired. This is required for terminal output with binary files.

CANNOT OPEN THE DESCRIPTION FILE.

The description file specified is protected.

CANNOT SET 'APPEND TO' FILE TO WRITE MODE.

The file following APPEND TO is a read-only file. The user should redeclare the file.

CAN'T EXCEED 1024.

Maximum record length is 1024 characters.

COMMAND ABORTED.

The command has come to an abnormal halt.

COMMAND STARTING WITH 'characters' IS BAD.

A component is missing from the command, is in the wrong position, or is misspelled.

CONFLICTING DATA TYPES IN LIST.

The key fields do not have the same field description in the data and activity description files.

COUNT FIELD HAS ALPHA CHARS.

IML has discovered a C type variable length record containing nonnumeric characters in the count field at the beginning of the record.

COUNT MUST BE GREATER THAN ZERO.

The number of digits in the count field must be an integer number.

DATA WON'T FIT FIELD.

The value to be stored in a field requires more characters than specified in the field description.

DATA:FORMAT MISMATCH.

The user has entered the wrong type of format character for the field being printed.

DECIMAL PLACES MUST BE <= FIELD LENGTH.

The user has specified more decimal places than the maximum size of the field.

DECIMAL PLACES MUST BE NUMERIC.

The decimal places specification must be an integer number.

DISCOVERED NON-NUMERIC IN NUMERIC FIELD.

A numeric field contains something other than a legal number.

DOESN'T FIT FORMAT.

The number entered for an INPUT or @INPUT statement contains more characters than specified in the field description.

ENDING 'IF' NOT ALLOWED.

An IF clause cannot follow the output file when only one file is used in the command.

ERASE ERROR ON OUTPUT FILE.

System error. Try again.

FIELD ENDS BEYOND END OF RECORD.

The field length or field storage (for a binary file) requires more characters than remain in the record, according to the length specified in the description file.

FIELD NAMED . . . field name OF FILE file name NOT SAME LENGTH AS OTHER FILE.

The key fields in the data and activity files must have the same field length in their respective description files.

FILE OVERFLOW.

The output file has exceeded the maximum file size permitted.

FILE SIZE AND RECORD LENGTH CONFLICT ON FILE file name.

The number of characters in the file divided by the record length indicates a partial record.

FILE TYPE NOT SAME AS IN DESCRIPTION INFO. FOR FILE file name.

The file being used is not the same type, symbolic or binary, as specified in the description file.

FOUND RECORD OF LENGTH ZERO.

The count field of a variable length record contains zero.

FOUND RECORD WHOSE LENGTH IS NOT MOD 3.

The count field at the beginning of a binary variable length record contains an integer that is not a multiple of 3.

HEADING OVERFLOW.

The current execution of the HEADINGS section requires more lines than the number of lines per page.

ILLEGAL DATA OPERATION.

The user has combined character and numeric fields in an expression or replacement.

ILLEGAL FORMAT.

The user has specified a nonexistent format number or has specified an illegal picture format.

ILLEGAL INPUT SOURCE.

T (TERMINAL) or N (NOTHING) was used illegally as a data or activity file name.

ILLEGAL NESTING.

Lower level IF statements are not in numerical order; or an ORIF, ELSE, or DO statement does not have a corresponding IF statement above.

ILLEGAL OPTION.

The option specified does not exist or is not valid for this command.

ILLEGAL STATEMENT AT 'characters'.

The specified characters are not part of a valid statement form or do not specify a valid field name.

INADEQUATE STORAGE FOR OUTPUT.

No available file storage. Try later.

INCOMPLETE STATEMENT.

Something is missing in the specified line.

INPUT FILE MUST BE TYPE SYMBOLIC.

When using the CREATE command, the input file must be symbolic; it cannot be binary.

INSUFFICIENT INFORMATION IN THE DESCRIPTION FILE.

IML found a terminating carriage return before reaching any field descriptions, or a carriage return is missing at the end of the description file.

INSUFFICIENT STORAGE AVAILABLE TO COMPLETE THE APPEND.

The file resulting from the APPEND command requires more storage than the maximum file size permitted.

INSUFFICIENT STORAGE FOR OUTPUT FILE.

The output file requires more storage than the maximum file size permitted.

INVALID FILE SPECIFIED.

A field name followed by :A or :D is not legal at that point, or a character other than A or D was specified.

JUNK IN INPUT.

IML is reading an input file which has data missing at the end of the file.

LAST RECORD BAD ON file name.

Using a variable length file, the last record is not terminated properly, or extra characters are at the end of the file.

LAST RECORD NOT TERMINATED PROPERLY.

The last record is too short, does not end with a carriage return, or contains extra characters.

LENGTH CANNOT EXCEED 512.

The field length must be less than or equal to 512 characters.

LENGTH MUST BE > ZERO.

The record length must be an integer from 1 to 1024.

LENGTH MUST BE NUMERIC.

The record length must be an integer from 1 to 1024.

LEVELS MUST BE CONSECUTIVE.

Nested IF statements must be in numerical order; for example, an IF.3 statement must be preceded by an IF, IF.1, and IF.2 statement.

MAX OF 200 FIELD NAMES ALLOWED.

The maximum number of unique data, activity, and temporary field names for a given IML command is 200.

MAX OF 24 CHARACTERS IN NUMERIC FIELDS.

The field length of numeric fields may contain no more than 24 characters including the sign and the decimal point.

MISSING DESCRIPTION.

A field description is incomplete.

MISSING RULES.

Additional rules are required before the line number specified.

MISSING 'TO'.

A field name or expression was specified, but was not followed by the word TO.

MISSING 'WITH' OR FIELD NAME.

The field list following the output file is incorrectly specified; for example, the word WITH is missing between a field name and its replacement.

MUST BE A MULTIPLE OF 3. LENGTH:

The record length must be an integer multiple of 3.

NAME TOO LONG, 31 MAX.

A field name may contain no more than 31 characters.

NEGATIVE NUMBER DOESN'T FIT PICTURE FORMAT.

A D, $, or * format does not specify a sign format character for the minus sign required.

NO FIELD NAMES GIVEN IN FILE DESCRIPTION.

At least one field description must appear in the description file.

NO SUCH FIELD NAME AS field name.

The specified field name does not appear in the description file.

NO TITLE GIVEN.

The command requires the title of the data description or rules file desired.

NOT A LEGAL TYPE.

Data types for symbolic fields must be C or N; data types for binary fields must be C, I, R, or D.

NUMBER OF DECIMAL PLACES MUST BE NUMERIC.

The decimal places specification must be an integer number.

NUMBER TOO LARGE FOR: field name.

The number entered requires more characters than allowed in the field description.

OUTPUT RECORD NOT LEGAL.

Using a variable length data file, the specified output record does not conform to the description file.

OVERFLOW.

A value generated exceeds the largest number allowed. IML continues the operation, using the largest legal number for that data type.

OVERLAPPING FIELD DEFINITIONS, CANNOT PROCEED.

More than one field in the description file could occupy the same location in the record. The user should specify the names of the fields he wants to enter in a WITH clause.

PARENTHESIS MISMATCH.

A right or left parenthesis is missing.

PARTIAL RECORD IN FIXED LENGTH RECORD FILE.

There is a record containing fewer characters than the specified record length. All fixed length records must contain the specified number of characters.

POSITION ERROR ON file name.

System error. Try again.

PROGRAM TOO BIG AT LINE number.

The maximum size of a rules file has been exceeded.

PROMPT OPTION NOT LEGAL WITH 'BY' CLAUSE.

The PROMPT option cannot be specified with the BY NAME clause.

READ ERROR ON file name.

The specified file is bad. The user should request from Tymshare an older version of the same file.

RECORD >1024 CHARS.

A record contains more than the maximum 1024 characters allowed.

RECORD LENGTH FIELD CONTAINS ALPHA CHARACTERS.

The character count field must contain an integer number.

RECORD LENGTH WON'T FIT IN COUNT FIELD.

The character count for the record being written requires more space than specified for the character count field.

RECORD OF LENGTH ZERO.

The count field of a variable length record contains zero.

RECORD TOO LONG.

A record contains more than 1024 characters.

RECORD WHOSE LENGTH IS NOT MOD 3.

The count field at the beginning of a binary variable length record contains an integer that is not a multiple of 3.

RESERVED WORD USED AS FIELD NAME, TRY AGAIN.

See page 47 for a list of reserved field names that cannot appear in the description file or the DECLARE section.

SAME FIELD NAME GIVEN TWICE.

More than one field description specifies the same field name for different positions in the record.

SCRAMBLED FILES ILLEGAL.

IML handles only normal binary and symbolic files.

SEMANTIC ERROR IN RULES.

The syntax is correct, but there is an error in logic. The preceding error message clarifies the error.

SORT ABORTED.

The SORT command has come to an abnormal halt.

START MUST BE NUMERIC OR . .

A field starting position must be a number or a period (.).

STARTING POSITION MUST BE 1 OR GREATER.

A field starting position must be greater than or equal to 1, or specified with a period (.).

STARTING POSITION MUST BE MOD 3+1.

A field starting position must be 1, a multiple of 3 plus 1, or a period (.) for binary 1, R, and D fields.

SYSTEM ERROR.

System error. Try again. Contact your Tymshare representative if the message occurs again.

TITLE GIVEN NOT FOUND.

The user specified a title that does not appear in the multiple description file.

TOO FEW FIELDS IN RECORD.

The description file does not contain any field descriptions.

TOO FEW LINES IN INPUT.

Creating L type variable length records, there are not enough Control V–carriage return combinations in the record just entered.

TOO MANY 'BY' FIELDS.

The BY clause in the command specifies more than 20 field names.

TOO MANY CHRS. FOR . . . field name.

The field contains more characters than the field length specified in the description file.

TOO MANY DECIMAL PLACES.

The user is attempting to enter a number with more decimal digits than stated in the field description.

TOO MANY DIGITS TO RIGHT OF DECIMAL FOR . . . field name.

The user is attempting to enter a number with more decimal digits than stated in the field description.

TOO MANY END OF RECORD CHARACTERS.

Creating S type variable length records, there is more than one terminator character in the record just entered.

TOO MANY FIELDS.

The maximum of 200 unique field names has been exceeded.

TOO MANY FIELDS IN RECORD.

The user has entered more fields than described in the description file.

TOO MANY KEY FIELDS, MAXIMUM IS 20.

No more than 20 field names may appear in the BY clause.

TOO MANY LINES IN INPUT.

Creating L type variable length records, there are too many Control V–carriage return combinations in the record just entered.

TOO MANY LITERALS.

The operation specifies more than 2440 characters of literals.

TRYING TO EXCEED MAXIMUM FILE SIZE.

The user is attempting to produce an output file that is larger than the maximum file size permitted.

UNSORTED DATA FILE HAS DISAPPEARED.

While SORT was executing, the unsorted data file was deleted by someone else.

VALUE NOT NUMERIC IN . . . field name.

The specified field is defined as numeric, and the user is attempting to enter something other than a legal number.

WHAT?

The information entered for the previous command or prompt is not understood. Reenter the response or command again.

WRITE ERROR ON file name.

System error. Try again.

WRONG FILE TYPE FOR FILE file name.

The description file specifies binary, and the file is symbolic; or the description file specifies symbolic, and the file is binary.

ZERO DIVIDE.

An operation results in an attempt to divide by zero. IML continues the operation, using zero as the result of that division.

INDEX

NOTE: Page numbers that appear in boldface type refer to those pages where the listed item receives the most detailed discussion.