5280

IBM

SL23-0032-1
File No. S5280-24

Licensed
Program

IBM 5280 Distributed Data System

COBOL Programmer's Guide

Program Numbers: 5708-CB1(OS/VS)
5708-CB2(DOS/VSE)

SL23-0032-1
File No. S5280-24

Licensed
Program

IBM 5280 Distributed Data System

COBOL Programmer's Guide

Program Numbers: 5708-CB1(OS/VS)
5708-CB2(DOS/VSE)

Second Edition (February 1981)

This is a major revision of, and obsoletes, SL23-0032-0.

Changes are periodically made to the information herein; before using this publication in connection with the operation of IBM systems, be sure you have the latest edition and any technical newsletters.

This edition applies to version 01, modification 00 of the IBM 5280 COBOL OS/VS Host Compiler and Library (program number 5708-CB1), and the IBM 5280 COBOL DOS/VSE Host Compiler and Library (program number 5708-CB2), and to all subsequent versions and modifications until otherwise indicated in new editions or technical newsletters.

Use this publication only for the purposes stated in the section "To The COBOL Programmer".

To the COBOL Programmer

Before Using This Guide

Before using this guide, you should be aware of the topics covered, the experience and knowledge you should have to use it, and the other IBM 5280 and host system manuals you will need. It is the purpose of this section to give you this information; reading the text that follows may save you time.

Scope of the Guide

To use this guide, you should be either experienced or trained in the COBOL language. The guide is not intended as a tutorial for a beginner wishing to learn the COBOL language. (The objectives of the guide are given in a later section.)

The IBM 5280 host compilers for OS/VS and DOS/VSE convert source programs written in COBOL into load modules executable on an IBM 5280 system.

The information in this guide, together with that in the *IBM 5280 COBOL Language Reference*, GL23-0031, should provide you with most of the information you need to write COBOL programs and compile them on the host system.

The guide also provides information on executing a COBOL program on the IBM 5280. You will need additional information that is to be found in IBM 5280 publications (in, for example, the *Message Manual*). A list of these manuals and their order numbers is at the end of this chapter.

Organization of the Guide

The information in this guide is presented in the order a 5280 COBOL program might be coded, compiled, executed, and debugged:

- Chapters 1 through 6 provide rules, guidelines, and examples for writing transaction I/O, data communications, and printer and other I/O.
- Chapter 7 provides information on the procedures in the host system for compiling your source module. This chapter also describes the compiler options and how to specify them when you compile a program
- Chapter 8 contains 5280 system information relating to the execution of a COBOL program; for example: allocating data sets, loading a COBOL program, error messages issued during execution, factors affecting performance, etc.
- Chapter 9 describes compiler and language facilities you can use to debug a program.

Refer to the "Contents" or "Index" for a more detailed listing of topics.

Objectives of the Guide

The objectives of the guide and how to use it are given in the sections that follow.

Writing A Work Station Application

Three chapters provide information on writing interactive applications using the 5280 keyboard and other system functions:

- Chapter 1 introduces transaction I/O, the Data Definition Statements (DDS), and the 5280 keyboard. Transaction I/O together with the formatting and editing capabilities of DDS, provide flexibility and ease in writing applications that interact with a work station.
- Chapter 2 describes how to implement your screen designs with DDS and gives the detailed rules for coding DDS.
- Chapter 3 describes how to code the COBOL statements for transaction I/O.

Data can also be written to a work station using a DISPLAY statement or with SEQUENTIAL I/O; these two methods are discussed in Chapter 6.

Data Communications

The exchange of data between two systems over a communications link using IBM 5280 COBOL programs is called *data communications.* Chapter 4 covers data communications and has the following information:

- A summary of COBOL data communications capabilities
- The IBM 5280 facilities needed to execute a COBOL program
- The coding information necessary to write a data communications program.
- An example of a COBOL program using the data communications facilities

Diskette I/O

Chapter 5 describes how to write a program using the I/O facilities that allow the interchange of data between a program and IBM 5280 diskettes. Subjects covered in this chapter include:

- Sequential, relative, and indexed file organizations, their differences and uses
- The random and sequential access methods
- Step-by-step guidelines for writing the statements required to process sequential, relative, and indexed file organizations. The use of sequential and random access, where applicable, is discussed in relation to each of the file organizations.
- I/O error processing options

Printer and Other I/O

Chapter 6 provides information on:

- Writing data to a printer using SEQUENTIAL I/O
- Using the DISPLAY and ACCEPT verbs in exchanging data between an operator at a work station screen and a COBOL program
- Using SEQUENTIAL I/O in the exchange of data between an operator at a work station and a COBOL program.

Compiling a COBOL Program on the Host System

Chapter 7 describes the procedures available for compiling a source program; it also provides information that can be used for modifying the procedures at your installation if necessary. Information is given on the data sets required, the related job control language, and the options available for compilation.

Executing a COBOL Program on an IBM 5280 System

Chapter 8 provides a guide to loading and executing a COBOL program on the IBM 5280 system. Subjects covered include:

- Manuals and other documentation required by the operator
- Rules for creating COBOL data sets
- The COBOL-initiated prompts and how to respond to them
- Improving performance
- How to interpret the status line in regards to keyboard, device, and COROL execution errors

Executing Jobs in Sequence on an IBM 5280 System

The COBOL job-to-job facility allows you to pass control from your program to another COBOL program or any other program (for example, a DE/RPG program, an assembler program, a system utility program, etc.). You achieve this linkage with a CALL to a COBOL library routine as described in Chapter 10.

Together with the PROMPT/NOPROMPT option described in Chapter 7, this facility permits you to control linkage among several programs with little or no operator action.

Debugging COBOL Programs

Chapter 9 discusses procedures and the facilities provided by the compiler and the COBOL languages for debugging COBOL programs. Coded examples of some of the debugging facilities provided by COBOL are included.

Related Publications

You should have the following publications available for reference when preparing and running COBOL applications:

References for Coding and Debugging COBOL Programs

- *IBM 5280 COBOL Language Reference*, GL23-0031
- *IBM 5280 COBOL Host Compilers Problem Determination Manual*, SL23-0043

Coding Aids for Transaction I/O

- *Data Description Specifications*, GX21-9362 (a pad of coding forms)
- *IBM Printer/Display Layout*, GX21-9174 (a pad of layout forms)

References for Executing a COBOL Program

- *IBM 5280 Operator's Guide*, GA21-9364
- *IBM 5280 Utilities Reference/Operations Manual*, SC21-7788
- *IBM 5280 System Concepts*, GA21-9352
- *IBM 5280 Message Manual*, GA21-9354
- *IBM 5280 Printer Operator's Guide*, GA21-9260
- *IBM 5280 Communications Reference Manual*, SC34-0247

General References for IBM 5280 System Information

- *IBM 5280 General Information*, GA21-9350
- *IBM 5280 Planning and Site Preparation Guide*, GA21-9351
- *IBM 5280 Master Index*, GA21-9356
- *IBM 5280 User's Setup Procedures*, GA21-9365
- *IBM 5280 Machine Verification Manual*, GA21-9357
- *IBM 5280 System Control Programming Reference/Operations Manual*, GC21-7824

Compilation Under OS/VS1

- *OS/VS1 JCL Reference*, GC24-5099
- *OS/VS1 JCL Services*, GC24-5100

Compilation Under OS/VS2

- *OS/VS2 JCL*, GC28-0692

Compilation Under DOS/VSE

- *DOS/VSE System Control Statements*, GC33-5376

Contents

Figures

Chapter 1. Transaction I/O — Introduction

Transaction I/O is an IBM extension to COBOL that gives flexibility in writing data to and reading data from a display screen. The purpose of this chapter is to introduce transaction I/O and provide some rules that must be followed in writing transaction I/O applications. Subjects covered include:

- Statements used to define display screens and what they can do in formatting the screen and editing data entered by an operator
- How the statements are compiled with the COBOL source program
- Steps to follow in writing an interactive application
- A summary of the 5280 keyboard and how some of the keys interact with your COBOL program

Defining Display Screens

The statements which you code to define display screens are called *Data Definition Statements* (DDS). The Data Definition Statements for a screen consist of one *record description statement* followed by one or more *field description statements.*

Uses of Data Definition Statements

Here are some examples of what you can do with the Data Definition Statements:

- Define the layout of the screen: the position of the fields that are to appear on the screen, the data, if any, that is to appear in the fields, and how the characters appear on the screen (highlight, blink, reverse image, underline, nondisplay, and column separators).
- Cause the data to be edited as the operator enters it. For example, you can define a field so that only alphabetic characters can be entered. If the operator tries to enter a numeric, the keyboard will lock and an error code will be displayed.
- Set the shift key on a character-by-character basis, so that it shifts either to the lower symbol or to the upper symbol automatically as the cursor enters each position in the field

The Copy Library and Data Definition Statements

The coded Data Definition Statements must be library text in a COBOL library. At compilation, the COBOL compiler brings them into your program and converts them into COBOL source statements.

The COBOL library is allocated by the facilities of the host system. The rules for defining the library for each of the host compilers are given in Chapter 7.

Copying Members

You must code a COPY verb, specifying the name of the library text, at an appropriate point in the Data Division of your COBOL program. At compilation, the compiler creates a record definition for each set of Data Definition Statements: for each record description statement, the compiler creates an 01 level group item; and, with certain exceptions, for each field description statement, it creates a 02 level data-item, as shown in figure 2.4 in Chapter 2.

In writing and reading information from the work station screen, you refer to the record definitions created from the Data Definition Statements in associated READ and WRITE statements. This is shown in Chapter 3 in figure 3.2.

General Rules for Data Definition Statements

Here are some guidelines and rules for preparing Data Definition Statements and entering them in a COBOL library:

1. You must prepare a record description statement, followed by at least one field description statement, for each screen image.
2. Some of the specifications you make in the record description statement can be taken as defaults by the field description statements that follow it. That is, if you write these items in a record description statement, you needn't write them in a field description statement.
3. Keep the record description statement and subsequent field description statement(s) for each screen image entirely within one library text.
4. If desirable, enter sets of data definition statements for multiple screen images within the same library text.

The COBOL Program and Data Definition Statements

The COBOL program treats all the information passed to it from one DDS-defined screen format as a single record. You can specify two kinds of fields:

- An *output* field (O-field) contains information that cannot be changed by the operator. For a normal output operation, the data in the field is supplied in the field description statement as a constant
- An *input/output* field (B-field) is both displayed (output) on the display screen and read (input) back into the program. The operator can enter new data over the data that is displayed by the program. This data can then be read back into the program.

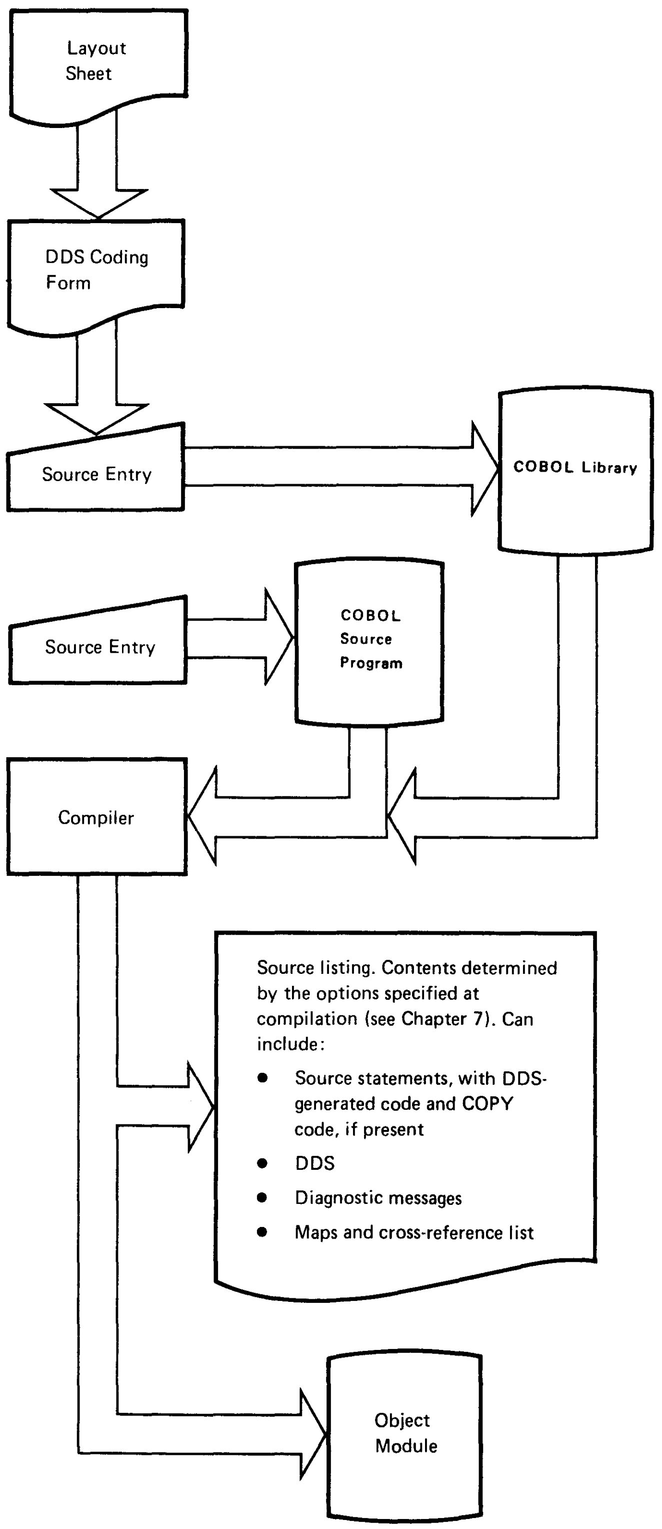
Both types of fields can be used in the same display screen formats.

By your entry in the field definition statement, you can also control how the data-items are defined in your program: either alphanumeric or numeric, signed or unsigned.

Steps in Writing an Interactive Program

The recommended steps in writing an interactive program, shown in figure 1.1, are:

1. Design the screen layouts which the operator is to interact with. Display screen layout sheets and coding forms are available to aid you in this task. Their order numbers are given in the first section of this book; their use is illustrated in the examples in Chapter 2.


Layout Sheet
DDS Coding Form
Source Entry
COBOL Library
Source Entry
COBOL Source Program
Compiler
Source listing. Contents determined by the options specified at compilation (see Chapter 7). Can include:
● Source statements, with DDS-generated code and COPY code, if present
● DDS
● Diagnostic messages
● Maps and cross-reference list
Object Module

1. Arrange fields on the layout sheet (order number GX21-9174) just as they will appear on the display screen.

2. Use the completed layout sheet as a guide for filling out the DDS coding sheet (order number GX21-9362).

3. Enter the DDS in a COBOL library.

4. Code source program; write COPY DDS statements as appropriate. Enter the COBOL source program.

5. Compile the source program.

Figure 1.1. Steps in preparing and compiling display screen formats

2. Define the screens with Data Definition Statements, placing them in a COBOL library. The rules for coding data definition statements are given in detail in Chapter 2.
3. Code the COBOL source program that is to read and write the screens. Information on the COBOL statements that perform transaction I/O is given in Chapter 3.
4. Compile the program. Information on compiler options and host job procedures is given in Chapter 7.

The Keyboard

This section describes the function and command keys, and their interaction with COBOL programs and the IBM 5280 system. The chapter is intended primarily for the COBOL programmer. For a tutorial, with a detailed explanation of the use of each key, the operator should look at the *Operator's Guide.*

Figure 1.2 shows a data entry keyboard.

The relative positions of most of the keys on the other keyboards available with IBM 5280, are the same as on the data entry keyboard. For a detailed description of all keyboards, see the *Operator's Guide.*

For descriptive purposes, the keys can be classified as *data keys, function keys*, and *command keys.*

- Data keys, when pressed, cause entered data to be processed by your COBOL program or the system.
- Function keys, when pressed, cause the IBM 5280 system to take some action.
- Command keys, when pressed, give control to your COBOL program. A code showing which key was pressed is passed to a control area, if specified, in the program.

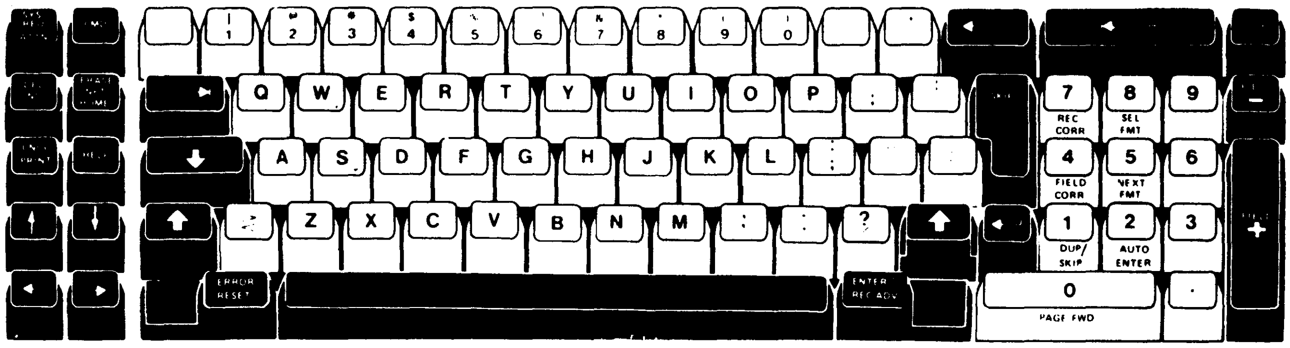


Figure 1.2 The data entry keyboard

In general, the function keys shown in figure 1.1 are colored dark, and the data keys are white. The operator presses a function key to enter the desired data or execute a desired function. Additionally, the top row of data keys can also be used as function keys or command keys. The next two sections describe the use of these keys and how they affect your COBOL programs.

Command and Function Key Codes

The following codes are returned in the work station control area (described under "Work Station Control Area" in Chapter 3) to your program when it receives control from the work station:

- *0* when either the Enter or Record Advance key is pressed.
- *99* when any other function key is pressed.

- One of the codes shown in figures 1.3 and 1.4 when the indicated command key is pressed. (Twenty-one (21) command keys are available for use with your COBOL programs.)

Command keys can make interaction between an operator and the program easier and more efficient. For example, in response to a program prompt, the operator can simply press one of the command keys instead of having to enter data. You can then determine the next logical action of your program based on the command key code returned to the program.

The operator selects a COBOL command key by (1) adjusting the shift to either lower case (alpha) or upper case (numeric), (2) pressing the CMD key, and (3) then pressing the desired command key.

The command keys as they appear on the data entry keyboard and the typewriter keyboard are shown in figures 1.3 and 1.4. The code passed to the program when each key is pressed is shown in the figure. The code is inserted in a data-item you define in your program. You relate the data-item to the CONTROL-AREA clause as described under "Work Station Control Area" in Chapter 3.

ALPHA SHIFT		NUMERIC SHIFT	
KEY	CODE	KEY	CODE
@	1	#	13
%	2	$	15
*	3		16
<	4	CORR	17
CORR	5	DUP	18
'	7	_	19
/	8	0	20
⟶	9	REL ADV	22
REC ADV	10	⟶	22
⟶|	11	SEL FMT	24
SEL FMT	12		

Figure 1.3. COBOL command keys and codes for data entry keyboards

LOWER CASE ⬇		UPPER CASE ⬆	
KEY	CODE	KEY	CODE
2	1	@	13
3	2	$	15
4	3	%	16
5	4	¬	17
6	5	&	18
8	7	*	19
9	8	(	20
0	9	_	22
-	10	+	23
=	11	⟵	24
⟵	12		

Figure 1.4. COBOL command keys and codes for typewriter keyboards

Function Keys

This section explains the use of the following function keys, and their interaction with the IBM 5280 system and your COBOL program:

Attention
Auto Dup/Skip
Auto Enter
Character Advance
Character Backspace
Character Delete
Character Insert
Cursor Left
Cursor Right
Duplicate
End-of-Job
Enter/Record Advance
Field Advance
Field Backspace
Field Exit
Field Exit Minus
Hexadecimal
Record Advance
Record Backspace
Reset
System Request

Two conditions are referred to frequently in the sections that follow: the *awaiting-field-exit* condition and the *awaiting-record-advance* condition. The following paragraphs explains what these terms mean.

In fields that require field exiting (for example, you have coded CHECK(FE) in the related field description statement), an awaiting-field-exit condition starts after the operator enters a character in the last position of the field. This condition is indicated on the screen by the cursor blinking in the last line of the field, and positions 15 and 16 of the Status Line contain 01.

An awaiting-record-advance condition exists after the operator enters a character into the last position of a record when the automatic enter function is disabled. The awaiting-record-advance condition is indicated on the screen by the blinking cursor beneath the last position of the record. The status line contains 00 in positions 15 and 16.

Attention

Pressing the Attention key permits the change of control between programs operating in background and foreground partitions. For example, while operating in the foreground, pressing the Attention key permits a program operating in a background partition to attach to the keyboard/display if operator action is required by that program. If action is required, the entire display is replaced by that program.

After operator action, the original program resumes control if the background program terminates. Otherwise, the operator can return control to the foreground program by pressing the Attention key.

For more information, see the *5280 System Concepts* manual under "Partition Interface".

Auto Dup/Skip

Pressing the Auto Dup/Skip key has the following effect:

- When the cursor enters a field, data is automatically duplicated in that field from the corresponding field in the previous record. You must have coded the CHECK(AD) keyword in the field description statement that describes the field.
- When the cursor enters a field, it automatically skips to the next field. You must have coded the CHECK(AS) keyword in the field description statement that describes the field.

See Chapter 2 for information on CHECK(AD) and CHECK(AS).

When automatic duplication or skip is in effect, a D in reverse image appears in position 21 in the status line at the top of the screen.

The operator can stop automatic duplication and skip by again pressing the Auto Dup/Skip key. Data can now be entered manually in all fields.

Auto Enter

When automatic enter (caused by pressing the Auto Enter key) is in effect, the operator doesn't have to press the Enter key after entering a character into the last position of the last field of a record. After the Auto Enter key is pressed, entering a character into the last position of a record has the same effect as pressing the Enter key. Control is then passed to the COBOL program for processing.

When automatic enter is in effect, an R appears (in reverse image) in position 23 in the status line at the top of the screen. To stop automatic enter, the operator again presses the Auto Enter key; a blank will now appear in position 23 of the status line.

Character Advance

Pressing the Character Advance key moves the cursor ahead one position in a field. The data in the position isn't changed. When the cursor is moved out of one field into the next, a field advance function is performed.

If automatic enter (described above) is in effect, advancing the cursor into the last position of the record has the same effect as pressing the Enter key.

If automatic enter isn't in effect, and the operator presses the Character Advance key when the cursor is in the last position of the record, an error will occur.

When awaiting-field-exit is indicated, pressing the Character Advance key has the same effect as pressing the Field Exit key.

Character Backspace

Pressing the Character Backspace key moves the cursor back one position. The data isn't changed. The key can be used to move the cursor from the first position of one field to the last position of the preceding field, unless automatic duplicate or automatic skip (as described under "Auto Dup/Skip" earlier in this chapter) is in effect. Then, the field is either automatically duplicated or skipped.

When awaiting-field-exit is indicated, pressing the Character Backspace key resets the condition. the cursor stays in the last position of the field; the operator can now enter a character.

When awaiting-record-advance is indicated, pressing the Character Backspace resets the awaiting-record-advance condition and puts the cursor in the last position of the last preceding field in which the operator can enter data.

Character Delete

Pressing the Character Delete key deletes the character at the cursor position. The characters within the field and to the right of the cursor shift to the left one position, and a blank is inserted in the rightmost position of the field. The cursor position doesn't change.

In a character check type field (when you code a C in the *Data Type* field of the field description statement that defines the field) the character delete function acts within subfields only. Subfields are adjacent character positions for which the same character type is specified in the parameter for the keyword SHIFT.

The operator cannot use the Character Delete key when you code a blank check — CHECK(BC) — or a mandatory fill — CHECK(MF) — in the related Data Definition Statements.

Character Insert

After pressing the Character Insert key, the operator can insert characters into a field at the current position of the cursor. (When the key is pressed a > appears in position 14 of the status line.)

After each character is entered, all data between the cursor and the right end of the field moves to the right one position. The operator can move the cursor within the field by using the Character Advance and Character Backspace keys.

After insertion, the operator must press the Reset key before exiting the field.

When a nonblank data character occupies the rightmost position of the field, an attempt to insert a character causes an error.

If you specify character check data in the field description statement that defines the field (a C in the *Data Type* field) any characters shifted to new positions must conform to the specifications for those positions that you code in the SHIFT keyword. Data can be shifted only within a range of positions for which the same character type is specified in the parameter for the SHIFT keyword. Data isn't shifted into a position for which a different data type is specified.

You can insert characters in a hexadecimal field when an *H* is specified in the *Data Type* field of the related field description statement. A hex entry character requires two keystrokes. The Insert key must be pressed before the first character is entered.

The operator cannot insert characters in fields for which you specify mandatory fill — CHECK(MF) — in the related field description statement.

Cursor Left

Pressing the Cursor Left key has the same effect as pressing the Character Backspace key.

Cursor Right

Pressing the Cursor Right key has the same effect as pressing the Character Advance key.

Duplicate

Pressing the Duplicate key copies characters into the field from the corresponding field in the previous record, unless the keyword CHECK(DD) (for duplicate disable) has been specified for the corresponding field description statement.

End-of-Job

Pressing the CMD key followed by the End-of-Job key closes all files and ends program execution. The operator is then prompted to load a program.

Enter/Record Advance

Pressing the Enter Key has the following effect:

- All edits you specify in the Data Definition Statements for all the fields on the screen are completed.
- If an error is found, a four-digit blinking error code appears in positions 8-11 in the status line at the top of the screen. See the section "Status Line" in Chapter 8. for a description of the different types of error codes that can appear and other related information in the status line.
- If no edit error is found, your COBOL program is given control.

Field Advance

When the operator presses the Field Advance key, the following takes place:

- Any edit function you specify in the Data Definition Statements for the field are started.
- If the edit is completed without error, the cursor moves to the next field where the operator can enter data.
- If the edit finds an error, a blinking error code will appear in positions 8-11 of the status line. The cursor remains in the right-most position of the field.

Pressing the field advance key before the cursor reaches the last position in the field doesn't affect any characters between the cursor and the end of the field.

If automatic enter (as described under "Auto Enter" earlier in this chapter) is in effect, pressing the Field Advance key when the cursor is in the last position of the record has the same effect as pressing the Enter key. That is, control is given to the COBOL program for processing.

Field Backspace

Pressing the Field Backspace key has the following effect:

- When the cursor is in the first position of the field, the cursor moves to the first position of the first previous field where automatic skip or automatic duplication isn't in effect. (Automatic skip and duplication are explained under "Auto Dup/Skip" earlier in this chapter.)
- When the cursor is in a position other than the first position, the cursor moves to the first position of the same field.

If awaiting-field-exit or awaiting-record-advance is indicated in the status line, pressing the Field Backspace key resets the status line and returns the cursor to the first position of the field.

When awaiting-record-advance is indicated, the field backspace function resets the condition and returns the cursor to the first position of the last preceding field in which the operator can enter data.

Field Exit

Pressing the Field Exit key has the following effect:

- For fields that contain signed numeric data (you coded an S in the *Data Type* field in the related Data Definition Statements), all data is shifted to the right, and unused positions are filled with zeros. The position immediately to the right of the field is set to blank to indicate a positive number.
- For fields in which data is to be right-adjusted (you specified CHECK(RZ), CHECK(RB) or CHECK(RL) in the related field description Statement), data to the left of the cursor is shifted to the right. Unused positions are filled with zeros or blanks, depending on your specification.
- If neither of the above is true, all positions to the right of the cursor are filled with blanks.

In all of the above cases, any other edits you specify in the Data Definition Statements are done. If no error is found, the cursor advances to the next field in which the operator can enter data.

If an error is found, a four-digit blinking error code appears in positions 8-11 in the status line at the tops of the screen. See the section "Status Line" in Chapter 8 for a description of the different types of error codes and other related information that can appear in the status line.

Field Exit Minus

Pressing the Field Exit Minus key affects only those fields in whose Data Definition Statements you have coded one of the following in the *Data Type* field:

D (digits only)
N (numeric shift)
S (signed numeric)
Y (numeric only)

Pressing the Field Exit Minus key has the same effect as pressing the Field Exit key, except the sign of the data in the field indicates a negative quantity.

In signed numeric fields, a minus sign is displayed in the position to the right of the field.

In fields for digits only, numeric only, or numeric data types when right-adjust is specified or awaiting-field-exit is indicated, the zone of the rightmost character (which must be a number) is changed to hexadecimal D.

The rightmost character in the field, signifying the negative sign, is displayed as follows:

Numeric	Digits Only/ Numeric Only
{ for 0	0* for 0
J for 1	1* for 1
K for 2	2* for 2
L for 3	3* for 3
M for 4	4* for 4
N for 5	5* for 5
O for 6	6* for 6
P for 7	7* for 7
Q for 8	8* for 8
R for 9	9* for 9

* Each digit will appear on the screen smaller than the standard size digit usually displayed and will have a bar just above it.

In fields for digits only, numeric only, or numeric data types without right-adjust specified and without awaiting-field-exit indicated, all positions from the cursor through the next-to-last position of the field are blanked, and the rightmost position is set to hexadecimal D0.

Hexadecimal

Pressing the Hexadecimal key allows the operator to enter a hexadecimal value. Two data keys in succession must be pressed for each hexadecimal character. The numbers 0 through 9 and the letters A through F are the only valid entries.

The character displayed above the cursor is the single character that results from the translation of the hexadecimal value entered. If the translation doesn't yield a displayable character, the value is displayed as ■. The hexadecimal representation of the characters entered by the operator is displayed in positions 18 and 19 in the status line at the top of the screen.

Record Advance

See "Enter/Record Advance".

Record Backspace

Pressing the Record Backspace key returns the cursor to the first position in the first field of the record where the operator can enter data manually.

Reset

By pressing the Reset Key, the operator can:

- Reset the status line and unlock the keyboard after an error is found.
- Cancel the effect of pressing the Command (CMD), Hexadecimal, and Insert keys.

Skip

Pressing the Skip key cause the remainder of the field to be filled with blanks. Any edits specified for the field are then done. If no errors are found, the cursor advances to the next field where the operator can manually enter data.

When awaiting-field-exit is indicated, either the Skip key or the Field Exit key can be pressed.

System Request

By pressing the System Request key, the operator temporarily stops the execution of the current program to load a program into another partition. Then, execution of the current program resumes.

If the new program is loaded into the same partition as the first program, execution of the first program stops.

Chapter 2. Transaction I/O — Screen Definition

This chapter provides the information necessary to define the screens used by a COBOL program in interactive processing between the program and the operator at the work station. The following topics are covered:

- An example of coded Data Definition Statements
- Coding Conventions
- Rules for coding the Data Definition Statements.

Example of Coding Data Definition Statements

The example in this section shows the steps followed in planning a screen format and coding the necessary Data Definition Statements. The same Data Definition Statements are used in the COBOL coding example shown figure 3.3 in Chapter 3.

For the purpose of the example, it's assumed that data entry operators entered information from a form like the following to update an employee master file:

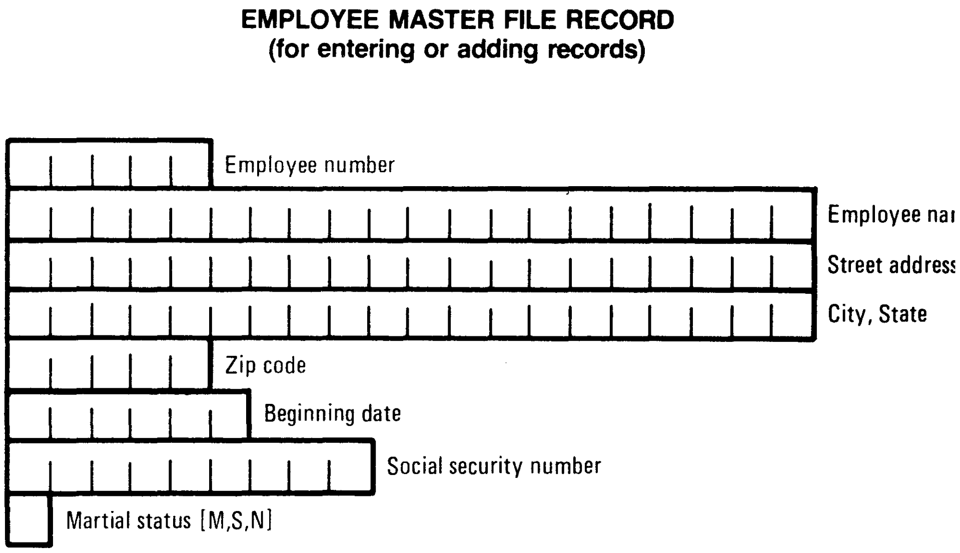
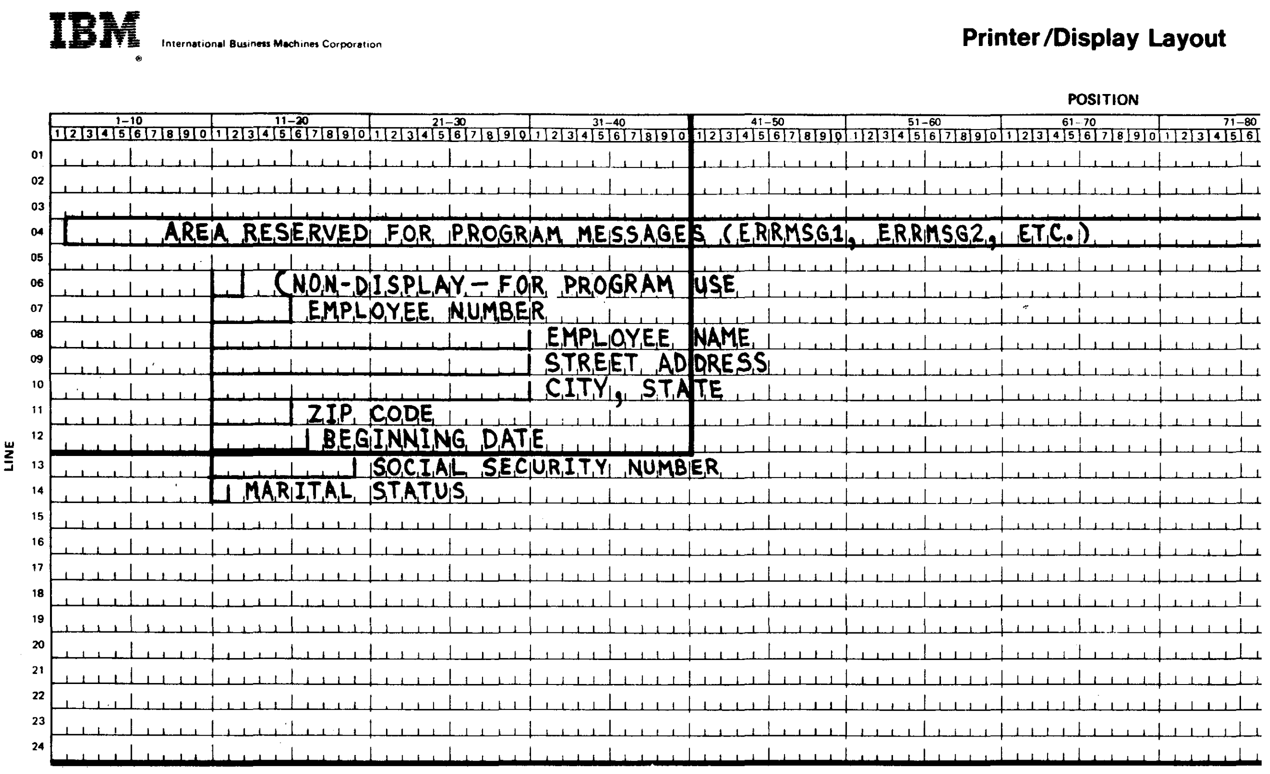


Figure 2.1. Sample form used by data entry operator

In the example, the screen is designed to resemble the sample form shown in the figure. The following editing characteristics will be defined for the fields (and explained in detail later):

- Data must be entered in all fields
- Certain fields (zip code, beginning date, and social security number) must be completely filled before the operator can move the cursor to the next field or enter the record.
- The beginning date in the previous record can be automatically entered in the beginning date field of the current record at the option of the operator.

Using the above information, the layout of the screen is planned on an *IBM Printer/Display Layout* form (order number GX21-9174) as shown below:



IBM International Business Machines Corporation

Printer/Display Layout

POSITION

LINE

```
04  AREA RESERVED FOR PROGRAM MESSAGES (ERRMSG1, ERRMSG2, ETC.)
06  (NON-DISPLAY – FOR PROGRAM USE
07  EMPLOYEE NUMBER
08  EMPLOYEE NAME
09  STREET ADDRESS
10  CITY, STATE
11  ZIP CODE
12  BEGINNING DATE
13  SOCIAL SECURITY NUMBER
14  MARITAL STATUS
```

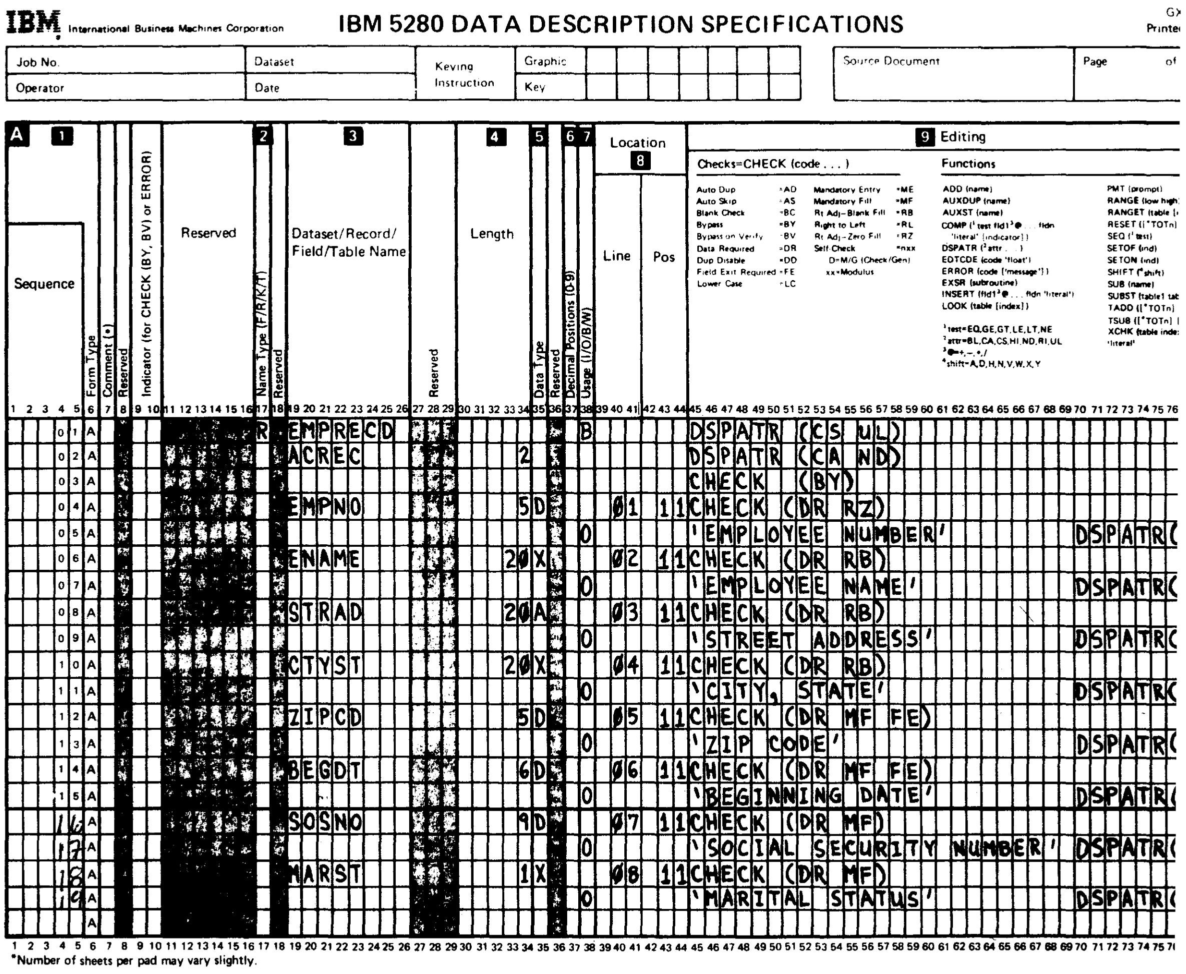
Figure 2.2. Printer/Display Layout form

Using the layout form makes it easier to see how the data will appear on the screen and to code the field description statements, in which you usually write the position on the screen where each field is to appear.

Keep in mind the size or sizes of the work station screens available at your location when designing the screens. They come in three sizes: 1920, 960, and 480. The dimensions for each size are as follows:

Size in Characters	Number of horizontal rows	Number of vertical columns
1920	24	80
960	12	80
480	6	80

Once the format of the screen has been planned, the next step is to write the Data Definition Statements. The *IBM 5280 Data Description Specifications* form, order number GX21-9362 can be used as an aid in coding the specifications. The data definition statements in figure 2.3 were written so that the screen format would look like the data entry form shown in figure 2.2.



IBM International Business Machines Corporation — IBM 5280 DATA DESCRIPTION SPECIFICATIONS

Sequence	Form Type	Name Type	Dataset/Record/Field/Table Name	Length	Data Type	Usage	Line	Pos	Editing (45–69)	Editing (70–76)
01	A	R	EMPRECD			B			DSPATR (CS UL)	
02	A		ACREC	2					DSPATR (CA ND)	
03	A								CHECK (BY)	
04	A		EMPNO	5	D		Ø1	11	CHECK (DR RZ)	
05	A					O			'EMPLOYEE NUMBER'	DSPATR(
06	A		ENAME	2Ø	X		Ø2	11	CHECK (DR RB)	
07	A					O			'EMPLOYEE NAME'	DSPATR(
08	A		STRAD	2Ø	A		Ø3	11	CHECK (DR RB)	
09	A					O			'STREET ADDRESS'	DSPATR(
10	A		CTYST	2Ø	X		Ø4	11	CHECK (DR RB)	
11	A					O			'CITY, STATE'	DSPATR(
12	A		ZIPCD	5	D		Ø5	11	CHECK (DR MF FE)	
13	A					O			'ZIP CODE'	DSPATR(
14	A		BEGDT	6	D		Ø6	11	CHECK (DR MF FE)	
15	A					O			'BEGINNING DATE'	DSPATR(
16	A		SOSNO	9	D		Ø7	11	CHECK (DR MF)	
17	A					O			'SOCIAL SECURITY NUMBER'	DSPATR(
18	A		MARST	1	X		Ø8	11	CHECK (DR MF)	
19	A					O			'MARITAL STATUS'	DSPATR(
	A									

*Number of sheets per pad may vary slightly.

Figure 2.3 Data Definition Statements Coding Example

The statements in figure 2.4 show what the Data Definition Statements in figure 2.3 look like after they are copied and compiled.

```
COPY-ID    STMT.#   SEQ.#   A...B.........COBOL SOURCE STATEMENTS...
              22               COPY DDS-EMPRECD.
EMPRECD       22    000001 01       EMPRECD.
EMPRECD       23    000002      02  ACREC      PIC X(00002).
EMPRECD       24    000004      02  EMPNO      PIC X(00005).
EMPRECD       25    000006      02  ENAME      PIC X(00020).
EMPRECD       26    000008      02  STRAD      PIC X(00020).
EMPRECD       27    000010      02  CTYST      PIC X(00020).
EMPRECD       28    000012      02  ZIPCD      PIC X(00005).
EMPRECD       29    000014      02  BEGDT      PIC X(00006).
EMPRECD       30    000016      02  SOSNO      PIC X(00009).
EMPRECD       31    000018      02  MARST      PIC X(00001).
```

Figure 2.4 Data Definition Statements as compiled by the COBOL compiler

The following text explains some of the statements in figure 2.3; later sections in this chapter give detailed rules for coding each field.

1 The *Sequence Number* field (columns 1-5). The number in this field appears in the source listing under the SEQ.# column, as shown in figure 2.4.

2 The *Name Type* field (column 17). For record description statements, this field contains an R. For field description statements, this field contains a blank.

3 The *Record Name* field (columns 19-26 for record description statements) or the *Field Name* field (columns 19-24 for field description statements). These names (EMPRECD, ACREC, EMPNO, etc.) will appear as the 01 and 02 level data items in the COBOL program, as shown in figure 2.4.

4 The *Length* field (columns 30-34 for field description statements only.) This field specifies the length of the field on the screen, and the size of the 02 level data-item in the COBOL program, as also shown in figure 2.4.

5 The *Data Type* field (column 35 for field description statements only). This field defines the data type and keyboard positioning, in this example *A* for alphabetic shift, *D* for digits only, and *X* for alphabetic only.

6 The *Decimal Positions* field (column 37 for field description statements only). This field determines whether the corresponding data-item in the COBOL program will be compiled as alphanumeric or numeric. If blank, the data-item will be alphanumeric; if any number between 0 and 9 (indicating a decimal position), the data-item will be numeric.

In the example, the *Decimal Positions* fields all contained blanks. As shown in figure 2.4, this created alphanumeric (X) data-items for all the fields.

7 The *Usage* field (column 38). This field defines whether or not the operator can enter data in the field. If it is *B* (for both input and output) the operator can enter data into the field, and the program can write data from the field to the screen. If it is *O* (for output only), the program can only write data to the screen using literal statements as will be shown later.

☞**Note 1**: You can access in your COBOL program only those fields defined with usage *B*. As shown in figures 2.3 and 2.4, only *B* fields are compiled as data-items.

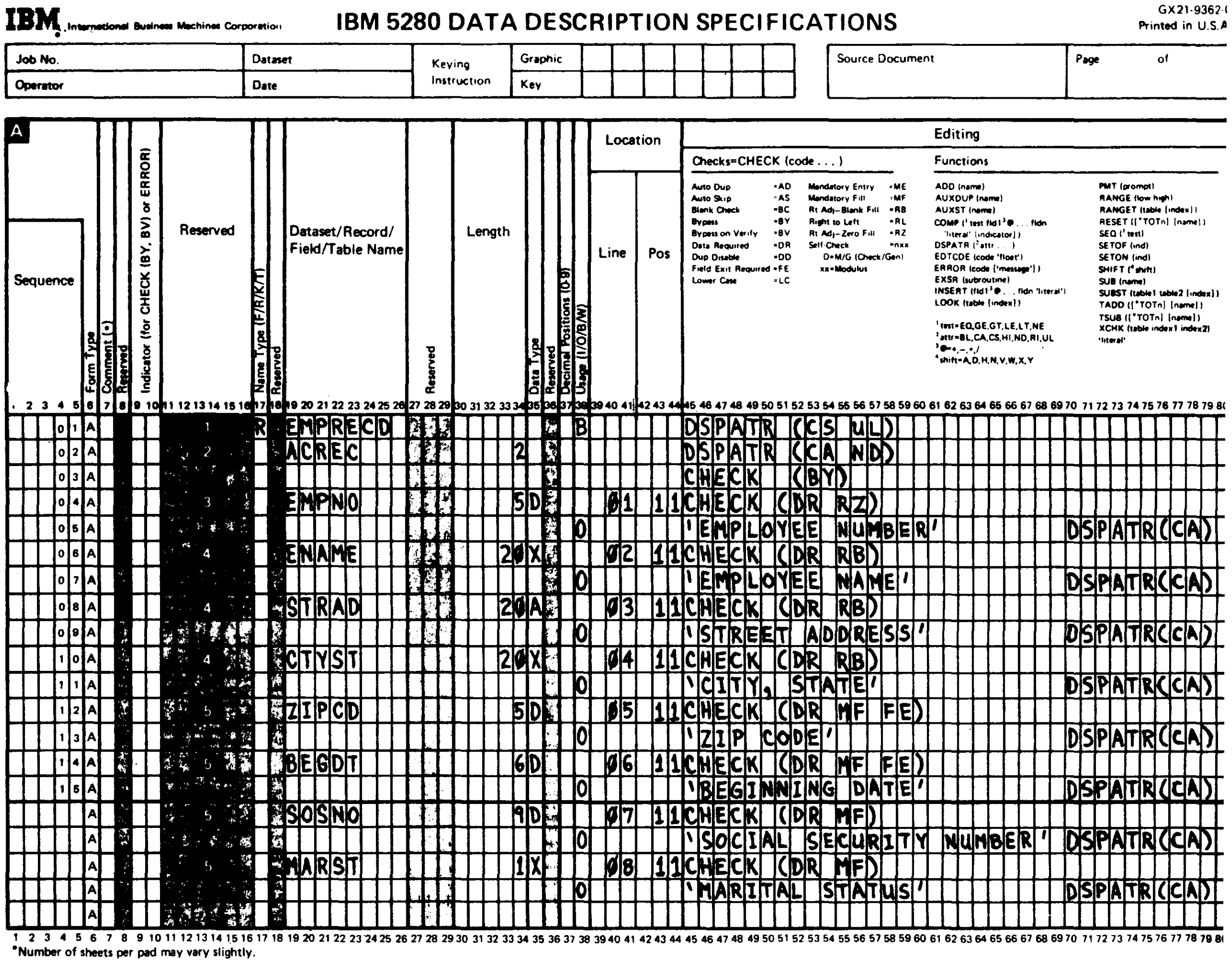
☞**Note 2**: *W* and *I* fields (shown on the coding form) aren't valid fields for COBOL programs.

8 The *Location* field (columns 39-44 for field description statements only). This field specifies the line and column number where the field is to appear on the screen.

The physical line number on the screen where the field will appear depends on (1) the value you code in columns 39-41 and (2) the value you code in the STARTING AT LINE clause in the associated WRITE statement in your program. See the section "Location Field" for the rules on how these two values affect the placement of fields on the screen.

The value you code in columns 42-44 determine the horizontal placement of the field.

9 The *Editing* field (columns 45-80). You write the keywords that control the editing, field attributes, and keyboard conditioning in this field. Some of these keywords used in this example will be explained in the following text. Refer to figure 2.5 while reading the text.



IBM International Business Machines Corporation — IBM 5280 DATA DESCRIPTION SPECIFICATIONS — GX21-9362 Printed in U.S.A

Job No.	Dataset	Keying Instruction	Graphic	Source Document
Operator	Date		Key	Page of

A — Sequence (1–5); Form Type (6); Comment (*) (7); Reserved (8); Indicator (for CHECK (BY, BV) or ERROR) (9–10); Reserved (11–16); Name Type (F/R/K/T) (17); Reserved (18); Dataset/Record/Field/Table Name (19–26); Reserved (27–29); Length (30–34); Data Type (35); Reserved (36); Decimal Positions (0-9) (37); Usage (I/O/B/W) (38); Location: Line (39–41), Pos (42–44); Editing (45–80)

Checks=CHECK (code . . .)

Auto Dup	=AD	Mandatory Entry	=ME
Auto Skip	=AS	Mandatory Fill	=MF
Blank Check	=BC	Rt Adj–Blank Fill	=RB
Bypass	=BY	Right to Left	=RL
Bypass on Verify	=BV	Rt Adj–Zero Fill	=RZ
Data Required	=DR	Self-Check	=nxx
Dup Disable	=DD	D=M/G (Check/Gen)	
Field Exit Required	=FE	xx=Modulus	
Lower Case	=LC		

Functions

ADD (name)	PMT (prompt)
AUXDUP (name)	RANGE (low high)
AUXST (name)	RANGET (table [index])
COMP ([1]test fld1[3]@ . . . fldn 'literal' [indicator])	RESET ([*TOTn] [name])
DSPATR ([2]attr . . .)	SEQ ([1]test)
EDTCDE (code 'float')	SETOF (ind)
ERROR (code ['message'])	SETON (ind)
EXSR (subroutine)	SHIFT ([4]shift)
INSERT (fld1[3]@ . . . fldn 'literal')	SUB (name)
LOOK (table [index])	SUBST (table1 table2 [index])
	TADD ([*TOTn] [name])
	TSUB ([*TOTn] [name])
	XCHK (table index1 index2)
	'literal'

[1]test=EQ,GE,GT,LE,LT,NE
[2]attr=BL,CA,CS,HI,ND,RI,UL
[3]@=+,–,*,/
[4]shift=A,D,H,N,V,W,X,Y

```
   01A          R EMPRECD            B      DSPATR (CS UL)
   02A            ACREC          2          DSPATR (CA ND)
   03A                                      CHECK  (BY)
   04A            EMPNO         5D    Ø1 11CHECK (DR RZ)
   05A                               0      'EMPLOYEE NUMBER'        DSPATR(CA)
   06A            ENAME        2ØX    Ø2 11CHECK (DR RB)
   07A                               0      'EMPLOYEE NAME'          DSPATR(CA)
   08A            STRAD        2ØA    Ø3 11CHECK (DR RB)
   09A                               0      'STREET ADDRESS'         DSPATR(CA)
   10A            CTYST        2ØX    Ø4 11CHECK (DR RB)
   11A                               0      'CITY, STATE'            DSPATR(CA)
   12A            ZIPCD         5D    Ø5 11CHECK (DR MF FE)
   13A                               0      'ZIP CODE'               DSPATR(CA)
   14A            BEGDT         6D    Ø6 11CHECK (DR MF FE)
   15A                               0      'BEGINNING DATE'         DSPATR(CA)
     A            SOSNO         9D    Ø7 11CHECK (DR MF)
     A                               0      'SOCIAL SECURITY NUMBER' DSPATR(CA)
     A            MARST         1X    Ø8 11CHECK (DR MF)
     A                               0      'MARITAL STATUS'         DSPATR(CA)
     A
```

*Number of sheets per pad may vary slightly.

Figure 2.5. Data Definition Statements Coding Example

1 EMPRECD is a record description statement.

1. The *B* in the *Usage* field is the the default for the *usage* field for all subsequent field statements. The *B* lets all fields in the record be used for both input and output. That is, the program can write data to the screen from them, and the operator can enter data into them on the screen. The *B* can be overridden in a subsequent field description statement, as will be shown.
2. DSPATR (for *display attribute*) assigns two default attributes to the field description statements: all fields will have column separators (CS) and will be underlined (UL) unless specifically overridden, as will be shown.

2 ACREC is a two-byte field that will not appear on the screen.

The statements in the *Editing* field are an example of how to suppress a field not intended for viewing by the operator. (In this example, ACREC is used by the program to determine whether the record is current or inactive.) Here's how this is achieved in the example:

1. In the DSPATR keyword, CA (for cancel) cancels the underline (UL) and column separators (CS) specified in the preceding record description

statement (EMPRECD). ND (for nondisplay) prevents any data in the field from appearing on the screen.

2. CHECK(BY) — BY stands for bypass — causes the cursor to skip the ACREC field and be positioned at the start of the next field (EMPNO) immediately after the record is written to the screen.

   Note that CHECK(BY) must be specified in a *secondary line* as explained later in this chapter under "Coding Conventions" and shown in figure 3.3.

3. Because the field is nondisplay, it wasn't necessary to specify a location.

3 EMPNO is a five-digit field for the employee number. Here is a summary of the specifications:

1. The *D* in the *Data Type* field allows the operator to enter only digits. If the operator tries to enter nondigits, the keyboard will lock and an error code will appear in the status line at the top of the screen. The operator can then refer to the *5280 Messages Manual* for an explanation of the message and the recommended action to correct the error.

2. The 01 in the *Location* field indicates the relative line number the field is to appear on the screen in relation to the STARTING AT LINE clause as discussed earlier. The number *11* causes the field to start in column 11 on the screen.

3. The *DR* (for data required) following the CHECK keyword means that the operator must enter at least one character (in this example, a digit) before exiting the field. Otherwise, the keyboard will lock and an error code will appear in the status line.

   The *RZ* (for right adjust with zeros) creates two conditions: (1) the operator must press the Field Exit key to go to the next field and (2) when the exit key is pressed, all digits are shifted to the rightmost positions; any unused positions to the left are filled with zeros.

4 The logic of the specifications for ENAME, STRAD, and CTYST is the same as for EMPNO with the following exceptions:

- The *RB* following CHECK causes a right adjust with blanks instead of zeros.
- The *X* in the *Data Type* field for ENAME and CTYST allows the operator to enter only alphabetic characters.
- The *A* in the *Data Type* field for STRAD allows the operator to enter both digits and alphabetic characters.

The fields with an *O* specified in the *Usage* field (column 36) are fields that can be used for output only. That is, the program can write the data enclosed in single quotes (') in the *Editing* field to the screen, but the operator cannot write data into the fields on the screen.

☞Note the following:

1. The *Field Name* field for output fields must be blank.
2. The fields needn't have a field location specified; they will appear on the screen after the preceding field.
3. The CA (for cancel) parameter following the DSPATR keyword cancels the underline and column separator attributes for these fields.
4. As shown in figure 2.4, output fields do not appear as data-items in the source listing.

5 Note the effect of CHECK parameters on the ZIPCD, BEGDT, SOSNO, and MARST fields:

1. *FE* forces the operator to press the Field Exit key to exit the ZIPCD and BEGDT fields.
2. Together, *DR* (for data required) and *MF* (for mandatory fill) force the operator to fill all available positions in each field before an exit can take place as follows:

- *DR* specifies that at least one nonblank character must be entered in the field.
- *MF* specifies that if a character is entered into one position of the field, all positions must be filled.

If the operator attempts to exit the field without filling the positions with the characters specified in the *Data Type* field, the keyboard will lock and an error code will appear in the status line at the top of the screen.

Coding Conventions

Primary, Continuation, and Secondary Lines

You can code two different types of lines for field and record statements:

- A *primary line*, which contains the fields, both required and optional, in columns 1-44 and, starting in column 45, the editing field, which contains keywords, keyword parameters, and literals.
- One or more *continuation lines*, which extend the editing field (begun on a preceding primary or continuation line).

  The primary line, or continuation line that comes before another continuation line, must have a + character or a - character as the last nonblank character. The next section explains the differences between these characters.

Field description statements can contain, in addition to primary and continuation lines just described above, one or more *secondary lines*. A CHECK keyword (whose functions are explained later in this chapter) containing a BY parameter, must always begin on a secondary line. The rules for coding a secondary line are:

- A secondary line must always follow a primary line or a continuation line.
- The primary or continuation line that comes before a secondary line must not have a + character or a - character specified as the last character.
- A secondary line must begin with a keyword.
- A secondary line can have continuation lines under the same rules specified for primary lines.

Comment statements, which contain an asterisk (*) in column 7, can be placed where desired between primary, secondary, and continuation lines.

Continuation Characters

If a literal, keyword, or keyword parameter doesn't fit on one line, you can continue it on the next line by coding a plus character (+) or a minus character (-) as the last nonblank character in the line.

A plus (+) character indicates that the compiler is to ignore any blanks before the first nonblank character in the continuation line that follows.

A minus (-) character causes the compiler to include any blanks before the first nonblank character in the continuation line that follows. The compiler always includes any blanks preceding the + or - continuation character.

Keywords

Keywords represent attributes and functions. Most keywords must be accompanied by a string of one or more parameters that further define the attributes or function. The string of parameters must follow the keyword and must be enclosed within a pair of parentheses.

The general syntax for a keyword and its parameter string is:

KEYWORD (*parameter string*)

When the parameter string contains two or more parameters, each consecutive parameter must be separated from the preceding parameter by at least one blank.

Two or more keywords can be specified on a single line. Any keyword that is specified without a parameter string must be separated from the following keyword by at least one blank.

Within these guidelines, the following three lines convey identical meanings to the compiler.

```
CHECK(DR FE RB)DSPATR(CS)
CHECK (DR FE RB) DSPATR(CS)
CHECK ( DR FE RB ) DSPATR ( CS )
```

In the above three lines, the only required blanks are between the parameters following the CHECK keyword. Whenever a keyword and its parameters aren't started and completed on the same line, the use of a continuation character is required. See *Continuation Characters*, earlier in this chapter.

Constants

Character constants consist of any combination of characters, including blanks. Character constants must be enclosed in apostrophes. An apostrophe required as data within the constant must be represented by two apostrophes. If necessary, you can continue a constant on one or more continuation lines.

Comment Statements

Comment statements allow you to insert comments among the Data Definition Statements, providing a means to document their logic if necessary.

Comment statements can be placed anywhere in the source statement sequence. They are ignored by the compiler. You write a comment by entering an asterisk (*) in column 7. The remaining positions of the line (columns 8 through 80) are then ignored by the compiler and are available for comments.

Record Description Statements

Record description statements name the record and describe characteristics that apply to the entire record.

Sequence Number Field (Columns 1 through 5)

Enter the statement sequence number, if desired. The sequence number, if entered, will appear next to the 01 level data-name in the source listing put out by the compiler. This name is defined in the *Record Name* field (columns 19-26) described below.

Sequence numbers in source statements are optional. They can be useful when you want to change, add, delete, or relocate a source statement. The only restriction is that the first two columns of the *Sequence* field cannot contain **.

Form Type Field (Column 6)

Enter an A in the *Form Type* field.

Name Type Field (Column 17)

The letter R is required in the *Name Type* field for record description statements.

Record Name Field (Columns 19 through 26)

A name is required on a record description statement. The name must not be more than eight (8) characters long. In the source listing put out by the compiler, the name specified will appear as a 01 level record-name and group item.

The rules for defining names are the same as for COBOL record-names (except for the limit of eight characters): valid characters in the *Record name* field are A through Z, 0 through 9, and - (the hyphen); the name must begin with an alphabetic character. See the *COBOL Language Reference* if you need more details on these rules.

Usage Field (Column 38)

The entry you make in the *Usage* field supplies a default value for subordinate field statements which contain a blank in column 38.

Valid entries in the *Usage* field are:

O	establishes a default of output only for subordinate field description statements.
B	establishes a default that is for both input and output in subordinate field description statements.
blank	indicates no default; an entry must be made in the *Usage* field in each subsequent field description statement.

Output (O) Fields

For those fields defined for output (you enter an O in the *Usage* field):

1. The operator cannot enter data into the field.
2. You must specify a nonnumeric literal in the *Editing* field starting in column 45. The literal can have any character in the EBCDIC set and must be enclosed within single quotes (`'literal'`). Any single quote used as part of the character string must be coded twice (' ').

The literal appears on the screen when your COBOL program issues an associated WRITE statement. The use of literals is shown at **3** through **5** in Figure 2.5 earlier in this chapter.

3. You must leave the *Field Name* field (columns 19-24 of the field description statement) blank.

Both Input and Output (B) Fields

For those fields defined for both input and output (you enter a B in the *Usage* field):

1. The operator can enter data on the screen in fields defined by the statements.
2. The operator can duplicate data from the corresponding positions in the preceding record by pressing the Duplicate key, unless you specify CHECK(DD) as explained later in this section.
3. Data can be automatically duplicated in the field when the the operator has pressed the Auto Dup/Skip key, and you have coded the CHECK(AD) keyword and parameter in the editing field of the associated field description statement. The data is automatically duplicated from the corresponding positions in the previous record when the cursor enters the field.

When using a field for both input and output, consider the following when writing your COBOL program:

1. The field description statements with a *B* specified in the *Usage* field define 02 level elementary items in the COBOL program.
2. When a WRITE is executed, any data in the associated record will be written to the screen. You should therefore ensure that such fields are either initialized with blanks or the data you want before the WRITE is executed.
3. The operator can write over any data appearing in the field.

To Prevent Entry of Data in a B-Field

To prevent the entry of data in a B-Field, use the CHECK(BY) keyword described later in this chapter. With this keyword, you can cause the cursor to bypass a field on the screen either conditionally or unconditionally.

Editing Field (Columns 45 through 80)

The entry you make in the *editing field* applies to subordinate field statements which don't contain overriding keywords in columns 45-80.

You can specify the following keywords in a record description statement:

Keyword	**Description**
CHECK	Keyboard level edits
DSPATR	Display attributes

CHECK(DD)

The CHECK(DD) keyword prevents the operator from using the DUP key for fields defined by subordinate field description statements. If CHECK(DD) is specified in the record description statement, you cannot override it in a subordinate field description statement.

CHECK allows a wider range of edits when specified in the field description statement, as described under "Field Description Statement" below.

DSPATR (attributes)

With the DSPATR (for *display attributes*) keyword, you can control the display attributes on the screen. The attributes specified by this keyword are combined with those specified in the field description statement (if any) to establish the display attributes to be used for a field.

See the explanation of DSPATR in the section "Field Description Statement" for a description of the attributes and the rules for specifying them.

(This page is intentionally left blank.)

Field Description Statements

A field description statement can be made up of primary, continuation, and secondary lines as described under "Coding Conventions" earlier in this chapter.

Sequence Number Field (Columns 1 through 5)

Enter the statement sequence number, if desired. The sequence number, if entered, will appear next to the 02 level elementary item in the source listing put out by the compiler. You define this name in the field name field (columns 19-24) described below.

Sequence numbers in source statements are optional. They can be useful when you want to change, add, delete, or relocate a source statement. The only restriction is that the first two columns of the *Sequence* field cannot contain **.

Form Type Field (Column 6)

Enter an A in the *Form Type* field.

Indicator Field (Columns 9 and 10)

Leave this field blank *except* when you want to cause the cursor to conditionally bypass a field and specify the CHECK(BY) keyword for this purpose. You specify CHECK(BY) in the *Editing* field as described later in this chapter.

Code any number from 1 through 99. Single digit numbers (1 through 9) must be right-adjusted with a leading blank.

The indicator you specify is associated with a boolean data-item you code in your program.

For the rules to follow in coding the INDICATOR clause with a data-item, see the *5280 COBOL Language Reference* manual under "OCCURS Clause" and "INDICATOR Clause".

For the format of the INDICATOR clause in the WRITE statement, see Chapter 3 of this manual or the *5280 COBOL Language Reference* manual under "WRITE".

For an example of the COBOL statements and the Data Definition Statements using conditional bypass with indicators, see Chapter 3 under "INDICATOR Example".

Field Name Field (Columns 19 through 24)

Leave this field blank when you are defining a field that is to be used for output only (column 38 the *Usage* field, has an O in it).

The *Field Name* field must be a valid COBOL data-name not more than eight (8) characters long. The name will appear as a 02 level elementary item in your COBOL program.

If you leave this field blank when the statement defines a field that is to be used for both input and output (column 38, the *Usage* field, has a *B* in it), the resulting 02 level data-item will appear as FILLER in your COBOL program.

The rules for defining names are the same as for COBOL record-names: valid characters in the *Field name* field are A through Z, 0 through 9, and - (the hyphen); the name must begin with an alphabetic character. See the *COBOL Language Reference* if you need more details on these rules.

Length Field (Columns 30 through 34)

Leave this field blank when you are defining a field that is to be used for output only (column 38 the *Usage* field, has an O in it).

You must specify a length when a data field is used for both input and output (the *Usage* field, column 38, has a B in it). The following rules apply:

The maximum length of the field depends on the type of field you specify:

1. Character fields can contain up to 256 positions; field description statements, as noted earlier, are compiled as 02 level data-items in your COBOL program.
2. Numeric fields can contain up to 18 positions.

Only the digits 0 through 9 are allowed in the *Length* field; either leading zeros or leading blanks are acceptable. The entry in the *Length* field must be right adjusted.

Data Type Field (Column 35)

Leave this field blank when:

- You code CHECK(BY) in the *Editing* field.
- This statement defines a field that is to be used for output only (the *Usage* field, column 38, has an *0* in it).

You must specify a data type when the field is used for both input and output (the *Usage* field, column 38, has a B in it) unless you specify CHECK(BY) in the *Editing* field. (The CHECK keyword is explained later in this chapter.)

Data-Item Class in COBOL Program

The class (numeric or alphanumeric) of the data-item compiled in the COBOL program depends on you entry in both the *Data Type* field and the *Decimal Positions* field, as shown in the following table:

Data Type field contents:	Decimal Position field contents:	Data class of field in COBOL program after compilation:
A, C, D, H, N, S, V, W, X, or Y	blank	alphanumeric (X)
A, C, H, V, W, X, or Y	0 - 9	unsigned numeric (9V9)
D, N, or S	0 - 9	signed numeric (S9V9)

Data Type Field Entries

You can make the following entries in the *Data Type* field:

Entry	Condition
A	Alphabetic shift – Any character can be entered. The shift is positioned to the lower symbol on each key on all keyboards. The operator can use the shift key to enter the upper symbol on the keys.
C	Character check – The characteristics of the keyboard are determined by the parameter for the SHIFT keyword in the field description statement. The SHIFT keyword is required when this data type is specified. See the SHIFT keyword later in this section.
D	Digits only – Only the numbers 0 through 9 can be entered. The shift is positioned to the lower symbols on typewriter keyboards and is positioned to the upper symbols on both data entry keyboards and proof keyboards. The operator cannot override the shift. Negative numbers are displayed with the sign over the units position of the number. A negative value is entered in the field when the operator, having entered one or more digits, presses the Field Exit Minus key.
H	Hexadecimal – Each character position requires two keystrokes and only the numbers 0 through 9 and the letters A through F can be entered. No shift key operation is required on data-entry keyboards.
N	Numeric shift – Any character can be entered. The shift is positioned to the lower symbols on typewriter keyboards and is positioned to the upper symbols on both the data entry keyboards and the proof keyboards. The operator can change the shift by using the Shift key. A negative value is entered in the field when the operator, having entered one or more digits, presses the Field Exit Minus key.
S	Signed numeric – Only the numbers 0 through 9 can be entered. The signed numeric data type implies right adjust with zero-fill, digits only, and field exit required. To exit the field, the operator must press the Field Exit key (for positive values) or the Field Exit Minus key (for negative values). The keyboard shift is positioned to the lower symbols on typewriter keyboards and to the upper symbols on data entry and proof keyboards. When signed numeric data is entered, the system designates an additional display position, which is adjacent to and to the right of the data field. This additional display position is the sign position. It is blank for positive values and is set to - (minus sign) for negative values. The corresponding field of the record in the COBOL program is not lengthened. The zone portion of the lower-order digit contains hexadecimal D instead of a hexadecimal F, indicating a negative value.

A negative value is entered in the field when the operator, having entered one or more digits, presses the Field Exit Minus key.

V	Right half only – Only the characters that are defined at system generation can be entered. The shift is positioned to the lower right symbols on appropriate keyboards. The shift key allows entry of the upper right symbols. (See the note below after the explanation for *W*.)
W	Right half shift – Any character can be entered. The shift is positioned to the lower right symbols on appropriate keyboards. The operator can change the shift by using the shift keys.

☞**Note**: V and W should be used only for those keyboards with more than two shift positions (such as the Katanana keyboard). Don't use these data types in programs that are to be executed on other keyboards or an unrecoverable error will occur at execution.

X	Alphabetic only – Only the letters A through Z, comma, period, hyphen, and space can be entered. The shift is positioned to the lower symbols on all keyboards. The shift cannot be changed by the operator.
Y	Numeric only – Only the numbers 0 through 9, comma, period, plus, minus, and space can be entered. The shift is positioned to the lower symbols on typewriter keyboards and to the upper symbols on both the data entry keyboards and the proof keyboards. The shift cannot be changed by the operator.

A negative value is entered in the field when the operator, having entered one or more digits, presses the Field Exit Minus key.

Decimal Positions Field (Column 37)

Leave this field blank if the data is alphabetic. (In the COBOL program, the data-item created by the field description statement will be defined as alphanumeric, regardless of what you specify in the *Data Type* field.)

Enter any value from 0 through 9 to indicate a numeric value. (In the COBOL program, the data-item created by the field description statement will be defined as numeric, regardless of what you specify in the *Data Type* field.) The 0 through 9 value defines the number of positions to the right of the decimal point for the data-item. You must also specify a length in the *Length* field (columns 30-34).

Usage Field (Column 38)

You needn't make an entry in the *Usage* field if you already have an entry in the *Usage* field of the record statement. However, if you make an entry, it will override the corresponding entry in the record description statement.

☞**Note**: If the *Usage* field of the record description statement is blank, you must make one of the following valid entries:

O	the field can be used for output only.
B	the field can be used for both input and output.
blank	the field is to use the default specified in the preceding record description statement.

Output (O) Fields

For those fields defined for output (you enter an *O* in the *Usage* field):

1. The operator cannot enter data into the field.
2. You must specify a nonnumeric literal in the *Editing* field starting in column 45. The literal can have any character in the EBCDIC set and must be enclosed within single quotes (`'literal'`). Any single quote used as part of the character string must be coded twice (' ').

   The literal appears on the screen when your COBOL program issues an associated WRITE statement. The use of literals is shown in figure 2.2.
3. You must leave the *Field Name* field (columns 19-24 of the field description statement) blank.

Here's an example of a field description statement defining a field for output only:

```
----+----1----+----2----+----3----+----4----+----5----+----6----+
     A                               O      'ENTER USER ID'
```

Input and Output (B) Fields

For those fields defined for both input and output (you enter a *B* in the *Usage* field):

1. A literal is not allowed in the statement.
2. The operator can enter data on the screen in fields defined by the statements unless you specify CHECK(BY) in the editing field. (The CHECK keyword is explained later in this chapter.)
3. The operator can duplicate data from the corresponding positions in the preceding record by pressing the Duplicate key, unless you specify CHECK(DD).
4. Data can be automatically duplicated in the field when the operator has pressed the Auto Dup/Skip key, and you have coded the CHECK(AD) keyword and parameter in the editing field of the associated field description statement. The data is automatically duplicated from the corresponding positions in the previous record when the cursor enters the field.

☞**Note**: The field as it appears on the screen will not contain the same digits if the COBOL program changes its contents before writing the record to the screen.

Here's an example of a field description statement that defines a field for both input and and output; the field, EMPNO, will accept only digit data up to five characters in length; each time a record appears on the screen, EMPNO will contain the same digits as were in the corresponding field in the preceding record:

```
----+----1----+----2----+----3----+----4----+----5----+----6----+
     A            EMPNO         5D   B      CHECK(AD)
```

When using a field for both input and output, consider the following when writing your COBOL program:

1. The field description statements define 02 level elementary items in the COBOL program.

2. When a WRITE is executed, any data in the associated record will be written to the screen. You should therefore ensure that such fields are either initialized with blanks or the data you want before the WRITE is executed.

3. The operator can write over any data appearing in the field unless CHECK(BY) is in effect.

To Prevent Entry of Data in a B-Field

To prevent the entry of data in a B-Field, use the CHECK(BY) keyword described later in this chapter. With this keyword, you can cause the cursor to bypass a field on the screen either conditionally or unconditionally.

Location Field (Columns 39 through 44)

In the location field, you code the location where the field will appear on the screen as follows: you specify the line number (vertical position) in columns 39-41 and the horizontal position in columns 42-44. The actual line where the data appears on the screen depends on whether or not you specify the STARTING AT LINE *n* clause in the corresponding WRITE statement in your COBOL program. Here are the rules that determine which line a field will appear on:

1. To determine the physical line on the screen, add the STARTING AT LINE value with the value you write in columns 39-41. Then subtract 1.
2. If you don't specify a STARTING AT LINE clause, the default value will be 3.
3. The minimum value that be specified in the STARTING AT LINE clause is 2. Note that prompts specified with the PMT (for *prompt*) keyword always appear on line 2. The PMT keyword is explained later in this chapter.
4. If you don't specify a STARTING AT LINE clause, or *Line* and *Position* fields, the starting position for the first field will be the first position in line 3. Each field that follows will be in the next available display position following the previous field.

The defaults for line and position entries ensure that the Status Line and the prompt line at the top of the screen aren't overlayed, whether STARTING AT LINE is specified or not. The following table shows the relationship between STARTING AT LINE values, line entries in columns 39-41, and the physical line on the screen where the field will appear.

STARTING Value	Cols. 39-41	Screen Line
Blank (default 3)	Blank	Next available position
Blank (default 3)	1	3
Blank (default 3)	2	4
Blank (default 3)	3	5
3	Blank	Next available position
3	1	3
3	2	4
3	3	5
4	Blank	Next available position
4	1	4
4	2	5
4	3	6

Entries in the *Line* and *Position* fields must be right adjusted. Either leading zeros or leading blanks are allowed. The following table gives the dimensions for the three screen sizes available with a 5280 system:

Size in Characters	Number of horizontal rows	Number of vertical columns
1920	24	80
960	12	80
480	6	80

Editing Field (Columns 45 through 80)

Entries in the *Editing* field are optional. An entry in this field can be a single keyword with a parameter string or multiple keywords, each with its parameter string. You can extend the *Editing* field with continuation lines.

CHECK (parameters)

The CHECK keyword allows you to cause data to be edited automatically by the system. If the data isn't entered as you specify, the keyboard will lock and an error message will appear on the screen.

CHECK requires at least one of the following parameters.

Parameter	**Meaning**
AD	Automatic duplication – when the cursor moves into the field for which CHECK(AD) is specified, data from the corresponding positions in the previous record is automatically copied into the field. The operator controls whether or not duplication will occur with the Auto Dup/Skip key as described in Chapter 1. During automatic duplication, data type and CHECK keyword edits are ignored.
AS	Automatic skip – when the cursor moves into a field for which CHECK(AS) is specified, blanks are inserted into the field, and the cursor moves to the next field where the operator can manually enter data. The operator controls whether or not a skip will occur with the Auto Dup/Skip key as described in Chapter 1.
BC	Blank check – The operator cannot enter blanks into a field for which CHECK(BC) is specified. However, the operator can bypass the field, leaving it blank, by using the Field Exit key when the cursor moves to the first position of the field.
BY	Bypass – The cursor bypasses the field, either conditionally or unconditionally. When the field is bypassed, the operator cannot enter data and the contents of the field remain unchanged.

A field can be conditionally bypassed if you code the condition (and indicator) and CHECK(BY) in a secondary line of the field description statement. The only entries you can make in the secondary line are an entry in the *Sequence* field, an indicator number in the *Indicator* field, the required *A* in the *Form Type* field, and CHECK(BY) in the *Editing* field.

DD	Duplication disable – The operator cannot use the Duplication Key in fields for which CHECK(DD) is specified.
DR	Data required – The operator must enter at least one nonblank character in fields for which CHECK(DR) is specified.
FE	Field exit required – The operator must press one of the Field Exit keys to exit the field. See Chapter 1 for an explanation of the Field Exit keys. Normally, when the last character in a field is entered, the field is exited automatically.
LC	Lowercase – The operator can enter both uppercase and lowercase characters if the typewriter keyboard is used. Without LC specified, all characters are treated as uppercase characters. LC is ignored if the typewriter keyboard is not being used.
ME	Mandatory entry – The operator must enter at least one character, either blank or nonblank.
MF	Mandatory fill – If one character is entered, the operator must enter characters into all the positions of the field.
RB	Right adjust with blank fill – when the operator presses the Field Exit key, all data is shifted from the position of the cursor to the rightmost positions of the field, if no data has been entered in these positions. The unused positions to the left of the data are filled with blanks. The operator must press the Field Exit key to exit the field.
RZ	Right adjust with zero fill – when the operator presses the Field Exit key, all data is shifted from the position of the cursor to the rightmost positions of the field, if no data has been entered in these positions. The unused positions to the left of the data are filled with zeros. The operator must press the Field Exit key to exit.
RL	Right to left – As the operator enters characters, the rightmost position is filled first; each additional character is added in the next free position to the left. The operator must press the Field Exit key to exit. A *V* or *W* must be specified in the *Data Type* field.

DSPATR (attributes)

With the DSPATR keyword, you can control how characters appear on the screen with the parameters shown in figure 2.6. The following two sections give special rules when coding a DSPATR keyword.

Attribute	Meaning
BL	Blink. The displayed field(s) blinks (flashes on and off).
CA	Cancel. When specified in a field description statement,CA cancels display attributes made in the record description statement. Use CA when an attribute or characteristic specified in the record description statement isn't desirable for the field being defined. Other parameters can then be specified to control the display for the record. The parameter CA affects only the field description statement for which it is coded.
CS	Column Separators. Column separators are displayed in the field(s). Column separators are thin vertical lines between character positions; they do not reduce display capacity.
HI	Highlight. The displayed characters are highlighted (displayed with increased intensity).
ND	Nondisplay. The field(s) in the record aren't displayed.
RI	Reverse image. The field(s) in the record is displayed with images reversed (dark characters are displayed on a light background).
UL	Underline. Each character in the field(s) is underlined.

Figure 2.6. Attributes that can be specified with the DSPATR keyword

Placement of Attribute Control Characters

A DSPATR causes control characters to be placed in the first position before and the first position after a field. (the control characters do not appear on the screen). Keep the following in mind when you code the *Location* field (columns 39-44):

1. The position immediately before, and immediately after the field must be blank. You control whether a position is blank or not by what you specify in the *Position* field (columns 42-44) and the *Length* field (columns 30-34).
2. If two or more fields with display attributes follow in succession:
   - A blank position must precede the first field. If the field begins in column 1, column 80 of the previous line must be blank.
   - At least one blank position must be between each two fields.
   - One blank position must follow the last field.
3. Rules (1) and (2) are valid regardless of the number of attributes (RI, UL, BL, etc.) you specify.

☞**Note**: Make sure the positions in which display attributes are assigned are cleared before a new screen format is written to the screen. If these positions are not cleared, the display attributes will apply to the new format. (You can clear the positions desired by writing a blank record to the screen.)

Assume, using the following Data Definition Statements, that EMPFILE has been written to the screen with a STARTING AT LINE value of 5, causing the field to appear on line 5, column 1. A control character for column separators (CS) will be placed in line 4, column 80:

```
----+----1----+----2----+----3----+----4----+----5----+----6----+
     A          R EMPFILE
     A            EMPNO          5D  B 01 01DSPATR(CS)
```

Assume, using the following Data Definition Statements, that INVFILE is subsequently written to the screen with a STARTING AT LINE value of 5 or greater, and line 4, column 80 has not been cleared. Although not specified, the PRTNO field will have column separators.

```
----+----1----+----2----+----3----+----4----+----5----+----6----+
     A          R INVFILE
     A            PRTNO           5D  B 01 01
```

Valid Combinations Of HI, RI, and UL

You cannot specify the HI, RI, and UL attributes for the same field, either explicitly in the same field description statement or in a field description statement in combination with a record description statement. You can, however, specify any combination of two of these attributes.

If you wish to cancel the defaults specified in a record description statement, code the CA (for cancel) attribute in the desired field description statement, and then code the desired HI, RI, and/or UL attributes.

PMT (message)

The PMT (for *prompt*) keyword allows you to display a prompting message on line 2 of the screen. The prompt appears when the cursor enters the field. The prompt is cleared when the cursor leaves the field.

The prompting message is displayed on the second physical display line starting in position 1 and continues on the following lines depending on length of the message.

The required parameter, *message*, can be any message, with a maximum length of 200 character positions. Any displayable character and spaces are valid in the message.

The message must be within single quotes (' message '). Any single quotes within the message must be coded twice (' ').

☞**Note**: The prompt message is cleared when the cursor exits the field.

SHIFT (codes)

The SHIFT keyword is required in field description statements in which the letter C is specified for *Data Type*. This keyword allows you to program the keyboard conditioning for each position in the field.

The required parameter, *codes*, is a string of characters, one for each character position in the field. The following characters are valid:

Character	**Meaning**
A	Alphabetic shift – Any character can be entered. The shift is positioned to the lower symbol on each key on all keyboards. The operator can use the Shift key to enter the upper symbol on the keys.
D	Digits only – Only the digits 0 through 9 can be entered. The shift is positioned to the lower symbols on typewriter keyboards and is positioned to the upper symbols on both data entry keyboards and proof keyboards. Negative numbers are displayed with the sign over the last digit of the number. The operator cannot override the shift.
H	Hexadecimal – Each character position requires two keystrokes and only the numbers 0 through 9 and the letters A through F can be entered. No shift key operation is required on data-entry keyboards.
N	Numeric shift – Any character can be entered. The shift is positioned to the lower symbols on typewriter keyboards and is positioned to the upper symbols on both the data entry keyboards and the proof keyboards. The operator can change the shift by using the Alphabetic Shift key.
V	Right half only – Only the characters that are defined at system generation can be entered. The shift is positioned to the lower right symbols on World Trade keyboards. The shift key allows entry of the upper right symbols.
W	Right half shift – Any character can be entered. The shift positioned to the lower right symbols on the World Trade keyboards. The operator can change the shift by using the shift keys.
	☞**Note**: V and W should be used only for those keyboards with more than two shift positions (such as the Katakana keyboard). Don't use these data types in programs that are to be executed on other keyboards or an unrecoverable error will occur at execution.
X	Alphabetic only – Only the letters A through Z, comma, period, hyphen, and space can be entered. The shift is positioned to the lower symbols on all keyboards. The shift cannot be changed by the operator.
Y	Numeric only – Only the numbers 0 through 9, comma, period, plus, minus, and space can be entered. The shift is positioned to the lower symbols on typewriter keyboards and to the upper symbols on both the data entry keyboards and the proof keyboards. The shift cannot be changed by the operator.

Keyword Conflicts and Compatibilities

Some keywords for field description statements cannot be used in combination with certain entries in the *Usage* field, certain entries in the *Data Type* field, or certain other keywords and parameters. The valid and invalid combinations are indicated in the following charts.

The Xs represent invalid combinations. For example, in figure 2.7 (Part 2), CHECK(DR) cannot be used for output fields (coded with O in the *Usage* field). Also notice that combining DR (data required) and BY (bypass) as parameters for CHECK is invalid. Also, DR cannot be specified twice for the same field.

Keyword	Usg. Fld.	Data Type Field Entry										Check Parameters													Keywords	
	O	A	C	D	H	N	S	W	X	Y	V	LC	AD	AS	BC	BY	BY1*	FR	ME	FE	MF	RB	RZ	RL	PMT	SHIFT
PMT	X	.	.	.	.	.	.	.	.	.	.	.	.	.	.	X	.	.	.	.	.	.	.	.	X	.
SHIFT	X	X	.	X	X	X	X	X	X	X	X	.	.	.	.	X	.	.	.	.	.	.	.	.	.	X

*BY1 refers to CHECK(BY) used as the exclusive entry in Editing field on a secondary line with an indicator specified.

Figure 2.7 (Part 1 of 2). Keyboard conflicts and compatibilities

Check Parameter	Usg. Fld.	Data Type Field Entry										Check Parameters													Keywords	
	O	A	C	D	H	N	S	W	X	Y	V	LC	AD	AS	BC	BY	BY1*	DR	FE	ME	MF	RB	RZ	RL	PMT	SHIFT
LC	X	.	.	.	.	.	.	.	.	.	.	X	.	.	.	X	.	.	.	.	.	.	.	.	.	.
AD	X	.	.	.	.	.	.	.	.	.	.	.	X	X	.	X	.	.	.	.	.	.	.	.	.	.
AS	X	.	.	.	.	.	.	.	.	.	.	.	X	X	.	X	.	.	.	.	.	.	.	.	.	.
BC	X	.	.	.	.	.	.	.	.	.	.	.	.	.	X	X	.	.	.	.	.	.	.	.	.	.
BY	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X
BY1*	X	.	.	.	.	.	.	.	.	.	.	.	.	.	.	X	X	.	.	.	.	.	.	.	.	.
DR	X	.	.	.	.	.	.	.	.	.	.	.	.	.	.	X	.	X	.	.	.	.	.	.	.	.
FE	X	.	.	.	.	.	.	.	.	.	.	.	.	.	.	X	.	.	X	.	.	.	.	.	.	.
ME	X	.	.	.	.	.	.	.	.	.	.	.	.	.	.	X	.	.	.	X	.	.	.	.	.	.
MF	X	.	.	.	.	.	X	.	.	.	.	.	.	.	.	X	.	.	.	.	X	X	X	.	.	.
RB	X	.	X	.	.	.	.	.	.	.	.	.	.	.	.	X	.	.	.	.	X	X	X	X	.	.
RZ	X	.	X	.	.	.	.	.	.	.	.	.	.	.	.	X	.	.	.	.	X	X	X	X	.	.
RL	X	X	X	X	X	X	X	.	X	X	.	.	.	.	.	X	.	.	.	.	.	X	X	X	.	.

*BY1 refers to CHECK(BY) used as the exclusive entry in Editing field on a secondary line with an indicator specified.

Figure 2.7 (Part 2 of 2). Keyboard conflicts and compatibilities

Chapter 3. Transaction I/O — Writing the COBOL Program

This chapter provides information on *transaction I/O* — the data set organization in the IBM 5280 COBOL language that permits the exchange of data between an operator at a work station and a COBOL program.

Subjects covered include:

- A description and coding examples of the I/O statements needed for transaction I/O: the FILE-CONTROL paragraph, and the OPEN, CLOSE, READ, and WRITE statements.
- I/O error processing, with an example of an exception/error procedure and use of the Status Key.
- How to code the statements needed to obtain return information (for example, codes returned when the operator presses a command key).
- A coding example of a program using transaction I/O.
- A coding example showing the use of indicators and CHECK(BY) (for Bypass).

To use the information given in this chapter, you'll also have to refer to the following chapters:

- Chapter 1, introduces transaction I/O and summarizes the keyboard and how it affects the operations of your program.
- Chapter 2, which describes how to code the Data Definition Statements (DDS) that design the screen and edits the operator entries.

Summary of COBOL Transaction I/O Statements

This chapter will explain transaction I/O using the example of the COBOL program in figure 3.2. (The Data Definition Statements used by this program are from the example in figure 2.3 in Chapter 2.) Here is a brief summary of the program:

1. The program writes a screen in which an operator will enter data on new employees.
2. The program checks the data entered for errors; if no errors are found, it updates the employee master file with the data. If an error is found, it writes an error message and allows the operator to correct the data.

The following sections provide a guide to writing a COBOL program with transaction I/O and some of the COBOL statements used. Statements discussed in the text are referenced in figure 3.2 with keyed numbers (**1**, **2**, etc.). For detailed information on the syntax and rules of the COBOL statements used, see the *IBM 5280 COBOL Language Reference*.

FILE-CONTROL Paragraph

The entries in the FILE-CONTROL paragraph for transaction I/O are:

```
SELECT file-name ASSIGN TO WORKSTATION [n]
ORGANIZATION IS TRANSACTION
[FILE STATUS IS data-name-3]
[ACCESS MODE IS SEQUENTIAL]
[CONTROL-AREA IS data-name-4]
```

WORKSTATION indicates the device is a 5280 data station.

n is an integer that specifies one of the three sizes of work station screens as follows:

480 for the 480-character screen.
960 for the 960-character screen.
1920 for the 1920-character screen.

Note 1: If you don't specify *n*, a screen size of 1920 characters is assumed.

Note 2: The screen size of the work station on which a program is to execute must be at least the same size specified by *n*. Otherwise, the program will not execute. For example, if you specify 1920, the program will execute on systems that support 480-, 960- and 1920-character data stations; if you specify 480, the program will execute only on stations that support 480-character data stations.

An example of the FILE-CONTROL paragraph is shown in **2** in figure 3.2. Note that for transaction I/O, ORGANIZATION is always TRANSACTION, and ACCESS is always SEQUENTIAL.

The FILE STATUS clause is explained under "I/O Error Processing" later in this chapter. The CONTROL-AREA clause is explained under the section "Return Information".

OPEN

The OPEN statement determines the availability of the file, and, if successful, results in the file being in an OPEN mode. You must always open the file with the I-O phrase, as shown in figure 3.2 at **21**.

WRITE

The WRITE statement places onto the work station screen a set of data-items in your program generated by Data Definition Statements. The statements are copied into the program at compilation, as shown in the example in figure 3.2 at **4**.

The format of the WRITE statement is as follows:

```
WRITE record-name
   [FROM identifier-1]
   [FORMAT IS literal-2]
   [STARTING AT LINE literal-3]
   [INDIC [IS | ARE]
   |  [INDICATOR [IS]]
   |  [INDICATORS [ARE]]
    identifier-4]
```

Note: Always specify the above clauses in the order shown. The rules and syntax for the WRITE statement are given in detail in the *5280 COBOL Language Reference*.

The FORMAT, STARTING AT LINE, and INDICATOR clauses are explained in the next three sections.

FORMAT Clause

The FORMAT clause designates the screen format, as defined by a set of Data Definition Statements, that is to be written to the screen. If FORMAT is not specified, the last specified format is written to the screen. If a series of WRITE statements is writing the same format to the screen, you need to specify the FORMAT clause only on the first WRITE in the series.

In the example, FORMAT was specified in each WRITE to allow the writing of different screen images, as explained below.

At **3** and **4** in figure 3.2, six sets of Data Definition Statements were copied into the program: EMPRECD, ERRMSG1, ERRMSG2, ERRMSG3, and PROMPT.

The Data Definition Statements that define each of the formats are shown at **23** through **27**. They cause the records at **3** and **4** to be generated into the COBOL program at compilation.

The WRITE statements at **11**, **13**, **15**, **17**, and **19** each reference one of these formats as follows:

- The WRITE with `FORMAT IS "PROMPT"` at **17** writes a message prompting the operator to enter employee data. This is followed by the the WRITE with `FORMAT IS "EMPRECD"` at **19**. This statement writes the fields in which the operator is to enter employee data.
- The WRITEs with FORMAT IS `"ERRMSG1"`, `"ERRMSG2"` and `"ERRMSG3"` at **11**, **13**, and **15**, write error messages to the screen when the program finds an incorrect entry by the operator.

☞**Note**: A READ statement must be executed after a WRITE before a subsequent WRITE is issued. In the example shown in figure 3.2, READs are therefore coded at **12**, **14**, **16**, **18**, and **20**.

STARTING AT LINE Clause

The STARTING AT LINE clause is optional. Together with the location field you specify in the Data Definition Statements, it determines the starting line on the screen of the data being written. See Chapter 2 under "Location Field" for information on how the starting line is determined.

INDICATOR Clause

In the INDICATOR clause, *identifier-4* specifies a data-item which is a boolean table of one or more elements. For a coding example using the INDICATOR clause, see the section "INDICATOR Example" later in this chapter.

For the rules to follow in coding the INDICATOR clause with a data-item, see the *5280 Language Reference* manual under "OCCURS Clause" and "INDICATOR Clause".

READ

The READ statement is normally issued when data is expected from the operator. When the operator has completed entry and pressed the appropriate exit key, control resumes at the next COBOL statement following the READ.

The format of the READ statement is:

READ *file-name* RECORD [INTO identifier-1]

The syntax and rules for writing a READ statement for transaction I/O are the same as for SEQUENTIAL I/O. See the *5280 COBOL Language Reference* for details.

Sequence of Transaction I/O Statements

Don't issue two WRITEs in your program without executing a READ in between. A READ must always be issued *after* a WRITE has been issued and *before* a subsequent WRITE can be issued.

The READ must be issued whether or not a response is expected from the operator.

You normally expect a response when one of the fields written to the screen has been defined for both input and output. That is, a B has been specified in the *Usage* field of the associated Data Definition Statements. After the READ is executed, your program can process any command key codes or data entered by the operator.

You normally don't expect a response when all of the fields have been defined for output only. That is, an O has been specified in the *Usage* field(s) of the associated Data Definition Statements. Nevertheless, the WRITE must be followed by a READ before another WRITE can be issued.

Clearing the Screen with WRITE

A WRITE clears all data on the screen in lines following the starting line of the data being written, and then writes the data defined (either explicitly or by default) by the FORMAT clause.

In the example shown in figure 3.2, when an error message is written at **11**, **13**, or **15**, all information on the lines that follow the message line are erased. Therefore, the screen with the original format is rewritten at **19**.

CLOSE

The CLOSE statement detaches the work station associated with the program. Once a transaction file is closed, it cannot be opened again by your program before again loading the program.

I/O Error Processing

This section discusses the use of the Status Key and Exception/Error procedures in handling possible transaction I/O errors.

Status Key

The Status Key is a 2-character data-item you define in the Data Division of your program and name in the FILE STATUS clause, as shown for TUBE-STAT in **2** in figure 3.3 later in this chapter.

It is recommended that you define a Status Key for all files *and* that your COBOL program check the contents after each I/O request. Otherwise, errors may go undiscovered by the program, producing results that are both destructive and difficult to diagnose.

If neither a Status Key nor an EXCEPTION/ERROR procedure (described later in this section) is present and an error occurs, the program will display a message in the status line at the top of the screen. A message identifier in the format *92nn* will appear in the status line; *nn* is the code that would have been placed in the Status Key had it been present.

See Appendix B for a complete list of the values that can be placed in the Status Key and their meanings.

Coding the Status Key

To code the status key:

1. Relate FILE STATUS and a data-item in the SELECT clause as shown below and in $\boxed{2}$ in figure 3.2:

   ```
   FILE STATUS IS TUBE-STAT
   ```

2. Code a data-item in Working Storage Section or the Linkage Section as shown below and at $\boxed{7}$ in figure 3.2:

   ```
   77 TUBE-STAT     PIC XX.
   ```

After each READ, WRITE, OPEN, and CLOSE operation, a return code posted in the status key shows the outcome of the operation. Your program can look at the status key, and then take the appropriate action.

The return codes for transaction I/O are the same as those for SEQUENTIAL I/O. The return codes and their meaning are given in Appendix B.

Exception/Error Procedures

You can also code a procedure to handle errors using the EXCEPTION/ERROR declarative, as shown at $\boxed{8}$ and and $\boxed{9}$ in figure 2.3. The procedure, WORK-STATION, takes control each time an error or exception occurs on SCREEN-FILE-PR3. You can include code in the procedures to diagnose the error, and take subsequent action in consideration of the error. The EXCEPTION/ERROR procedure is used only when a file is in open status. Therefore, if any operation is attempted against a file which has already been closed, or was never opened, then the EXCEPTION/ERROR procedure is not executed. COBOL will return a Status Key value of 92.

The USE FOR DEBUGGING declarative (coded before the EXCEPTION/ERROR procedures) and other debugging aids are described in Chapter 9.

Example Using Status Key and ERROR Procedures

In the example in figure 3.3, the procedure WORK-STATION at $\boxed{8}$ receives control each time a nonzero return code is placed in the status key. The procedure displays a message and the value in the status key, and then stops the run.

Return Information

Two different types of information can be returned to a COBOL program after a transaction I/O operation in the following areas:

- *Attribute Data Area*, which provides information on the status of the terminal which loaded your program. You will probably need this information only when (1) your installation has different size (1920-, 960-, and/or 480-character) screens and (2) your program needs to know the size of the screen at the data station to which it is writing screen images.
- *Work Station Control Area*, which contains a special code when control is returned to the COBOL program from the work station. The code indicates which function or command key the operator pressed. The command keys and their corresponding codes are given in figure 1.3 and 1.4 in Chapter 1.

The work station control area and the attribute data area are optional. If used, you code them as data-items in your program according to the guidelines given in the following sections.

Attribute Data

You can get information on a terminal status after each transaction I/O operations as follows:

1. Code an entry under SPECIAL-NAMES equating ATTRIBUTE-DATA to a mnemonic name as shown below and at **1** in figure 3.2:

   ```
   ATTRIBUTE-DATA IS TERMINAL-INFO.
   ```

2. Code data description entries in the WORKING-STORAGE SECTION with the characteristics shown in the example below and in figure 3.2 at **5**.

   ```
       01   WSTATION-INFO.
            02 TTYPE              PIC X.
            02 SSIZE              PIC X.
            02 LOCATION           PIC X.
            02 ON-OFF-LINE        PIC X.
            02 ALLOC-STATUS       PIC X.
            02 INPUT-STATUS       PIC X.
            02 DATA-STATUS        PIC X.
            02 INQ-STATUS         PIC X.
   ```

Data-item	Value returned	Meaning
TTYPE	D	Display
SSIZE	1	1920 Characters (24 x 80)
	2	960 Characters (12 x 80)
	3	480 Characters (6 x 80)
LOCATION	L	Local
ON-OFF-LINE	O	Online
ALLOC-STATUS	A	Allocated to this program.
INPUT-STATUS	N	Input is not allowed.
DATA-STATUS	N	No data is pending.
INQ-STATUS	N	Terminal is not in inquiry mode.

Figure 3.1. Explanation of values returned in the ATTRIBUTE-DATA area

3. At the appropriate point in the COBOL program, code an ACCEPT statement to access the data. In the example at **10**, the following statement was coded:

   ```
   ACCEPT WSTATION-INFO FROM TERMINAL-INFO.
   ```

   The logic in the statements that follow the ACCEPT statement in the example causes a message to be issued and the run to be stopped if the work station that loaded the COBOL program does not have a 1920-character display screen. (See the section "FILE-CONTROL Paragraph" earlier in this chapter for the rules on specifying screen size.)

Work Station Control Area

The work station control area tells the COBOL program which command key the operator pressed; this information is provided after a READ operation.

To build a work station control area:

1. Relate a data-item and CONTROL-AREA in the SELECT clause as shown below and at 3 in figure 3.2:

   ```
   CONTROL-AREA IS WSTATION-CONTROL-AREA
   ```

2. Code an area in the Data Division using the data-item as a 01 level group item as shown below and at 6 in figure 2.3:

   ```
         01  WSTATION-CONTROL-AREA.
             03 COMMAND-KEY        PIC X(2).
             03 FILLER             PIC X(10).
   ```

The value returned in COMMAND-KEY is either 00 (a normal return) when the record is exited after the operator presses the appropriate system function key, or a number indicating one of the COBOL command keys. See figures 1.3 and 1.4 in Chapter 1 for a description of these keys.

The program in the example shown in figure 3.2 checks the command key code at 22 after each READ.

Transaction I/O Examples

The examples in this section show:

- Interactive entry and processing between an operator and a COBOL program. The example shows how transaction I/O statements are coded; explanatory text is in the earlier section of this chapter.
- The use of the INDICATOR clauses, complete with explanatory text.

Example: Interactive Entry and Processing

The example in figure 3.2 writes a record to the screen, checks the response when the subsequent read is executed, and if no error is found, updates the employee master file.

The Data Definition Statements for EMPRECD at 23 are explained in Chapter 2 under "Example of Coding Data Definition Statements."

```
    IDENTIFICATION DIVISION.
    PROGRAM-ID.  ADD-NEW-EMPLOYEES.
    AUTHOR.  A NAME.
    ENVIRONMENT DIVISION.
    CONFIGURATION SECTION.
    SOURCE-COMPUTER. IBM-370.
    OBJECT-COMPUTER. IBM-5280.
    SPECIAL-NAMES.
2       ATTRIBUTE-DATA IS TERMINAL-INFO.
    INPUT-OUTPUT SECTION.
    FILE-CONTROL.
3       SELECT SCREEN-FILE-PR3 ASSIGN TO WORKSTATION 1920
          ORGANIZATION IS TRANSACTION
          ACCESS MODE IS SEQUENTIAL
          FILE STATUS IS TUBE-STAT
          CONTROL-AREA IS WSTATION-CONTROL-AREA.
        SELECT EMPMAS-FILE ASSIGN TO DISK
           ORGANIZATION IS RELATIVE
           ACCESS MODE IS RANDOM
           RELATIVE KEY IS RKEY
           FILE STATUS IS DISK-STAT.
    DATA DIVISION.
    FILE SECTION.
    FD  SCREEN-FILE-PR3 LABEL RECORDS ARE OMITTED.
    01  SCREEN-RECORD PIC X(88).
    FD  EMPMAS-FILE
        VALUE OF OWNER-ID IS "BELL"
        LABEL RECORDS ARE STANDARD.
3       COPY DDS-EMPRECD.
001 01      EMPRECD.
002     02  ACREC     PIC X(00002).
004     02  EMPNO     PIC X(00005).
006     02  ENAME     PIC X(00020).
008     02  STRAD     PIC X(00020).
010     02  CTYST     PIC X(00020).
012     02  ZIPCD     PIC X(00005).
014     02  BEGDT     PIC X(00006).
016     02  SOSNO     PIC X(00009).
018     02  MARST     PIC X(00001).
    WORKING-STORAGE SECTION.
4       COPY DDS-ERRMSG1.
001 01      ERRMSG1  PIC X.
4       COPY DDS-ERRMSG2.
001 01      ERRMSG2  PIC X.
4       COPY DDS-ERRMSG3.
001 01      ERRMSG3  PIC X.
4       COPY DDS-PROMPT.
001 01      PROMPT   PIC X.
    01  LAST-RECORD-SAVE      PIC X(88) VALUE SPACES.
5   01 WSTATION-INFO.
        02 TTYPE              PIC X.
        02 SSIZE              PIC X.
        02 LOCATION           PIC X.
        02 ON-OFF-LINE        PIC X.
        02 ALLOC-STATUS       PIC X.
        02 INPUT-STATUS       PIC X.
        02 DATA-STATUS        PIC X.
        02 INQ-STATUS         PIC X.
```

Figure 3.2. Example of transaction I/O statements (Part 1 of 5)

```
6  01 WSTATION-CONTROL-AREA.
        03 COMMAND-KEY        PIC X(2).
        03 FILLER             PIC X(10).
    01  SWITCHES.
        02 STOP-RUN           PIC 9 VALUE 0.
        02 CONTINUE           PIC 9 VALUE 1.
        02 INVALID-EMPNO      PIC 9 VALUE 0.
        02 ACTIVE-RECORD      PIC 9 VALUE 0.
        02 ERROR-FOUND        PIC 9 VALUE 0.
        02 WRONG-FUNC-KEY     PIC 9 VALUE 0.
7   77  TUBE-STAT PIC XX.
    77  DISK-STAT             PIC XX.
    77  RKEY                  PIC 99999.
    PROCEDURE DIVISION.
    DECLARATIVES.
    DEBUG-SECTION SECTION.
       USE FOR DEBUGGING ON READ-MASTER.
           DISPLAY "READ-MASTER ENTERED".
           DISPLAY "REKEY= " RKEY.
           DISPLAY "EMPNO= " EMPNO.
8  IO-ERROR SECTION.
       USE AFTER ERROR PROCEDURE ON SCREEN-FILE-PR3.
    WORK-STATION.
            DISPLAY "ERROR ON WORK STATION I/O".
            DISPLAY "FILE STATUS IS " TUBE-STAT.
            DISPLAY "RUN STOPPED.".
            STOP RUN.
9  DISKETTE-IO-ERROR SECTION.
       USE AFTER ERROR PROCEDURE ON EMPMAS-FILE.
    DISKETTE.
            DISPLAY "ERROR ON DISKETTE I/O".
            DISPLAY "FILE STATUS IS " DISK-STAT.
            DISPLAY "RUN STOPPED." .
            STOP RUN.
    END DECLARATIVES.
    EXECUTE SECTION.
    CHECK-TERMINAL-ROUTINE.
        PERFORM OPENS.
10      ACCEPT WSTATION-INFO FROM TERMINAL-INFO.
        IF SSIZE NOT EQUAL TO "1"
           DISPLAY "YOUR SCREEN IS TOO SMALL."
           DISPLAY "THIS PROGRAM REQUIRES 1920-CHARACTERS SCREEN"
           DISPLAY "CLOSING DOWN"
           MOVE 1 TO STOP-RUN.
    MAIN-ROUTINE.
        PERFORM WORK-STATION-READS UNTIL STOP-RUN = 1.
        PERFORM CLOSES.
        STOP RUN.
```

Figure 3.2. Example of transaction I/O statements (Part 2 of 5)

```
 WORK-STATION-READS.
****************************************************************
*THIS PARAGRAPH WRITES A RECORD TO THE SCREEN, CHECKS THE      *
*RESPONSE WHEN THE SUBSEQUENT READ IS EXECUTED, AND, IF NO     *
*ERROR IS FOUND, UPDATES THE EMPLOYEE MASTER FILE.             *
*                                                              *
*THE LOGIC FOLLOWED UPON ENTRY IS AS FOLLOWS:                  *
*                                                              *
*1.  ON THE FIRST ENTRY, WRITES THE FIRST SCREEN, AND ISSUES   *
*    A READ TO CHECK THE ENTRY BY THE OPERATOR.                *
*                                                              *
*   IF AN ERROR ISN'T FOUND, THE EMPLOYEE MASTER RECORD IS     *
*   UPDATED. IF AN ERROR IS FOUND, THE APPROPRIATE SWITCH IS   *
*   SET, AND WORK-STATION-READS IS AGAIN EXECUTED AS DESCRIBED*
*   IN (2) BELOW.                                              *
*                                                              *
*2.  ON THE SECOND OR SUBSEQUENT ENTRY, DETERMINES IF AN       *
*    ERROR WAS FOUND ON THE PREVIOUS ENTRY.                    *
*                                                              *
*    IF AN ERROR WAS FOUND, A MESSAGE IS WRITTEN AND THE       *
*    PREVIOUS SCREEN REWRITTEN.                                *
*                                                              *
*    OTHERWISE, WRITES A SCREEN IN WHICH THE OPERATOR CAN      *
*    ENTER FRESH DATA.                                         *
*                                                              *
****************************************************************
     IF INVALID-EMPNO = 1
11      WRITE SCREEN-RECORD FROM ERRMSG1
           FORMAT IS "ERRMSG1"
           STARTING AT LINE 3
12      READ SCREEN-FILE-PR3
     ELSE IF ACTIVE-RECORD = 1
13      WRITE SCREEN-RECORD FROM ERRMSG2
           FORMAT IS "ERRMSG2"
           STARTING AT LINE 3
14      READ SCREEN-FILE-PR3
      ELSE IF WRONG-FUNC-KEY = 1
15      WRITE SCREEN-RECORD FROM ERRMSG3
           FORMAT IS "ERRMSG3"
           STARTING AT LINE 3
16      READ SCREEN-FILE-PR3.
     IF ERROR-FOUND = 0
       MOVE SPACES TO SCREEN-RECORD
17      WRITE SCREEN-RECORD
          FORMAT IS "PROMPT"
          STARTING AT LINE 3
18      READ SCREEN-FILE-PR3
     ELSE IF ERROR-FOUND = 1
       MOVE LAST-RECORD-SAVE TO SCREEN-RECORD.
************************************************
*                                              *
*         IF/ELSE ENDS HERE                    *
*                                              *
*                                              *
************************************************
```

Figure 3.2. Example of transaction I/O statements (Part 3 of 5)

```
19          WRITE SCREEN-RECORD
            FORMAT IS "EMPRECD"
            STARTING AT LINE 6.
20          READ SCREEN-FILE-PR3 RECORD INTO EMPRECD.
         MOVE EMPRECD TO LAST-RECORD-SAVE.
         PERFORM SET-SWITCHES.
         PERFORM COMMAND-KEY-CHECK.
         IF CONTINUE = 1
            PERFORM READ-MASTER.
         IF CONTINUE = 1
         PERFORM UPDATE-MASTER.
     READ-MASTER.
         MOVE EMPNO TO RKEY.
         SUBTRACT 1000 FROM RKEY GIVING RKEY.
         READ EMPMAS-FILE
            INVALID KEY
               MOVE 0 TO CONTINUE
               MOVE 1 TO ERROR-FOUND, INVALID-EMPNO.
         IF CONTINUE = 1
            IF ACREC NOT EQUAL TO "F"
               MOVE 1 TO ACTIVE-RECORD, ERROR-FOUND
               MOVE 0 TO CONTINUE
         ELSE MOVE "A" TO ACREC.
     UPDATE-MASTER.
         MOVE SCREEN-RECORD TO EMPRECD.
         REWRITE EMPRECD
            INVALID KEY
               MOVE 1 TO INVALID-EMPNO
               MOVE 0 TO CONTINUE.
21   OPENS.
         OPEN I-O SCREEN-FILE-PR3
                  EMPMAS-FILE.
     CLOSES.
         CLOSE EMPMAS-FILE
               SCREEN-FILE-PR3.
22   COMMAND-KEY-CHECK.
         IF COMMAND-KEY = "00"
           MOVE 1 TO CONTINUE
        ELSE IF COMMAND-KEY = "02"
           MOVE 1 TO STOP-RUN
           MOVE 0 TO CONTINUE
        ELSE MOVE 1 TO WRONG-FUNC-KEY
              MOVE 1 TO ERROR-FOUND
              MOVE 0 TO CONTINUE.
     SET-SWITCHES.
         MOVE 0 TO ACTIVE-RECORD, INVALID-EMPNO,
                   WRONG-FUNC-KEY, STOP-RUN, ERROR-FOUND.
         MOVE 1 TO CONTINUE.
```

Figure 3.2. Example of transaction I/O statements (Part 4 of 5)

DDS SOURCE LISTING

```
0
01A        R EMPRECD          B      DSPATR(CS UL) 23
02A          ACREC         2         DSPATR(CA ND)
03A                                  CHECK(BY)
04A          EMPNO        5D    01 11CHECK(DR RZ)
05A                           O      'EMPLOYEE NUMBER' DSPATR(CA)
06A          ENAME       20X    02 11CHECK(DR RB)
07A                           O      'EMPLOYEE NAME' DSPATR(CA)
08A          STRAD       20A    03 11CHECK(DR RB)
09A                           O      'STREET ADDRESS' DSPATR(CA)
10A          CTYST       20X    04 11CHECK(DR RB)
11A                           O      'CITY, STATE'  DSPATR(CA)
12A          ZIPCD        5D    05 11CHECK(DR MF FE)
13A                           O      'ZIP CODE'  DSPATR(CA)
16A          BEGDT        6D    06 11CHECK(DR MF FE)
17A                           O      'BEGINNING DATE' DSPATR(CA)
18A          SOSNO        9D    07 11CHECK(DR MF)
19A                           O      'SOCIAL SECURITY NUMBER' DSPATR(CA)
20A          MARST        1X    08 11CHECK(DR MF)
21A                           O      'MARITAL STATUS - M OR S' DSPATR(CA)
01A        R ERRMSG1          B      DSPATR(RI HI BL) 24
02A                           O 01 02'EMPLOYEE NUMBER INCORRECT. MUST BE-
02A                                   MORE THAN 1000 OR LESS THAN 2000.'
01A        R ERRMSG2                 DSPATR(RI HI BL) 25
02A                           O 01 02'EMPLOYEE NUMBER WRONG. ALREADY ASS-
02A                                  IGNED TO ANOTHER EMPLOYEE.'
01A        R ERRMSG3                 DSPATR(RI HI BL) 26
02A                           O 01 02'YOU HIT WRONG COMMAND KEY. ONLY EN-
02A                                  TER, 02, AND 03 ALLOWED.'
01A        R PROMPT           B      DSPATR(RI HI) 27
02A                           O 01 02'ENTER INFORMATION ON NEW EMLOYEE.-
02A                                                                   '
```

Figure 3.2. Example of transaction I/O statements (Part 5 of 5)

Indicator Example

An example of the use of transaction I/O and indicators is shown in figure 3.3. (See the *5280 COBOL Language Reference* manual under "OCCURS Clause" and "INDICATOR Clause" for detailed rules on coding the INDICATOR clause with a data-item.)

The logic of the program shown in figure 3.3 is as follows:

- The program writes four fields to the screen on succeeding lines in the following order: ENAME, STRAD, CTYST, and EMPNO. The cursor appears in the first position of the first field, ENAME.
- The operator enters the employee name, street address, city and state, and employment number in the designated fields and presses the Enter key.
- The program checks whether or not the employee number entered by the operator is valid.
- If the employee number is valid, the program updates EMPMAS-FILE and writes a fresh screen format.
- If the employee number is not valid, (1) the program writes an error message to the screen indicating that the number entered isn't valid and (2) writes a screen format with the information previously entered. The cursor bypasses the ENAME, STRAD, and CTYST fields and appears in the first position of the EMPNO field.

The use of indicators allows the program to control whether or not ENAME, STRAD, and CTYST are bypassed by the cursor. The following text explains how this is achieved:

1. The indicators 51, 52, and 53 are coded on secondary lines of the statements at $\boxed{4}$, $\boxed{5}$, and $\boxed{6}$ with the CHECK(BY) keyword. The secondary lines follow immediately after the lines that define the fields to be bypassed.
2. The indicator numbers 51, 52, and 53 are associated with a boolean data-item defined at $\boxed{1}$.
3. Before an error message is written, the indicators are set to *on* at $\boxed{2}$.
4. The WRITE statement at $\boxed{3}$ contains the related INDICATOR clause which writes the new screen format and causes the cursor the bypass the ENAME, STRAD, and CTYST fields.

```
     IDENTIFICATION DIVISION.
     PROGRAM-ID.  INDICATORS.
     AUTHOR.  A NAME.
     ENVIRONMENT DIVISION.
     CONFIGURATION SECTION.
     SOURCE-COMPUTER. IBM-370 WITH DEBUGGING MODE.
     OBJECT-COMPUTER. IBM-5280.
     INPUT-OUTPUT SECTION.
     FILE-CONTROL.
         SELECT SCREEN-FILE-PR3 ASSIGN TO WORKSTATION 1920
           ORGANIZATION IS TRANSACTION
           ACCESS MODE IS SEQUENTIAL
           FILE STATUS IS TUBE-STAT
           CONTROL-AREA IS WSTATION-CONTROL-AREA.
         SELECT EMPMAS-FILE ASSIGN TO DISK
            ORGANIZATION IS RELATIVE
            ACCESS MODE IS RANDOM
            RELATIVE KEY IS RKEY
            FILE STATUS IS DISK-STAT.
     DATA DIVISION.
     FILE SECTION.
     FD  SCREEN-FILE-PR3 LABEL RECORDS ARE OMITTED.
     01  SCREEN-RECORD PIC X(65).
     01  SCREEN-RECORD-CLEAR PIC X(65).
     FD  EMPMAS-FILE
         LABEL RECORDS ARE STANDARD.
         COPY DDS-INDIC.
0001 01      EMPRECD.
0002     02  ENAME    PIC X(00020).
0005     02  STRAD    PIC X(00020).
0008     02  CTYST    PIC X(00020).
0011     02  EMPNO    PIC X(00005).
     WORKING-STORAGE SECTION.
1    01 IFIELDS.
         02  INDCATOR-FIELD PIC 1 OCCURS 3 TIMES INDIC 51.
     01  INDCATOR-ON  PIC XXX VALUE '111'.
         COPY DDS-ERRMSG1.
0001 01      ERRMSG1  PIC X.
     01  WSTATION-CONTROL-AREA.
         03 COMMAND-KEY       PIC X(2) VALUE '00'.
         03 FILLER            PIC X(10).
     01  SWITCHES.
         02 STOP-RUN          PIC 9 VALUE 0.
         02 CONTINUE          PIC 9 VALUE 1.
         02 INVALID-EMPNO     PIC 9 VALUE 0.
         02 ERROR-FOUND       PIC 9 VALUE 0.
     77  TUBE-STAT            PIC XX.
     77  DISK-STAT            PIC XX.
     77  RKEY                 PIC 9999.
     PROCEDURE DIVISION.
     DECLARATIVES.
     DEBUG-SECTION SECTION.
        USE FOR DEBUGGING ON READ-MASTER.
            DISPLAY 'READ-MASTER ENTERED'.
            DISPLAY 'REKEY= ' RKEY.
            DISPLAY 'EMPNO= ' EMPNO.
```

Figure 3.3. Example of indicators and conditional bypass (Part 1 of 3)

```
    IO-ERROR SECTION.
       USE AFTER ERROR PROCEDURE ON SCREEN-FILE-PR3.
    WORK-STATION.
            DISPLAY 'ERROR ON WORK STATION I/O'.
            DISPLAY 'FILE STATUS IS ' TUBE-STAT.
            DISPLAY 'RUN STOPPED.'.
            STOP RUN.
    DISKETTE-IO-ERROR SECTION.
       USE AFTER ERROR PROCEDURE ON EMPMAS-FILE.
    DISKETTE.
            DISPLAY 'ERROR ON DISKETTE I/O'.
            DISPLAY 'FILE STATUS IS ' DISK-STAT.
            DISPLAY 'RUN STOPPED.' .
            STOP RUN.
    END DECLARATIVES.
    EXECUTE SECTION.
    MAIN-ROUTINE.
        PERFORM OPENS.
        PERFORM WORK-STATION-READS UNTIL STOP-RUN = 1.
        PERFORM CLOSES.
        STOP RUN.
    WORK-STATION-READS.
        IF INVALID-EMPNO = 1
           WRITE SCREEN-RECORD FROM ERRMSG1
           FORMAT IS 'ERRMSG1'
           STARTING AT LINE 4
           READ SCREEN-FILE-PR3 RECORD
2         MOVE INDCATOR-ON TO IFIELDS
           MOVE EMPRECD TO SCREEN-RECORD
3         WRITE SCREEN-RECORD
              FORMAT IS 'EMPRECD'
              STARTING AT LINE 6
              INDICATORS ARE INDCATOR-FIELD
        ELSE IF ERROR-FOUND = 0
          MOVE SPACES TO SCREEN-RECORD-CLEAR
          MOVE ZEROS TO IFIELDS
          WRITE SCREEN-RECORD
             FORMAT IS 'EMPRECD'
             STARTING AT LINE 4.
        READ SCREEN-FILE-PR3 RECORD INTO EMPRECD.
        PERFORM SET-SWITCHES.
        IF CONTINUE = 1
           PERFORM READ-MASTER.
        IF CONTINUE = 1
        PERFORM UPDATE-MASTER.
```

Figure 3.3. Example of indicators and conditional bypass (Part 2 of 3)

```
READ-MASTER.
    MOVE EMPNO TO RKEY.
    SUBTRACT 1000 FROM RKEY GIVING RKEY.
    READ EMPMAS-FILE
       INVALID KEY
          MOVE 0 TO CONTINUE
          MOVE 1 TO ERROR-FOUND, INVALID-EMPNO.
UPDATE-MASTER.
    MOVE SCREEN-RECORD TO EMPRECD.
    REWRITE EMPRECD
       INVALID KEY
          MOVE 1 TO INVALID-EMPNO
          MOVE 0 TO CONTINUE.
OPENS.
    OPEN I-O SCREEN-FILE-PR3
             EMPMAS-FILE.
CLOSES.
    CLOSE EMPMAS-FILE
          SCREEN-FILE-PR3.
SET-SWITCHES.
    MOVE 0 TO INVALID-EMPNO,
              STOP-RUN, ERROR-FOUND.
    MOVE 1 TO CONTINUE.
```

DDS SOURCE LISTING

```
01A          R EMPRECD            B      DSPATR(CS UL)
02A            ENAME         20X    01 11CHECK(DR RB)
03A  51                                  CHECK(BY) 4
04A                               O      'EMPLOYEE NAME' DSPATR(CA)
05A            STRAD         20A    02 11CHECK(DR RB)
06A  52                                  CHECK(BY) 5
07A                               O      'STREET ADDRESS' DSPATR(CA)
08A            CTYST         20X    03 11CHECK(DR RB)
 9A  53                                  CHECK(BY) 6
10A                               O      'CITY, STATE'  DSPATR(CA)
11A            EMPNO          5D    04 11CHECK(DR RZ)
12A                               O      'EMPLOYEE NUMBER' DSPATR(CA)

01A          R ERRMSG1            B      DSPATR(RI HI BL)
02A                               O 01 02'EMPLOYEE NUMBER INCORRECT. MUST BE-
02A                                       MORE THAN 1000 OR LESS THAN 2000.'
```

Figure 3.3. Example of indicators and conditional bypass (Part 3 of 3)

Chapter 4. Data Communications Programming with COBOL

This chapter provides the information necessary to code the COBOL statements that allow data exchange between two systems over a communications link; throughout this chapter and book, this exchange of data is called *data communications.*

Also, the chapter has the following information:

- A summary of COBOL data communications capabilities
- The IBM 5280 facilities needed to execute a COBOL program

COBOL also provides support of data communications giving more direct control over all communication facilities than as described in this chapter. See Appendix A for a description of direct data communications support.

To execute a COBOL program using data communications, the proper communications environment must be established using the preparation utilities. Information on this subject can be found in the *Communications Reference Manual.*

Uses of Data Communications Programs

You might want to write a COBOL data communications program:

- When you want to interact with an operator within one system and need to access data in a remote system.
- When you want to process data online as it is received from the host.
- When you want to match your 5280 communications program with a host application program.
- When the IBM 5280 Data Communications Utilities don't meet your needs. The functions provided by these utilities are sending batch data, receiving batch data, and sending inquiries and receiving replies. Complete information on the functions of the utilities and how to use them is given in the *Communications Reference Manual.*

Eligible Systems

The systems with which a COBOL program can communicate are the same as those for an assembler language program. See the chapter "Data Communications with Assembler Language" in the *Communications Reference Manual* under "BSC Programming with Assembler Language" and "SNA Programming with Assembler Language" for a list of the eligible systems.

COBOL Programs and the Communications Access Method

A COBOL program works with the communications access method (CAM) to transfer data. Part of the tasks in communicating is done by CAM; you must code the others in your COBOL program.

COBOL Program Responsibilities

A COBOL program interfaces to CAM with a series of library routines. The COBOL compiler places these library routines in your object module at compilation. You initiate a desired communications task (reading records, for example) by initiating the appropriate library routine from your program, in which you code the following information:

- CALL verbs, which initiate the COBOL library routines that open and close data sets, and read and write records.
- Data items, which designate input and output areas in your COBOL program and other information needed by the library routines.
- Data items, in which the library routines put status information after they complete execution. Your program can look at this return information to decide upon subsequent action.

Details on how to code the above information follows after the next section.

System Requirements for Communication Programs

To write and execute a COBOL data communications program, you need:

- The IBM 5280 Communications Utilities licensed program, which contains the communications access method (CAM) and the utilities that prepare the communications environment
- A communications adapter
- When using SNA, the elapsed time counter feature
- A partition with sufficient storage to run the COBOL program. (The amount of storage required by the program is printed on the compiler output listing if either the LOAD or the LIST option is in effect during compilation. See Chapter 7 for information on specifying compiler options.)

For the minimum size of the partition required by CAM, see Chapter 1 of the *5280 Communications Reference Manual.*

Before Running a Data Communication Program

Before running a data communication program, you must ensure that the proper communications environment exists for your program. The communications environment is established with two of the IBM 5280 Data Communication Preparation Utilities:

- The Communications Configuration Utility, to which the communications environment is described; the utility places this information in a *communications configuration record.*
- The Communications Load Utility, which loads the communications access method (CAM) into main storage.

See the *IBM 5280 Communications Reference Manual*, for detailed information on these two utilities.

Writing COBOL Communication Statements

COBOL provides four routines which handle communications; their names and functions are shown in the following table:

Routine	Function
AVCHOPEN	Open routine. Prepares for read and write operations between two systems.
AVCHCLOZ	Close routine. Provides three options: (1) ends communications operations or (2) signals an end-of-file to the receiver and the beginning of a new file or (3) turns around the direction of transmission from send to receive.
AVCHREAD	Read routine. Reads a record into a buffer in the COBOL program from the sending station.
AVCHWRT	Write routine. Writes a record from a buffer in the COBOL program to the receiving station.

Figure 4.1. The COBOL library routines that perform data communications

These routines are invoked in the COBOL program with CALL statements. The following sections explains how to code the CALL statements, the parameters you must supply, and the information returned to your program after the routines have executed.

Communication Routine Parameters

In each CALL to a communications routine, you point to a parameter list in either the Working Storage Section or the Linkage Section of your program (the detailed layouts of these areas are given in the routine descriptions that follow).

Communication Program Example — Explanation

Figure 4.4 at the end of the chapter gives an example of a COBOL program using data communications, both sending and receiving data; the program in the example:

1. Reads records from a diskette file containing information for updating an employee master file on line at another system.
2. Transmits the records to the remote system.
3. After sending all the update records, reads records sent by the remote system confirming receipt and giving diagnostics, if any.

The text in the following section will refer to this example by numbered keys in explaining the pertinent COBOL statements.

Open Routine

The following is the format of the CALL statement used to initiate the Open routine:

```
CALL "AVCHOPEN" USINGparameter-1 buffer-2
```

parameter-1 points to a status area you code in either the Working Storage Section or the Linkage Section in the Data Division of your program in the following format:

```
01  parameter-1
    02 return-code    PIC 99.
    02 FILLER         PIC 9999.
    02 buffer-size    PIC 9999.
```

return-code is a two-digit completion code returned by the Open routine after execution; the codes that can be returned and their meanings are given in figure 4.2.

buffer-size is the length in bytes of the records to be read or the *maximum* length of records to be written.

buffer-2 is a 01 level data-item which you define in the Data Division of your program for the area from which data is to be sent or in which data is to be received.

An example of the statements used to call the Open routine is shown at **8** in figure 4.4. Note that the CALL statement points to the parameter list `OPEN-READ-WRITE-PARAMS` at **2** and the buffer EMPRECD at **1**, defined in the Data Division.

☞ *For SNA users*: If the host requires logon data, the first 80 bytes of *buffer-2* must contain the data when the Open routine is invoked in the program.

☞ *For switched line users*: A timeout may occur which causes a line drop if data transfer doesn't take place within a specified amount of time. You might want to prompt the operator with a STOP statement, as is shown at **5** in figure 4.4, to ensure the stations are connected.

For BSC, a line connection with the remote station should be made immediately before the first READ or WRITE; for SNA, before the OPEN.

Code	Meaning
00	Execution was successful.
04	For the Read routine only. End of data. Successful completion.
08	Exception condition. Conditions on the line or at the remote station caused an unsuccessful completion of the last message, but there is no error in the COBOL program.
12	Application error. An error in the logic of the COBOL communication statements. For example: the program attempted to call the Read routine without having previously called the Open routine.
16	Permanent error. An unrecoverable error in, for example, the hardware, CAM, the line, etc., has been found.

Figure 4.2. Return codes from the COBOL communication library routines.

Close Routine

The format of a call to the Close routine is:

```
CALL "AVCHCLOZ" USING parameter-1
```

parameter-1 points to a status area you code in either the Working Storage Section or the Linkage Section in the Data Division of your program in the following format:

```
01  parameter-1
    02  return code    PIC 99.
    02  FILLER         PIC 9999.
    02  close-option   PIC A.
```

return-code is a two-digit completion code returned by the Close routine after execution; the codes that can be returned and their meanings are shown in figure 4.2.

close-option is a one-character code that can cause one of three conditions by the Communications access method (CAM). The actions and the subsequent action taken by the user program are shown in the following figure:

close-option	**CAM/User Action**
F	Terminates communications operations. To resume communications, the Open routine must again be called.
E	Sends end-of-file indication to receiver. (A return code of 04 is posted if the receiver is using IBM 5280 COBOL data communications.) The sender can then begin transmission of another file to the receiver. Don't call the Open routine after a close with the E option.
R	Turns around the line direction from transmit (through the Write routine) to receive (through the Read routine). An error in the *return-code* field after execution of a close with the *R* option means the turn-around was not successful. Don't call the Open routine after a close with the *R* option. Use of the *R* option between transmit and receive operations is optional.
Any character other than the above	F option assumed.

Figure 4.3. Options available with the Close routine

An example of the statements used to call the Close routine is shown at 9 and 10. At 9, the *R* close option is used, because the line direction is changed from transmit to receive. At 10, the *F* close option is used to end the communications link.

Note that the CALL statement points to the parameter list **CLOSE-PARAMS** at 3.

☞**Note**: don't call the Close routine if, after receiving data using the Read routine, you want to indicate an error condition found by your program. If the Close routine is not executed before stopping the COBOL program with a STOP RUN verb, a negative response will be sent to the sending station.

Terminating Without Close

At 4 in the example shown in figure 4.4, the run is stopped if either a disk error or printer error occurs without calling the Close routine, thus informing the sending program that the run was not successful.

Read Routine

The format of a call to the Read routine is:

```
  CALL "AVCHREAD" USINGparameter-1
```

parameter-1 points to a status area you code in either the Working Storage Section or the Linkage Section in the Data Division of your program in the following format:

```
01  parameter-1
    02 return-code  PIC 99.
    02 FILLER       PIC 9999.
```

return-code is a two-digit completion code returned by the Read routine after execution; the codes that can be returned and their meanings are shown in figure 4.2.

An example of the statements used to call the Read routine is shown at **7** in Figure 4.4. The statement points to the parameter list **2** `OPEN-READ-WRITE-PARAMS` in the Data Division.

Padding Unused Bytes in Read Buffer

If the record read in is smaller than the *buffer-size* (defined in the Open routine), the remaining bytes will be padded with blanks.

Write Routine

The format of a Write statement is:

```
  CALL "AVCHWRT" USINGparameter-1
```

parameter-1 points to a status area you code in either the Working Storage Section or the Linkage Section in the Data Division of your program in the following format:

```
01  parameter-1
    02 return-code  PIC 99.
    02 FILLER       PIC 9999.
    02 record-size  PIC 9999.
```

return-code is a two-digit completion code returned by the Write routine after execution; the codes that can be returned and their meanings are shown in figure 4.2.

record-size is the length in bytes that of the records you want to send. It cannot be more than the *buffer-size* value you specify in the Open routine.

An example of the statements used to call the Write routine is shown in **6** in Figure 4.4.

Communication Program Example — Listing

The following is a listing of the COBOL communications program described earlier in this chapter under "Example of COBOL Communications Program". Note that all information that normally appears to the left of the statement numbers (the COPY ID column) has been truncated for this example.

```
     IDENTIFICATION DIVISION.
     PROGRAM-ID.  SEND-RECEIVE.
     AUTHOR.  A NAME.
     ENVIRONMENT DIVISION.
     CONFIGURATION SECTION.
     SOURCE-COMPUTER. IBM-370.
     OBJECT-COMPUTER. IBM-5280.
     INPUT-OUTPUT SECTION.
     FILE-CONTROL.
         SELECT PRINT-FILE ASSIGN TO PRINTER
           ORGANIZATION IS SEQUENTIAL
           ACCESS IS SEQUENTIAL
           FILE STATUS IS PRINT-FILE-STAT.
         SELECT UPDATE-FILE ASSIGN TO DISK
           ORGANIZATION IS RELATIVE
           ACCESS IS SEQUENTIAL
           RELATIVE KEY IS RKEY
           FILE STATUS IS UPDATE-FILE-STAT.
     DATA DIVISION.
     FILE SECTION.
     FD  UPDATE-FILE
         LABEL RECORDS ARE STANDARD.
1       COPY DDS-EMPRECD.
0001 01      EMPRECD.
0002     02  ACREC    PIC X(00002).
0004     02  EMPNO    PIC X(00005).
0006     02  ENAME    PIC X(00020).
0008     02  STRAD    PIC X(00020).
0010     02  CTYST    PIC X(00020).
0012     02  ZIPCD    PIC X(00005).
0016     02  BEGDT    PIC X(00006).
0018     02  SOSNO    PIC X(00009).
0020     02  MARST    PIC X(00001).
     FD  PRINT-FILE
         LABEL RECORDS ARE STANDARD.
     01  PRINT-RECORD.
         05  RCDCD PIC X(88).
     WORKING-STORAGE SECTION.
     01  SWITCHES.
         02 COMM-ERROR         PIC 9 VALUE 0.
         02 OPEN-ERROR         PIC 9 VALUE 0.
         02 CLOSE1-ERROR       PIC 9 VALUE 0.
         02 CLOSE2-ERROR       PIC 9 VALUE 0.
         02 WRITE-SEND-ERROR   PIC 9 VALUE 0.
         02 READ-RECEIVE-ERROR PIC 9 VALUE 0.
         02 STOP-READS         PIC 9 VALUE 0.
         02 STOP-WRITES        PIC 9 VALUE 0.
     77  UPDATE-FILE-STAT      PIC XX.
     77  PRINT-FILE-STAT       PIC XX.
     77  RKEY                  PIC 99999.
     77  RCOUNTER              PIC 99999 VALUE 0.
     77  CAUSE                 PIC X(8) VALUE SPACES.
2    01 OPEN-READ-WRITE-PARAMS.
         02 RETURN-CODE        PIC 9(2) VALUE 0.
         02 FILLER             PIC 9(4).
         02 RECORD-SIZE        PIC 9(4) VALUE 88.
```

Figure 4.4. Coding example: data communications (Part 1 of 3)

```
3   01 CLOSE-PARAMS.
        02 CRETURN-CODE       PIC 9(2) VALUE 0.
        02 FILLER             PIC 9(4).
        02 CLOSE-OPTION       PIC A VALUE SPACES.
    PROCEDURE DIVISION.
    DECLARATIVES.
4   DISK-ERROR SECTION.
       USE AFTER ERROR PROCEDURE ON UPDATE-FILE.
    DISKX.
            DISPLAY "ERROR ON UPDATE-FILE I/O DURING SEND".
            DISPLAY "FILE STATUS IS " UPDATE-FILE-STAT.
            DISPLAY "TRANSMISSION STOPPED. RUN STOPPED.".
            STOP RUN.
    PRINT-ERROR SECTION.
       USE AFTER ERROR PROCEDURE ON PRINT-FILE.
4   PRINTERX.
            DISPLAY "ERROR ON PRINTER-FILE I/O DURING WRITE".
            DISPLAY "FILE STATUS IS " PRINT-FILE-STAT.
            DISPLAY "TRANSMISSION STOPPED. RUN STOPPED.".
            STOP RUN.
    END DECLARATIVES.
    EXECUTE SECTION.
    MAIN-ROUTINE.
        PERFORM OPEN1.
5       STOP
         "ENSURE LINE CONNECTED TO REMOTE STATION & HIT ENTER.".
        IF COMM-ERROR IS EQUAL TO 0
           PERFORM REMOTE-STATION-WRITES
              UNTIL STOP-WRITES IS EQUAL TO 1.
        IF COMM-ERROR IS EQUAL TO 0
           PERFORM CLOSE1.
        IF COMM-ERROR IS EQUAL TO 0
           DISPLAY RCOUNTER
              " RECORDS SENT SUCCESSFULLY; NOW AWAITING RESPONSE"
           PERFORM OPEN2.
        IF COMM-ERROR IS EQUAL TO 0
           PERFORM REMOTE-STATION-READS
              UNTIL STOP-READS IS EQUAL TO 1.
        IF COMM-ERROR IS EQUAL TO 1
           PERFORM ERROR-ROUTINE
        ELSE DISPLAY
           "SESSION COMPLETED SUCCESSFULLY.".
        PERFORM CLOSE2.
        STOP RUN.
    REMOTE-STATION-WRITES.
        READ UPDATE-FILE RECORD
           AT END
              MOVE 1 TO STOP-WRITES.
        IF STOP-WRITES IS EQUAL TO 0
           IF ACREC IS EQUAL TO "A"
              ADD 1 TO RCOUNTER
6                CALL "AVCHWRT" USING OPEN-READ-WRITE-PARAMS.
        IF RETURN-CODE GREATER THAN 0
           MOVE 1 TO WRITE-SEND-ERROR, COMM-ERROR, STOP-WRITES.
```

Figure 4.4. Coding example: data communications (Part 2 of 3)

```
    REMOTE-STATION-READS.
7        CALL "AVCHREAD" USING OPEN-READ-WRITE-PARAMS.
          IF RETURN-CODE IS EQUAL TO 4
             MOVE 1 TO STOP-READS
          ELSE IF RETURN-CODE GREATER THAN 4
             MOVE 1 TO COMM-ERROR,
                       READ-RECEIVE-ERROR,
                       STOP-READS.
        IF STOP-READS IS EQUAL TO 0
           WRITE PRINT-RECORD FROM EMPRECD.
8   OPEN1.
        OPEN INPUT UPDATE-FILE.
        CALL "AVCHOPEN" USING OPEN-READ-WRITE-PARAMS EMPRECD.
           IF RETURN-CODE GREATER THAN 0
              MOVE 1 TO COMM-ERROR,
                        OPEN-ERROR.
        STOP "PRESS ENTER TO CONTINUE".
9   CLOSE1.
        CLOSE UPDATE-FILE.
        MOVE "R" TO CLOSE-OPTION.
        CALL "AVCHCLOZ" USING CLOSE-PARAMS.
           IF CRETURN-CODE GREATER THAN 0
              MOVE 1 TO COMM-ERROR
              MOVE 1 TO CLOSE1-ERROR.
    OPEN2.
        OPEN INPUT UPDATE-FILE
             OUTPUT PRINT-FILE.
10   CLOSE2.
        CLOSE UPDATE-FILE.
        MOVE "F" TO CLOSE-OPTION.
        CALL "AVCHCLOZ" USING CLOSE-PARAMS.
           IF CRETURN-CODE GREATER THAN 0
              MOVE 1 TO COMM-ERROR
              MOVE 1 TO CLOSE2-ERROR.
    ERROR-ROUTINE.
        IF OPEN-ERROR IS EQUAL TO 1
           MOVE "OPEN" TO CAUSE
        ELSE IF CLOSE1-ERROR IS EQUAL TO 1
           MOVE "CLOSE1" TO CAUSE
        ELSE IF CLOSE2-ERROR IS EQUAL TO 1
           MOVE "CLOSE2" TO CAUSE
        ELSE IF WRITE-SEND-ERROR IS EQUAL TO 1
           MOVE "WRITE" TO CAUSE
        ELSE IF READ-RECEIVE-ERROR IS EQUAL TO 1
           MOVE "READ" TO CAUSE.
        DISPLAY "TRANSMISSION INTERRUPTED"
        DISPLAY "ERROR FOUND DURING " CAUSE "RUN STOPPED."
        DISPLAY "RETURN CODE= " RETURN-CODE.
```

Figure 4.4. Coding example: data communications (Part 3 of 3)

Chapter 5. Diskette Input/Output

Diskette Input/Output (I/O) is the transfer of data between a program and IBM 5280 diskette devices. The chapter provides examples of the statements you write to perform diskette I/O, and some considerations you must make in writing I/O routines, namely:

- Choosing a file organization (sequential, relative, or indexed)
- Choosing the access method (sequential or random)
- Assigning data sets and devices to file statements in the program
- Sharing files among more than one program
- Guidelines and examples in coding routines to create, read, and update the three types of file organizations

Input-Output Summary

A COBOL I/O operation that transmits data to or from main storage involves three elements:

- The COBOL statements that define and request the operation
- An access method that performs the operation
- A file on an actual device on which the operation is performed

To perform a diskette I/O operation, you must code several statements in your program to make the operation possible. Some of these statements are:

- The FILE-CONTROL paragraph, which relates your file to the diskette device, and describes the access mode, file organization, and other factors relating to the file.
- The OPEN and CLOSE verbs in the Procedure Division. OPEN builds the environment necessary to transmit data to and from a device; CLOSE completes all unfinished I/O operations.

  Before attempting to transmit data, you must always issue an OPEN; you should always issue a CLOSE to ensure that any data remaining in an intermediate buffer is transferred.
- READ, WRITE, REWRITE, and DELETE verbs in the Procedure Division that cause records to be transmitted either to or from the files you specify in the ASSIGN clause.

The remainder of the chapter provides more detailed information on the items just summarized.

File Organization and Access Method

You define a file organization and an access method in the FILE-CONTROL paragraph of your program. The types of file organizations and access methods you can define are:

- Sequential, for which one access method is possible: *sequential.* For this organization, you specify ORGANIZATION IS SEQUENTIAL in the FILE-CONTROL paragraph of your program.

- Relative, for which two access methods are possible: *sequential* and *random*. For this organization, you specify ORGANIZATION IS RELATIVE in the FILE-CONTROL paragraph of your program.
- Indexed, for which two access methods are possible: *sequential* and *random*. For this organization, you specify ORGANIZATION IS INDEXED in the FILE-CONTROL paragraph of your program.

As stated, you can specify one of two access methods, depending on the file organization: sequential (ACCESS IS SEQUENTIAL) or random (ACCESS IS RANDOM).

Random Access Method

For random access, the order of reference to a record is determined by a value you specify in your program. For records with a relative organization, this value is called a *relative key*; for records with an indexed organization, this value is known as a *record key*.

Sequential Access Method

For sequential access, the order of reference to a record is determined by the position of the record in the file. That is, records are read or written serially as follows:

- For files with a sequential organization, according to their physical location.
- For files with a relative organization, according to their relative key.
- For files with an indexed organization, according to their record key.

In the example that follows, the term *current record pointer* (CRP) is used to explain the relative position of a record in a file after an OPEN or READ verb is executed. The concept of the CRP applies when the sequential access method is used for files with sequential, random, or indexed organizations.

```
OPEN   INPUT                          CRP=record 1
READ           Read record 1          CRP=record 2
READ           Read record 2          CRP=record 3
      •
      •
      •
READ           Read record X          CRP=X+1
REWRITE        Rewrite record X       CRP=X+1
READ           End-of-file occurs
READ           Fails (Status Key 94)  No CRP
```

In order for a READ to be successful, the previous OPEN or READ for that file must have been successful. You can determine whether or not a READ or OPEN is successful by examining the Status Key. (The Status Key is a data-item coded in your program as described under "Status Key" under "Error Processing Options" later in this chapter.)

When all the records in a file are to be processed, using the sequential access method will be faster than using the random access method.

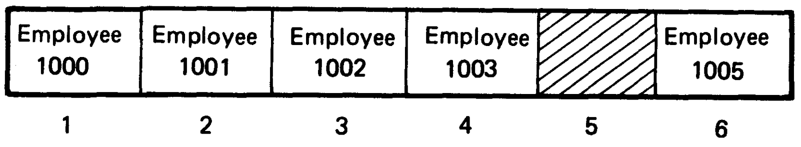
Sequential Organization

In files with a sequential organization, records are placed into the data set one after the other: the first record entered occupies the first position in the file, the second occupies the second position, and so on. Records are retrieved in the same order. The first record is read, then the second, and so on.

Relative Organization

In files with a relative organization, a relationship exists between records and their positions in the file. The relative position of a record might be equal to a program counter or a field value. The relative position might also be derived by a formula or conversion technique.

The following is an example of a file with relative organization:



In this example, a company has a maximum of 1000 employees, and they are assigned employee numbers of 1000 through 1999. Record 1 contains data on employee 1000, record 2 contains data on employee 1001, and so forth. The conversion technique is simply to take the employee number and to subtract 999 to obtain the relative record number.

The file does not need to be created sequentially or read sequentially, although either is quite possible. Notice that record 5 is a null record. This is because either no employee was ever assigned that number, or the employee has left the company. In either case the storage space is unused.

Indexed Organization

In indexed files, a relationship exists between a record and a key you designate in the record. The key for each record must be unique. This allows a program to refer to each record by its key.

COBOL supports two types of indexed files. They are described later in this chapter in the section "Processing Files with an Indexed Organization".

Record Formatting

The format of the record in an IBM 5280 system data set is determined when you allocate the data set as described in Chapter 8. The RECORD CONTAINS and BLOCK CONTAINS clauses don't affect this format.

The size of the record as defined in the COBOL program and the size of the record as allocated must always be the same.

FILE-CONTROL Paragraph

You describe your file and assign a diskette device in the FILE-CONTROL paragraph. Here are some of the entries you make in this paragraph:

```
SELECT file-name
ASSIGN TO DISK
["device-address1"[ "[*volid1.]dsname1"]]
[INDEX ["device-address2"[ "[*volid2.]dsname2"]]]
[SIZE nnnn]
[SHARE | SHARER]
      •
      •
      •
```

☞**Note**: Other clauses in the FILE-CONTROL section include the ORGANIZATION, ACCESS MODE, RECORD KEY, RELATIVE KEY, and FILE STATUS clauses. They are described in the *Language Reference Manual*; some are shown in the examples given later in this chapter.

The INDEX and SIZE clauses are discussed later in this chapter under "Processing Files with an Indexed Organization". SHARE and SHARER are discussed under "Sharing Files".

The following text describes the entries SELECT clause and the *assignment-name* entries of the ASSIGN clause.

file-name is the name of the file you specify in the associated File Description (FD) Entry.

DISK tells COBOL that the device is a diskette.

You can optionally specify the following parameters:

device-address1 is the physical address (4000 or 4400, for example) or logical address of the device on which the diskette must be mounted.

volid1 is the identification given when the volume was initialized with the diskette initialization utility.

dsname1 is the name given to the data set when it was allocated, usually assigned with the diskette label maintenance utility.

If you specify only the required (*file-name*) parameter, the operator must respond to the following prompt that appears on the screen when the program is loaded.

```
Enter the following information for file file-name
Dataset name:
Device address:
Owner id:
                        Press ENTER
```

No prompt will be displayed under the following conditions:

- You specify *device address1, volid1* and *dsname1.*
- You specify a VALUE OF *owner-id* clause (described below) in your program when the volume is protected.

Otherwise, the operator can respond to this prompt as described under "Prompts for Diskette Files" in Chapter 8.

FD Entry — VALUE OF OWNER-ID Clause

You can write a VALUE OF OWNER-ID clause in the field description (FD) entry. This clause allows you to control the use of protected diskette volumes by the operator.

The operator must fill in the *Owner-id* field (1) if the volume is protected and (2) if you *don't* code a VALUE OF *owner-id* clause under the FD entry in your program.

Whether or not a volume is protected is determined by the setting of the accessibility byte. This byte is set under the "modify volume label" option of the diskette label maintenance utility. See *Utilities Reference/Operations Manual* for detailed information on how to protect a volume.

The VALUE OF *owner-id* clause is coded as follows:

```
FD  file-name
    VALUE OF OWNER-ID IS "BELL"
    LABEL RECORDS ARE STANDARD.
```

In the above example the *owner-id* BELL was specified when the data set was initialized with the diskette initialization utility (also described in the *Utilities Reference/Operations Manual*).

Sharing Files

Sharing files refers to the use of a file by more than one program executing at the same time. The ability of another program to use a file at the same time as your program depends on how the following COBOL language statements are coded in the two programs:

- The SHARE option in the SELECT clause.
- The OPEN verb, in which the modes OUTPUT, I-O (for update), INPUT, or EXTEND mode can be coded.

In general, you can code your program so that diskette files are:

- Unshared
- Shared read
- Shared read/write

Unshared Files

When a file is unshared, only the program that opens the file has access to it, both for reading and for writing. If another program attempts to access the file, the requesting program will receive Status Key value of 93. This indicates the file is already opened. Files are always unshared unless you specifically specify otherwise, as will be shown later in this chapter.

Shared Files

When you specify SHARER (for *shared read*) in the FILE-CONTROL paragraph, another program can access the file for reading data, but cannot update it.

When you specify SHARE (for *shared read* and *shared read/write*) files can be accessed by more than one program at a time. All programs can both read and write to the file.

When Files Can Be Shared

Whether or not a second program can access a file that has already been opened depends on:

- The mode (INPUT, OUTPUT, I-O, or EXTEND) coded with OPEN in the first program.
- The share option (blank, SHARER, or SHARE) coded in the SELECT clause of the first program.
- The mode coded with OPEN in the second program.
- The share option (blank, SHARER, or SHARE) coded in the SELECT clause of the second program.

☞**Note**: You cannot add records to files with indexed organizations if those files are being shared with other programs.

Figure 5.1 shows the combinations of the SHARE/SHARER options and the OPEN modes when files are to be shared among programs.

IF A PROGRAM USES ...		THEN OTHER PROGRAMS CAN USE ...	
As the Share Option:	As the OPEN Mode:	As the Share Option:	As the OPEN Mode:
SHARE	I-O	SHARE	I-O or INPUT
SHARE	INPUT	SHARE or SHARER	I-O or INPUT
SHARER	I-O	SHARE	INPUT
SHARER	INPUT	SHARE or SHARER	INPUT

Figure 5.1. Valid combinations of SHARE/SHARER and OPEN modes

When a file is opened in the INPUT mode, then only READ operations can take place. When a file is opened in the I-O mode, then, depending on the file organization, READ, WRITE, REWRITE, and DELETE operations can take place.

No Sharing with OUTPUT and EXTEND Modes

The share options are ignored when the OPEN verb specifies a mode of OUTPUT or EXTEND. Files opened in these modes are always opened for exclusive use and cannot be shared while they are so opened. If the file is then subsequently opened in another mode such as INPUT or I-O, the share options on the SELECT clause are then used to determine how the file will be shared.

I/O Error Processing

Four options are available determining the outcome of an I/O request, and for detecting and handling I/O errors:

- The Status Key
- The EXCEPTION/ERROR Procedure
- The Invalid KEY clause
- The AT END Clause

This section explains the use of these options; the syntax and rules for coding these options is given in detail in the *5280 COBOL Language Reference*.

Status Key

The Status Key is a 2-character data-item you define in the Data Division of your program and name in the FILE STATUS clause. In the example in figure 5.2, PRINT-FILE-STAT at $\boxed{1}$ and UPDATE-FILE-STAT at $\boxed{2}$ are defined in the WORKING-STORAGE SECTION at $\boxed{3}$.

Upon return to the COBOL program, the Status Key contains a value that defines the status of the last request on the file.

It is recommended that you define a Status Key for all files *and* that your COBOL program check the contents after each I/O request. Otherwise, errors may go undiscovered by the program, producing results that may be both destructive and difficult to diagnose.

If neither a Status Key nor an EXCEPTION/ERROR procedure (described in the next section) is present and an error occurs, the program will display a message in the Status Line at the top of the screen. A message identifier in the format 92nn will precede the message; *nn* is the code that would have been placed in the Status Key had it been present.

See Appendix B for a complete list of the values that can be placed in the Status Key, their meanings, and a cross-reference between Status Key values and the message identifiers as displayed in the Status Line.

EXCEPTION/ERROR Procedures

You can also code a procedure to handle errors using the EXCEPTION/ERROR declarative. In the example in figure 5.2, each time an error occurs on UPDATE-FILE or PRINT-FILE, an ERROR/EXCEPTION routine at $\boxed{4}$ or $\boxed{5}$ receives control, displays some messages, and stops the run.

The EXCEPTION/ERROR procedure is used only when a file is in open status. Therefore, if any operation is attempted against a file which has already been closed, or was never opened, then the EXCEPTION/ERROR procedure is not executed. COBOL will return a Status Key value of 92.

```
        IDENTIFICATION DIVISION.
        PROGRAM-ID.  SEND-RECEIVE.
                  •
                  •
                  •
        INPUT-OUTPUT SECTION.
        FILE-CONTROL.
            SELECT PRINT-FILE ASSIGN TO PRINTER
              ORGANIZATION IS SEQUENTIAL
              ACCESS IS SEQUENTIAL
1             FILE STATUS IS PRINT-FILE-STAT.
            SELECT UPDATE-FILE ASSIGN TO DISK
              ORGANIZATION IS SEQUENTIAL
              ACCESS IS SEQUENTIAL
2             FILE STATUS IS UPDATE-FILE-STAT.
                  •
                  •
                  •
3       WORKING-STORAGE SECTION.
        77  UPDATE-FILE-STAT      PIC XX.
        77  PRINT-FILE-STAT       PIC XX.
                  •
                  •
                  •
        PROCEDURE DIVISION.
        DECLARATIVES.
        DISK-ERROR SECTION.
4          USE AFTER ERROR PROCEDURE ON UPDATE-FILE.
        DISKX.
                DISPLAY "ERROR ON UPDATE-FILE I/O DURING SEND".
                DISPLAY "FILE STATUS IS " UPDATE-FILE-STAT.
                DISPLAY "TRANSMISSION STOPPED. RUN STOPPED.".
                STOP RUN.
        PRINT-ERROR SECTION.
5          USE AFTER ERROR PROCEDURE ON PRINT-FILE.
        PRINTERX.
                DISPLAY "ERROR ON PRINTER-FILE I/O DURING WRITE".
                DISPLAY "FILE STATUS IS " PRINT-FILE-STAT.
                DISPLAY "TRANSMISSION STOPPED. RUN STOPPED.".
                STOP RUN.
        END DECLARATIVES.
        EXECUTE SECTION.
        MAIN-ROUTINE.
            PERFORM OPEN1.
                  •
                  •
                  •
```

Figure 5.2. COBOL statements for Status Key and EXCEPTION/ERROR routine

The INVALID KEY Clause

When using the random access method, you can specify an INVALID KEY clause with READ, WRITE, REWRITE, and DELETE verbs for files having an indexed or relative organization. The clause consists of a set of COBOL statements you want executed each time an invalid key is found.

An EXCEPTION/ERROR procedure, if specified, will not be executed.

If the FILE STATUS clause is specified, a value is placed in the Status Key to indicate the condition.

The AT END Clause

You can specify an AT END clause with a READ in sequential access mode. When the end-of-file is detected, control is passed to the imperative statement that follows AT END. An EXCEPTION/ERROR procedure, if specified, will not be executed.

If the FILE STATUS clause is specified, a value is placed in the Status Key to indicate the condition.

Error Handling Considerations

Figure 5.3 shows the actions taken for all the combinations of AT END, INVALID KEY, and EXCEPTION/ERROR procedure based on the first digit of the Status Key.

☞Note the following:

- The return is always to the next verb unless the request that caused the error contained an AT END or INVALID KEY clause.
- The EXCEPTION/ERROR procedure is executed only if the file is in the open status.
- If an AT END or INVALID KEY clause is present, an EXCEPTION/ERROR procedure will not be executed when either of these two conditions occur.

First Digit of Status Key	No EXCEPTION/ERROR Procedure		With EXCEPTION/ERROR Procedure	
	AT END/INVALID KEY	No AT END/INVALID KEY	AT END/INVALID KEY	No AT END/INVALID KEY
0	The verb after the AT END or INVALID KEY clauses executes.	See Note 1 and Note 2.	The verb after the AT END or INVALID KEY clause executes.	The verb after the I/O request executes.
1 (AT END condition)	The AT END clause executes.	See Note 1.	The AT END clause executes.	The EXCEPTION/ ERROR procedure executes, followed by the verb after the I/O request.
2 (INVALID KEY condition)	The INVALID KEY clause executes.	See Note 2.	The INVALID KEY clause executes.	The EXCEPTION/ ERROR procedure executes, followed by the verb after the I/O request.
3 (Permanent error)	The verb after the I/O request executes.	The verb after the I/O request executes.	The verb after the INVALID KEY or AT END clause executes.	The EXCEPTION/ ERROR procedure executes, followed by the verb after the I/O request.
9 (Other errors)	The verb after the I/O request executes.	The verb after the I/O request executes.	The verb after the INVALID KEY or AT END clause executes.	The EXCEPTION/ ERROR procedure executes, followed by the verb after the I/O request.

Note 1. The AT END phrase must be specified when ACCESS is sequential, when no EXCEPTION/ERROR routine is specified; otherwise, the compiler issues a severe error message and makes the object module unexecutable.

Note 2. The INVALID KEY phrase must be specified when ORGANIZATION IS INDEXED or ORGANIZATION IS RELATIVE, and ACCESS IS RANDOM are specified; otherwise, the compiler issues a severe error message and makes the object module unexecutable.

Figure 5.3. COBOL statement execution with AT END and INVALID KEY

Permanent I/O Errors

If an I/O error occurs while accessing a file, COBOL will set the Status Key to 30. COBOL will not accept additional READ or WRITE requests following an I/O error. Therefore, if an I/O error occurs, you should CLOSE the file. It may be necessary to create the file again.

Processing Files with Sequential File Organization

The table in figure 5.4 summarizes the COBOL statements used for reading, writing and updating a sequential file.

Division	Reading	Writing	Updating
Environment Division	SELECT ASSIGN FILE STATUS ACCESS IS SEQUENTIAL	SELECT ASSIGN FILE STATUS ACCESS IS SEQUENTIAL	SELECT ASSIGN FILE STATUS ACCESS IS SEQUENTIAL
Procedure Division	OPEN INPUT OPEN I-O READ CLOSE	OPEN OUTPUT OPEN EXTEND WRITE CLOSE	OPEN I-O READ REWRITE CLOSE

Figure 5.4. Statements used with sequential file organizations

ASCII File Processing

You can process ASCII-encoded files with a sequential organization by specifying the CODE-SET clause in the file description entry (FD entry). The rules for coding the CODE-SET clause are given in in the *5280 COBOL Language Reference.*

If ASCII files are to be processed, your 5280 system must have included ASCII support when it was installed.

Creating a Sequential File

The following guidelines apply in creating a file with a sequential organization:

1. Use WRITEs to create a sequential file.
2. Specify OUTPUT mode in the corresponding OPEN. Otherwise, a logic error will occur, setting the Status Key to 92.

   After the OPEN OUTPUT, any data formerly present in the file is no longer accessible.
3. If the end of the extent is reached on a diskette file, a boundary error occurs, setting the Status Key to 34.

See "COBOL Requirements for Data Sets" in Chapter 8 for the rules that apply when allocating data sets for your program.

Reading from a Sequential File

The following guidelines apply in reading records from a file with a sequential organization:

1. Use the READ verb to access the records.
2. INPUT or I-O mode must be specified in the corresponding OPEN. Otherwise, a logic error will occur, setting the Status Key to 92.
3. The records are read in the order in which they appear on the file. When the end of file is reached, the Status Key is set to 10. If the file does not

contain any records, the Status Key is also set to 10 on the first READ request.

Updating a Sequential File

The following guidelines apply in updating records from a file with a sequential organization:

1. Use a REWRITE to replace an existing record in a file.
2. The record to be rewritten must have been the last record read.
3. If there was no preceding READ, or if the preceding READ was unsuccessful, the Status Key is set to 92.
4. The I-O mode must be specified in the corresponding OPEN. Otherwise, a logic error will occur on a subsequent REWRITE, setting the Status Key to 92.

Multivolume Record Processing

The following rules and guidelines apply to multivolume files:

1. Only files with SEQUENTIAL organization can reside on multiple volumes.
2. A file is a multivolume file if the Header 1 label contains a 'C' or 'L' in the multivolume indicator field.
3. To create a multivolume file, you must allocate the data set on all volumes you will use and the first volume you mount must contain a 'C' or 'L' for the multivolume indicator.

   (The multivolume indicator can be set when the data set is allocated with the data set maintenance utility as described in the *Utilities Reference/Operations Manual.*)

   For multivolume diskette OUTPUT files, when end-of-volume is found, the next volume is requested; after the volume is mounted, the WRITE is done.

   COBOL will mark each volume with a 'C' in the HDR1 multivolume indicator field and set the appropriate sequence number for that volume. When a CLOSE verb is issued, COBOL will mark that volume as the last volume ('L').
4. To EXTEND a multivolume file, you must allocate the data set on all volumes you will be using. The first volume you mount must be the last volume of the original data set which is being extended, ('L') in the multivolume HDR1 indicator field. If you created the file using COBOL, this field will have been set by COBOL.

   When end-of-volume is found, the next volume is requested; after it is mounted, the Write is done.

   - COBOL will mark each volume with a 'C' in the HDR1 multivolume indicator field.
   - If the starting volume had a sequence number assigned to it, COBOL will assign the appropriate sequence number to all subsequent volumes.
   - When a CLOSE verb is issued, COBOL will mark that volume as the last volume.
5. For reading or updating a multivolume file, the first volume of the file must be mounted. If the HDR1 sequence number is blank, then COBOL assumes that the first volume is mounted. If it is not blank, COBOL will

ensure that the first volume is mounted and will prompt the operator to mount the first volume if it is not.

If end-of-volume is recognized during execution of a READ statement, and logical end-of-file has not been reached, the following actions are taken:

1. The next volume is requested.
2. After the volume is mounted, the first data record on it is made available.

   If sequence checking is being performed, as indicated by a nonblank sequence number field in the first volume, then COBOL will ensure that all subsequent volumes are mounted in their proper order. The operator will be prompted to mount the correct volume if one is mounted out of order.

Example

The program example in figure 5.5 updates `I-O-FILE`. It reads `INPUT-FILE` and the `I-O-FILE` until a match is found between `INPUT-EMPLOYEE-NUMBER` and `I-O-EMPLOYEE-NUMBER`. It then replaces the original record in `I-O-FILE` with `INPUT-RECORD`.

```
IDENTIFICATION DIVISION.
PROGRAM-ID.  UPDATE-SEQUENTIAL.
ENVIRONMENT DIVISION.
CONFIGURATION SECTION.
SOURCE-COMPUTER.  IBM-370.
OBJECT-COMPUTER.  IBM-5280.
SPECIAL-NAMES.  CONSOLE IS SCREEN.
INPUT-OUTPUT SECTION.
FILE-CONTROL.
    SELECT INPUT-FILE
      ASSIGN TO DISK.
    SELECT I-O-FILE
      ASSIGN TO DISK
      FILE STATUS IS SK.
DATA DIVISION.
FILE SECTION.
FD  INPUT-FILE LABEL RECORD STANDARD.
01  INPUT-RECORD.
    05   INPUT-EMPLOYEE-NUMBER    PICTURE   9(6).
    05   INPUT-EMPLOYEE-NAME      PICTURE   X(28).
    05   INPUT-EMPLOYEE-CODE      PICTURE   9.
    05   INPUT-EMPLOYEE-SALARY    PICTURE   9(6)V99.
    05   FILLER                   PICTURE   X.
FD  I-O-FILE      LABEL RECORDS STANDARD.
01  I-O-RECORD.
    05  I-O-EMPLOYEE-NUMBER      PICTURE 9(6).
    05  I-O-EMPLOYEE-NAME        PICTURE X(28).
    05  I-O-EMPLOYEE-CODE        PICTURE    9.
    05  I-O-EMPLOYEE-SALARY      PICTURE 9(6)V99.
    05  FILLER                   PICTURE X.
WORKING-STORAGE SECTION.
01  DISP-RECORD.
    05  OP-NAME      PICTURE   X(5).
    05  FILLER       PICTURE  XX VALUE SPACE.
    05  SK           PICTURE  XX VALUE "ZZ".
```

Figure 5.5. Example of processing files with a sequential organization (Part 1 of 2)

```
 PROCEDURE DIVISION.
 OPEN-FILES.
     OPEN INPUT INPUT-FILE I-O I-O-FILE.
     IF SK NOT = "00"
         MOVE "OPEN" TO OP-NAME
         PERFORM ERROR-OUT-1
         GO TO STOP-RUN.
     PERFORM READ-INPUT.
     PERFORM READ-UPDATE.
 UPDATE-TEST-LOOP.
     IF INPUT-EMPLOYEE-NUMBER = I-O-EMPLOYEE-NUMBER
         PERFORM I-O-REWRITE-1
         GO TO UPDATE-TEST-LOOP.
     IF INPUT-EMPLOYEE-NUMBER GREATER THAN I-O-EMPLOYEE-NUMBE
         PERFORM READ-UPDATE
         GO TO UPDATE-TEST-LOOP.
*  I-O-EMPLOYEE-NUMBER IS GREATER THAN INPUT-EMPLOYEE-NUMBER
*  NO MATCH HAS BEEN FOUND SO PUT OUT ERROR MESSAGE
     PERFORM ERROR-OUT-1.
     DISPLAY INPUT-RECORD UPON SCREEN.
     MOVE SPACES TO DISP-RECORD.
     PERFORM READ-INPUT.
     GO TO UPDATE-TEST-LOOP.
 CLOSE-FILES.
     CLOSE INPUT-FILE  I-O-FILE.
 STOP-RUN.
     STOP RUN.
 ERROR-OUT-1.
     DISPLAY DISP-RECORD UPON SCREEN.
 I-O-REWRITE-1.
     REWRITE I-O-RECORD FROM INPUT-RECORD.
     IF SK NOT = "00"
        MOVE "RWRTE" TO OP-NAME
        PERFORM ERROR-OUT-1.
     PERFORM READ-INPUT.
     PERFORM READ-UPDATE.
 READ-INPUT.
     READ INPUT-FILE AT END GO TO CLOSE-FILES.
 READ-UPDATE.
     READ I-O-FILE AT END MOVE "NOT FOUND" TO DISP-RECORD
     PERFORM ERROR-OUT-1 GO TO CLOSE-FILES.
```

Figure 5.5. Example of processing files with a sequential organization (Part 2 of 2)

Processing Files with a Relative Organization

The table in figure 5.6 shows the COBOL Statements available for relative file processing.

Division	Reading	Writing	Updating
Environment Division	SELECT ASSIGN ORGANIZATION IS RELATIVE RELATIVE KEY FILE STATUS ACCESS IS SEQUENTIAL ACCESS IS RANDOM	SELECT ASSIGN ORGANIZATION IS RELATIVE RELATIVE KEY FILE STATUS ACCESS IS SEQUENTIAL ACCESS IS RANDOM	SELECT ASSIGN ORGANIZATION IS RELATIVE RELATIVE KEY FILE STATUS ACCESS IS SEQUENTIAL ACCESS IS RANDOM
Procedure Division	OPEN INPUT OPEN I-O READ CLOSE	OPEN OUTPUT WRITE CLOSE	For ACCESS IS SEQUENTIAL OPEN I-O READ REWRITE DELETE CLOSE For ACCESS IS RANDOM OPEN I-O READ WRITE REWRITE DELETE CLOSE

Figure 5.6. Statements used with relative file organizations

Creating a Relative File

The following guidelines apply in creating a file with a relative organization:

1. Use WRITE statements to create a relative file.
2. Specify the OUTPUT mode in the corresponding OPEN. When a file is opened in OUTPUT mode, data formerly present in the file is no longer accessible.
3. If you specify ACCESS IS SEQUENTIAL, you can first write the desired number of records. A subsequent CLOSE causes null records to be inserted from the last record written, to the end-of-extent.

   Null records are empty records represented by the characters x'FF', which is inserted in the first byte of each record. Null records also contain the 5280 delete character specified when the data set was allocated.
4. If you specify ACCESS IS RANDOM, the OPEN causes null records to be inserted in the entire file.
5. Your program will never receive a null record. When you use the sequential access mode, only non-null records are returned; when you

use the random access mode, the Status Key is set to 23, indicating the record cannot be found.

6. Relative files must be on single volumes.

See "Allocating Data Sets for Program Files" in Chapter 8 for the rules that apply when allocating data sets for your program using IBM 5280 facilities.

Contents of Relative Key

When issuing a WRITE with the sequential access mode, the first record written will have relative record number one, the second two, the third three, and so on. If a relative record key was specified, it will be updated to contain the number of the record just written. If the end of extent is reached when a WRITE request is executed, an invalid key condition will occur, setting the Status Key to 24 (boundary error).

When issuing a WRITE with the random access mode, place the desired record number in the relative key data-item before issuing the WRITE. Invalid Key conditions after a WRITE occur when:

- A non-null record already exists. The Status Key is set to 22 (for duplicate record).
- The record number is outside the extent of the file. The Status Key is set to 24 (for boundary condition).

Example — Creating a Relative File

The program example in figure 5.7 creates a master file of employee records. Employee numbers from 1001 through 2001 are entered in the employee number field (EMPNO).

```
       IDENTIFICATION DIVISION.
       PROGRAM-ID. LOADIT.
       ENVIRONMENT DIVISION.
       CONFIGURATION SECTION.
       SOURCE-COMPUTER. IBM-370.
       OBJECT-COMPUTER. IBM-5280.
       SPECIAL-NAMES.
           CONSOLE IS SCREEN.
       INPUT-OUTPUT SECTION.
       FILE-CONTROL.
           SELECT EMPMAS-FILE ASSIGN TO DISK
              ORGANIZATION IS RELATIVE
              ACCESS MODE IS SEQUENTIAL
              RELATIVE KEY IS RKEY
              FILE STATUS IS STATUS-KEY.
       DATA DIVISION.
       FILE SECTION.
       FD  EMPMAS-FILE
           LABEL RECORDS ARE STANDARD.
           COPY DDS-EMPRECD.
000001 01      EMPRECD.
000002     02  ACREC    PIC X(00002).
000004     02  EMPNO    PIC S9(05)V.
000006     02  ENAME    PIC X(00020).
000008     02  STRAD    PIC X(00020).
000010     02  CTYST    PIC X(00020).
000012     02  ZIPCD    PIC X(00005).
000016     02  BEGDT    PIC X(00006).
000018     02  SOSNO    PIC X(00009).
000020     02  MARST    PIC X(00001).
       WORKING-STORAGE SECTION.
       77  STATUS-KEY PIC X(2).
       77  RKEY PIC 9(5) VALUE 1.
       77  DEMPNO PIC 9(5) VALUE 0.
       PROCEDURE DIVISION.
       BEGIN.
           OPEN OUTPUT EMPMAS-FILE.
               MOVE SPACES TO EMPRECD.
               MOVE "F" TO ACREC.
           DISPLAY
          "EMPLOYEE MASTER FILE BEING CREATED."
                 UPON SCREEN.
           PERFORM LOAD 1000 TIMES.
           DISPLAY
             "MASTER FILE CREATED. READY FOR UPDATE RECORDS"
                 UPON SCREEN.
       FIN.
           CLOSE EMPMAS-FILE.
           STOP RUN.
       LOAD.
           ADD RKEY 1000 GIVING DEMPNO.
           MOVE DEMPNO TO EMPNO.
           WRITE EMPRECD INVALID KEY
                 DISPLAY "  EMPLOYMENT NUMBER IS " EMPNO.
                 DISPLAY "  INVALID KEY IS " RKEY.
                 DISPLAY "  DEMPNO IS " DEMPNO.
                 DISPLAY "  STATUS KEY IS " STATUS-KEY.
                 GO TO FIN.
```

Figure 5.7. Creating a relative file

Reading from a Relative File

The following guidelines apply in reading records from a file with a relative organization:

1. Use the READ verb to access the records.
2. Specify INPUT or I-O mode in the corresponding OPEN. Otherwise, a logic error will occur, setting the Status Key to 92.
3. Records in a file with a relative organization can be read sequentially (ACCESS IS SEQUENTIAL is specified in the SELECT clause) or randomly (ACCESS IS RANDOM).

   For sequential access, reading a relative file is like reading a sequential file, except only successive, non-null records are returned. Moreover, if you specify RELATIVE KEY IS in the SELECT clause, the relative record number will be returned in the relative key data-item.

   For random access, you must place a relative record number in the key data-item before reading the record. An invalid key condition occurs if either a null record is found (the Status Key is set to 23), or the record is past the end-of-file or the relative key is 0, (the Status Key is set to 24).

Updating a Relative File

The following guidelines apply in updating records in a file with a relative organization:

1. The file that is to be updated must have previously been created.
2. Specify I-O mode in the corresponding OPEN. Otherwise, a logic error will occur on a subsequent READ, WRITE, or REWRITE, setting the Status Key to 92.
3. Records in a file with a relative organization can be read sequentially (ACCESS IS SEQUENTIAL is specified in the SELECT clause) or randomly (ACCESS IS RANDOM).

   For sequential access, updating is done with a REWRITE or DELETE, always preceded by a READ. If there is no preceding READ, or the preceding READ statement was unsuccessful, the REWRITE or DELETE will fail, setting the Status Key to 92.

   For random access, updating is done only with a WRITE to a null record, or with a REWRITE or DELETE to an existing record; no preceding READ is necessary. The following invalid key conditions can occur for random access:

   - No record is found (the Status Key is set to 23).
   - Duplicate relative record Key (the Status Key is set to 22). This occurs when a non-null record is found in a file at the location indicated by the Key in a WRITE.
   - Boundary violation (the Status Key is set to 24).

Example — Reading and Updating a Relative File

Figure 5.8 shows the required input/output statements needed to read and update a relative file.

```
                  •
                  •
                  •
    SELECT EMPMAS-FILE ASSIGN TO DISK
       ORGANIZATION IS RELATIVE
       ACCESS MODE IS RANDOM
       RELATIVE KEY IS RKEY
       FILE STATUS IS DISK-STAT.
DATA DIVISION.
FILE SECTION.
FD  EMPMAS-FILE
    LABEL RECORDS ARE STANDARD.
    COPY DDS-EMPRECD.
01      EMPRECD.
    02  ACREC     PIC X(00002).
    02  EMPNO     PIC S9(05)V.
            •
            •
            •
WORKING-STORAGE SECTION.
77  DISK-STAT            PIC XX VALUE SPACES.
77  RKEY                 PIC 9999 VALUE ZERO.
DATA DIVISION.
            •
            •
            •
    OPEN I-O EMPMAS-FILE-PR3.
            •
            •
            •
READ-MASTER.
    SUBTRACT 1000 FROM EMPNO GIVING RKEY.
    READ EMPMAS-FILE
       INVALID KEY
          MOVE 0 TO CONTINUE
          MOVE 1 TO ERROR-FOUND, INVALID-EMPNO.
    MOVE SCREEN-RECORD TO EMPRECD.
    REWRITE EMPRECD
       INVALID KEY
          MOVE 1 TO INVALID-EMPNO
          MOVE 0 TO CONTINUE.
            •
            •
            •
    CLOSE EMPMAS-FILE.
```

Figure 5.8. Example of processing files with a relative organization

(This page is intentionally left blank.)

Processing Files with an Indexed Organization

Figure 5.9 shows the COBOL statements which can be used for indexed file processing.

Division	Reading	Writing	Updating
Environment Division	SELECT ASSIGN ORGANIZATION IS INDEXED RECORD KEY FILE STATUS ACCESS IS SEQUENTIAL ACCESS IS RANDOM	SELECT ASSIGN ORGANIZATION IS INDEXED RECORD KEY FILE STATUS ACCESS IS SEQUENTIAL ACCESS IS RANDOM	SELECT ASSIGN ORGANIZATION IS INDEXED RECORD KEY FILE STATUS ACCESS IS SEQUENTIAL ACCESS IS RANDOM
Procedure Division	OPEN INPUT OPEN I-O READ CLOSE	OPEN OUTPUT WRITE CLOSE	ACCESS IS SEQUENTIAL OPEN I-O READ REWRITE DELETE CLOSE For ACCESS IS RANDOM OPEN I-O READ WRITE REWRITE DELETE CLOSE

Figure 5.9. Statements used with indexed file organizations

Two Types of Indexed Files

COBOL supports two types of indexed files. In one type, an entry for each record is stored in a separate data set called an index data set. The entry consists of the record's key and the record's location. Accessing this data set is similar to the *key indexed access method* described in the *5280 System Concepts* manual.

In the other type of indexed organization supported by COBOL, the records are kept in sequence in the file by the system each time you add or delete one. Accessing this type of organization with the random access method is the equivalent of using the *direct by key access method* described in the *5280 System Concepts* manual.

Effects of Sequential and Random Access

You can access the record in an indexed file either sequentially (ACCESS IS SEQUENTIAL) or randomly (ACCESS IS RANDOM).

With sequential access, the records are accessed in ascending sequence according to the value of the record keys.

You can expect better performance with the sequential access method when:

- The file is being created.
- The entire file is being updated, rather than just a few random records, and a separate index data set hasn't been specified.

FILE-CONTROL Paragraph for Indexed Organization

The following is a partial format of the FILE-CONTROL Paragraph:

```
SELECT file-name
ASSIGN TO DISK
["device-address1" ["[*volid1.]dsname1"]]
[INDEX "device-address2" ["[*volid2.]dsname2"]]]
[SIZE nnnn]
[SHARE | SHARER]
      •
      •
      •
```

☞**Note**: Other clauses in the FILE-CONTROL section include the ORGANIZATION, ACCESS MODE, RECORD KEY, RELATIVE KEY, and FILE STATUS clauses. They are described in the *Language Reference Manual*; some are shown in the examples given later in this chapter.

Device-address1, Volid1, and Dsname1

The data set specified by this parameter contains the application records.

device-address1 is the physical address (4000 or 4400, for example) or logical address of the device on which the diskette must be mounted.

volid1 is the identification given when the volume was initialized with the diskette initialization utility.

dsname1 is the name given to the data set when it was allocated, usually assigned with the diskette label maintenance utility.

The effect of the above specification on the prompt given when the program is loaded, and the operator response to the prompt, is explained in the section "FILE-CONTROL Paragraph" earlier in this chapter.

INDEX Clause

Specify the INDEX clause when you want two data sets: one for the application records and another for the index; *device-address2, volid2, and dsname2* provide the same information for the index data set as the *device-address1, volid1, and dsname1* provide for the data set containing the application records.

The data set specified by *device-address2, volid2, and dsname2* will contain the record keys and location, depending on how the data set was created as will be explained later.

SIZE Clause

The SIZE clause is optional. With the clause, you control the number of keys in main storage and the performance of your program. In *nnnn*, specify up to four digits to indicate the number of keys from the index data set that are to be kept in main storage during execution. Consider the following when specifying the SIZE clause:

1. The value *nnnn* must be greater than or equal to 2.
2. A default value of 5 is used when ACCESS IS RANDOM is coded. The in-storage index isn't used when ACCESS IS SEQUENTIAL is in effect.
3. By increasing the value *nnnn*, performance may be improved because of less search time to find the desired record. However, you must also consider the additional storage required when increasing the size of the in-storage index.
4. The SIZE clause can be specified with either type of indexed file.
5. The amount of storage needed for the in-storage index is (keysize + 3) x nnnn.

SHARE Clause

This clause is described earlier in this chapter under "Sharing Files".

Rules and Considerations for Index Data Sets

1. If you code a password in the FD statement of the application record data set, the system will assume that the password is also valid for the index data set.
2. Any SHARE options you specify for the application record data set will also apply to the index data set.
3. Both the index and application record data sets must be on single volumes.
4. Byte 0 of the four-byte location is ignored when the data set is being accessed. It is set to X'00' when COBOL creates the data set.
5. Bytes 1 through 3 contain the relative record number of a record in the application data set.

5. You can create an index data set in two formats:

   1. From a COBOL program by specifying OUTPUT mode in the corresponding OPEN. Each record in the index data set will contain both the record key and the corresponding four-byte location of a corresponding application record in the application record data set.

      Note: A COBOL program can access a data set created in this format by a DE/RPG program.

   2. With the ADDROUT function of the sort program or some other user-written program. Each record in the index data set will contain the location of a corresponding application record in the application record data set.

Other rules and considerations for index data sets depend on which of the above two formats were used to create them.

Index Data Sets Created by COBOL Programs

When an index data set is created by a COBOL program, the following applies:

1. When allocating the index data set, specify a record size equal to the length of the key plus 4. Otherwise, at execution, an error will occur, setting the file status key to 95.
2. The application records are placed in their associated data set in the order they are written.

   Note: When random access is used, the records may be written in any order. When sequential access is used, they must be written in ascending order by record key.
3. The index records are inserted into the index data set in the proper ascending sequence.
4. The keys in the index data set must be unique.

When Created by Other Programs

When an index data set contains only the four-byte locations of application data set records, consider the following:

1. The index data set contains only the four-byte locations of the application record; no key fields are within the index data set.
2. Specify only ACCESS IS SEQUENTIAL and the INPUT mode or the I-O mode in the corresponding OPEN.
3. COBOL assumes that the length of the records in the index data set is four (4) bytes; if the length isn't four bytes, the file status key will be set to 95 indicating an invalid file.

Adding Records to Indexed Files

You cannot add records to files with indexed organizations if those files are being shared with other programs. An attempt to add a record to a shared indexed file causes a error condition and the Status Key to be set to 92. (See "Sharing Files" earlier in this chapter for more information.)

Creating an Indexed File

The rules and recommendations for creating a file with an indexed organization are:

1. Use the sequential access method (rather than the random) for better performance.
2. Specify OUTPUT mode in the corresponding OPEN.
3. When creating an indexed file with sequential access the records must be written in ascending key sequence. If a record is not in ascending key sequence, an invalid key condition will occur, setting the Status Key to 21 (for sequence error).
4. You must place a value in the record key data-item before issuing a WRITE.
5. If the end-of-extent is reached on a WRITE request, an invalid key condition occurs setting the Status Key to 24 (for boundary error).

See "Allocating Data Sets for Program Files" in Chapter 8 for the rules that apply when allocating data sets for your program using IBM 5280 facilities.

Example

The examples in figure 5.10 and figure 5.11 create files of employee master records. Employee numbers from 5000 to 5999 are entered in the employee number field (EMPNO). An index organization with one data set is shown in figure 5.10; the example in figure 5.11 shows an index organization with one data set for application records and another for record keys. The only coding difference is the addition of an INDEX clause in the File Control Paragraph of the example shown in figure 5.11.

```
IDENTIFICATION DIVISION.
PROGRAM-ID. LOAD-EMPMAS.
ENVIRONMENT DIVISION.
CONFIGURATION SECTION.
SOURCE-COMPUTER. IBM-370.
OBJECT-COMPUTER. IBM-5280.
SPECIAL-NAMES.
    CONSOLE IS SCREEN.
INPUT-OUTPUT SECTION.
FILE-CONTROL.
    SELECT EMPMAS-FILE ASSIGN TO DISK
       ORGANIZATION INDEXED
       RECORD KEY IS EMPNO
       ACCESS SEQUENTIAL
       FILE STATUS IS STATUS-KEY.
DATA DIVISION.
FILE SECTION.
FD  EMPMAS-FILE
    LABEL RECORDS ARE STANDARD.
    COPY DDS-EMPRECD.
WORKING-STORAGE SECTION.
77  STATUS-KEY PIC X(2).
77  XEMPNO PIC 9(5).
PROCEDURE DIVISION.
BEGIN.
    OPEN OUTPUT EMPMAS-FILE.
        MOVE SPACES TO EMPRECD.
        MOVE "F" TO ACREC.
    DISPLAY
   "EMPLOYEE MASTER FILE WITH 1000 MASTER RECORDS BEING CREATED"
          UPON SCREEN.
    PERFORM LOAD VARYING XEMPNO
    FROM 5000 BY 1 UNTIL XEMPNO IS EQUAL TO 6000.
    DISPLAY
      "MASTER FILE CREATED. READY FOR UPDATE RECORDS"
          UPON SCREEN.
FIN.
    CLOSE EMPMAS-FILE.
    STOP RUN.
LOAD.
    MOVE XEMPNO TO EMPNO.
    WRITE EMPRECD INVALID KEY
          DISPLAY "  INVALID KEY IS " EMPNO
          DISPLAY "  STATUS KEY IS " STATUS-KEY
          GO TO FIN.
```

Figure 5.10. Creating an indexed file — without index data set

```
IDENTIFICATION DIVISION.
PROGRAM-ID. LOAD-EMPMAS.
ENVIRONMENT DIVISION.
CONFIGURATION SECTION.
SOURCE-COMPUTER. IBM-370.
OBJECT-COMPUTER. IBM-5280.
SPECIAL-NAMES.
    CONSOLE IS SCREEN.
INPUT-OUTPUT SECTION.
FILE-CONTROL.
    SELECT EMPMAS-FILE
       ASSIGN TO DISK
       INDEX
       ORGANIZATION IS INDEXED
       ACCESS MODE IS SEQUENTIAL
       RECORD KEY IS EMPNO
       FILE STATUS IS STATUS-KEY.
DATA DIVISION.
FILE SECTION.
FD  EMPMAS-FILE
    LABEL RECORDS ARE STANDARD.
    COPY DDS-EMPRECD.
01      EMPRECD.
    02  ACREC    PIC X(00002).
    02  EMPNO    PIC X(00005).
    02  ENAME    PIC X(00020).
    02  STRAD    PIC X(00020).
    02  CTYST    PIC X(00020).
    02  ZIPCD    PIC X(00005).
    02  BEGDT    PIC X(00006).
    02  SOSNO    PIC X(00009).
    02  MARST    PIC X(00001).
WORKING-STORAGE SECTION.
77  STATUS-KEY PIC X(2).
77  XEMPNO PIC 9(5).
PROCEDURE DIVISION.
BEGIN.
    OPEN OUTPUT EMPMAS-FILE.
        MOVE SPACES TO EMPRECD.
        MOVE 'F' TO ACREC.
    DISPLAY
   'EMPLOYEE MASTER FILE WITH 1000 MASTER RECORDS BEING
          UPON SCREEN.
    PERFORM LOAD VARYING XEMPNO
    FROM 5000 BY 1 UNTIL XEMPNO IS EQUAL TO 6000.
    DISPLAY
      'MASTER FILE CREATED. READY FOR UPDATE RECORDS'
          UPON SCREEN.
FIN.
    CLOSE EMPMAS-FILE.
    STOP RUN.
LOAD.
    MOVE XEMPNO TO EMPNO.
    WRITE EMPRECD INVALID KEY
          DISPLAY '    INVALID KEY IS ' EMPNO
          DISPLAY '    STATUS KEY IS ' STATUS-KEY
          GO TO FIN.
```

Figure 5.11. Creating an indexed file — with index data set

Reading an Indexed File

The following guidelines apply in reading records from a file with an indexed organization:

1. Use the READ verb to access the records.
2. Specify INPUT or I-O mode in the corresponding OPEN. Otherwise, a logic error will occur, setting the Status Key to 92.
3. Records in a file with an indexed organization can be read sequentially (ACCESS IS SEQUENTIAL is specified in the SELECT clause) or randomly (ACCESS IS RANDOM).

   For sequential access, reading an indexed file is like reading a sequential file: the records are read in ascending order by the sequence of the keys.

   For random access, you must place a value in the record key data-item before reading the record. An invalid key condition occurs if no record with the specified key is found (the Status Key is set to 23).

Updating an Indexed File

The rules and recommendations for updating a file with an indexed organization are:

1. You can use either the sequential access method or the random access method. When updating the entire file, or most of the file, use the sequential access method for the best performance. When updating only a few records in the entire file, use the random access method.
2. For sequential access, updating is done by a REWRITE or DELETE, always preceded by a READ.
3. For random access, updating is done with a WRITE to a null record, or with a REWRITE or DELETE to an existing record; no preceding READ is necessary.
4. Errors which may be received include: no space for insert (Status Key 90) or two invalid key conditions: duplicate record (the Status Key is set to 22) or no record found (the Status Key is set to 23).

The example in figure 5.12 shows the updating of selected records in an indexed file. The input records contain the key for the record, the depositor name, and the amount of the transaction. Random access is used to update the transaction records.

```
IDENTIFICATION DIVISION.
PROGRAM-ID.  UPDATE-INDEXED.
ENVIRONMENT DIVISION.
CONFIGURATION SECTION.
SOURCE-COMPUTER.  IBM-370.
OBJECT-COMPUTER.  IBM-5280.
SPECIAL-NAMES.  CONSOLE IS TYPEWRITER.
INPUT-OUTPUT SECTION.
FILE-CONTROL.
    SELECT INDEXED-FILE
      ASSIGN TO DISK
      ORGANIZATION INDEXED
      ACCESS RANDOM
      RECORD KEY IS REC-ID FILE STATUS IS SK.
    SELECT IN-FILE
      ASSIGN TO DISK.
    SELECT PRINT-FILE
      ASSIGN TO PRINTER.
```

Figure 5.12. Updating an indexed file (Part 1 of 3)

```
DATA DIVISION.
FILE SECTION.
FD  INDEXED-FILE LABEL RECORDS STANDARD.
01  DISK-RECORD.
    05  REC-ID.
        10  REC-GEN-FLD PICTURE   X(5).
        10  REC-DET-FLD PICTURE   X(5).
    05  DISK-FLD1   PICTURE   X(10).
    05  DISK-NAME   PICTURE   X(20).
    05  DISK-BAL    PICTURE   S9(6)V99.
FD    IN-FILE    LABEL RECORD STANDARD.
01    IN-REC.
    05    IN-ID.
        10    IN-GEN-FLD    PICTURE X(5).
        10    IN-DET-FLD    PICTURE X(5).
    05    IN-NAME            PICTURE X(20).
    05    IN-AMT            PICTURE S9(6)V99.
FD  PRINT-FILE   LABEL RECORD OMITTED.
01  PRINT-RECORD-1.
    05  PRINT-ID       PICTURE X(10).
    05  FILLER         PICTURE X(5).
    05  PRINT-NAME     PICTURE X(20).
    05  FILLER         PICTURE X(5).
    05  PRINT-BAL      PICTURE $$$$,$$$.99-.
    05  FILLER         PICTURE X(5).
    05  PRINT-AMT      PICTURE $$$$,$$$.99-.
    05  FILLER         PICTURE X(5).
    05  PRINT-NEW-BAL  PICTURE $$$$,$$$.99-.
01  PRINT-RECORD-2     PICTURE X(86).
WORKING-STORAGE SECTION.
77  GO-TO-SWITCH       PICTURE 9 VALUE 1.
77  LINE-COUNT         PICTURE 99 COMPUTATIONAL.
01  PAGE-HEAD.
    05  FILLER         PICTURE X(36)  VALUE SPACES.
    05  FILLER         PICTURE X(14)  VALUE "UPDATE REPORT".
    05  FILLER         PICTURE X(36)  VALUE SPACES.
01  PAGE-FOOT.
    05  FILLER         PICTURE X(78)  VALUE SPACES.
    05  FILLER         PICTURE A(6)   VALUE "PAGE".
01  ERROR-MESSAGE.
    05  OP-NAME        PICTURE X(7).
    05  FILLER         PICTURE XX  VALUE SPACES.
    05  SK             PICTURE XX  VALUE "ZZ".
    05  PG-NUMBER      PICTURE 99 VALUE 00.
```

Figure 5.12. Updating an indexed file (Part 2 of 3)

```
PROCEDURE DIVISION.
BEGIN-PROCESSING.
    OPEN INPUT IN-FILE
        I-O INDEXED-FILE
        OUTPUT PRINT-FILE.
    IF SK NOT = "00" MOVE "OPEN" TO OP-NAME
        PERFORM ERROR-ROUTINE-1 THRU ERROR-ROUTINE-2
        GO TO END-JOB-2.
    PERFORM PAGE-START.
READ-INPUT.
    READ IN-FILE AT END GO TO END-JOB-1.
RANDOM-PROCESS-1.
    MOVE IN-ID TO REC-ID. MOVE SPACES TO PRINT-RECORD-1.
    READ INDEXED-FILE INVALID KEY GO TO END-JOB-1.
    IF SK NOT  = "00" MOVE "READ-D" TO OP-NAME
        PERFORM ERROR-ROUTINE-1 THRU ERROR-ROUTINE-2
        GO TO READ-INPUT.
    MOVE DISK-NAME TO PRINT-NAME.
    MOVE DISK-BAL TO PRINT-BAL.
    MOVE IN-AMT TO PRINT-AMT.
    ADD IN-AMT TO DISK-BAL.
    MOVE DISK-BAL TO PRINT-NEW-BAL.
    PERFORM WRITE-PARA-1 THRU WRITE-PARA-2.
RANDOM-PROCESS-2.
    GO TO READ-INPUT.
WRITE-PARA-1.
    IF LINE-COUNT = 60
        PERFORM PAGE-END THROUGH PAGE-START.
    WRITE PRINT-RECORD-1.
    ADD 1 TO LINE-COUNT.
    REWRITE DISK-RECORD INVALID KEY GO TO CHK-ERR.
CHK-ERR.
    IF SK NOT = "00" MOVE "REWRITE" TO OP-NAME
        PERFORM ERROR-ROUTINE-1 THRU ERROR-ROUTINE-2.
WRITE-PARA-2.
    EXIT.
PAGE-END.
    ADD 1 TO PG-NUMBER.
    WRITE PRINT-RECORD-2 FROM PAGE-FOOT
        AFTER ADVANCING 3.
PAGE-START.
    WRITE PRINT-RECORD-2 FROM PAGE-HEAD
        AFTER ADVANCING PAGE.
    MOVE 1 TO LINE-COUNT.
ERROR-ROUTINE-1.
    DISPLAY ERROR-MESSAGE UPON TYPEWRITER.
ERROR-ROUTINE-2.
    EXIT.
END-JOB-1.
    IF SK NOT =  "00" MOVE "READ" TO OP-NAME
        PERFORM ERROR-ROUTINE-1 THRU ERROR-ROUTINE-2.
    CLOSE INDEXED-FILE.
    IF SK NOT =  "00" MOVE "CLOSE" TO OP-NAME
        PERFORM ERROR-ROUTINE-1 THRU ERROR-ROUTINE-2.
END-JOB-2.
    CLOSE IN-FILE  PRINT-FILE.
    STOP RUN.
```

Figure 5.12. Updating an indexed file (Part 3 of 3)

Chapter 6. Other I/O

This information supplements that in the *5280 COBOL Language Reference* for the following types of I/O:

- Printer I/O
- Writing data to a work station screen using SEQUENTIAL I/O
- STOP 'literal'. The STOP statement makes it possible to write data to the status line (line 1) of the work station screen.
- DISPLAY and ACCEPT. These two COBOL verbs make possible the interchange of low volume data between a program and the printer or work station screen.

Printer I/O

Here are guidelines and rules for specifying I/O statements that are to transmit data to 5280 printers:

1. The format of the FILE-CONTROL paragraph is as follows:

```
 SELECT file-name ASSIGN TO PRINTER ["device address"]
[ORGANIZATION IS SEQUENTIAL]
[ACCESS MODE IS SEQUENTIAL]
[FILE STATUS IS data-name]
```

2. The WRITE ADVANCING facility is supported.
3. The maximum number of characters that can be written on one line is 198.
4. The printer cannot be shared with another program. It will be available to other programs after a CLOSE has been executed.

Example of Printer I/O Statements

Figure 6.1 gives an example of the input/output statements required to write data to a printer.

```
IDENTIFICATION DIVISION.
PROGRAM-ID.  SEND-RECEIVE.
          •
          •
          •
INPUT-OUTPUT SECTION.
FILE-CONTROL.
    SELECT PRINT-FILE ASSIGN TO PRINTER
      ORGANIZATION IS SEQUENTIAL
      ACCESS IS SEQUENTIAL
      FILE STATUS IS PRINT-FILE-STAT.
          •
          •
          •
FD  PRINT-FILE
    LABEL RECORDS ARE STANDARD.
01  PRINT-RECORD.
    05  RCDCD PIC X(87).
WORKING-STORAGE SECTION.
77  PRINT-FILE-STAT       PIC XX VALUE SPACES.
01  EMPRECD.
    02  ACREC    PIC X(00002).
    02  EMPNO    PIC S9(05)V.
    02  ENAME    PIC X(00020).
          •
          •
          •
PROCEDURE DIVISION.
DECLARATIVES.
PRINT-ERROR SECTION.
   USE AFTER ERROR PROCEDURE ON PRINT-FILE.
PRINTERX.
        DISPLAY "ERROR ON PRINTER-FILE I/O DURING WRITE".
        DISPLAY "FILE STATUS IS " PRINT-FILE-STAT.
        DISPLAY "TRANSMISSION STOPPED. RUN STOPPED.".
        STOP RUN.
END DECLARATIVES.
          •
          •
          •
    OPEN OUTPUT PRINT-FILE.
          •
          •
          •
    WRITE PRINT-RECORD FROM EMPRECD.
          •
          •
          •
    CLOSE PRINT-FILE.
```

Figure 6.1. Example of COBOL statements used for printer I/O

DISPLAY and ACCEPT

Here are some rules and guidelines when using DISPLAY and ACCEPT statements:

1. If the screen is clear, the first character string transmitted by a DISPLAY statement appears in line 3. If data is already displayed on the screen, the character string will appear after the last line displayed.

When the last line on the screen is filled, the next character string will be displayed in line 3; succeeding character strings will follow until the last line is filled, and wrap around again.

2. When you use DISPLAY, the maximum number of characters that will appear on one line of the work station display is 78. If a character string is greater than 78 characters, the remaining characters will be sent to succeeding lines in 78-character increments until the entire string is displayed.
3. You can use DISPLAY and ACCEPT for the printer and the work station screen. DISPLAY transmits character strings to the printer by default unless you specifically specify the screen.
4. If you specify the printer as the I/O device in your program either implicitly or explicitly, the operator can re-direct the DISPLAY transmission to the work station screen when the COBOL program is loaded.
5. If you specify the work station screen, the operator cannot re-direct transmission to the printer.
6. ACCEPT can process only 78 characters of data. Any data over 78 characters will be truncated.
7. The printer cannot be shared with another program while a DISPLAY to the printer is being executed.

Example of DISPLAY

Figure 6.2 gives an example of the statements needed to display data on a work station screen.

```
IDENTIFICATION DIVISION.
PROGRAM-ID.  SEND-RECEIVE.
AUTHOR.  J BELL.
ENVIRONMENT DIVISION.
CONFIGURATION SECTION.
SOURCE-COMPUTER. IBM-370 WITH DEBUGGING MODE.
OBJECT-COMPUTER. IBM-5280.
SPECIAL-NAMES.
    CONSOLE IS SCREEN.
                 •
                 •
                 •
PROCEDURE DIVISION.
                 •
                 •
                 •
   IF COMM-ERROR = 0
    DISPLAY
     "RECORDS SENT SUCCESSFULLY; NOW AWAITING RESPONSE"
        UPON SCREEN.
                 •
                 •
                 •
```

Figure 6.2. Example of statements used for DISPLAY

SEQUENTIAL Work Station I/O

Here are the rules for input/output between a work station and a COBOL program:

1. The entries in the FILE-CONTROL paragraph are:

   ```
   SELECT file-name ASSIGN TO WORKSTATION [n]
    ORGANIZATION IS SEQUENTIAL
   [FILE STATUS IS data-name-3]
   [ACCESS MODE IS SEQUENTIAL]
   ```

   WORKSTATION indicates this is a work station for a sequential file.

   n is an integer that specifies one of the three sizes of work station screens as follows:

   480 for the 480-character screen.
   960 for the 960-character screen.
   1920 for 1920-character screen.

   If you don't specify *n*, a screen size of 1920 characters is assumed.
2. The maximum record size that can be written is 78 bytes. If the record defined in your program is greater than 78 bytes, the excess characters will be truncated. If the record is less than 78 bytes, the right-most bytes will be padded with blanks.
3. Specify OUTPUT mode in the OPEN statement for the file defined for the work station.
4. If a sequential file and a transaction file are both assigned to the work station, only one of them can be open at a time. However, multiple *sequential* files assigned to the work station can be open simultaneously.
5. ACCEPT and DISPLAY statements can be issued while a SEQUENTIAL I/O file for the work station is opened.
6. The ADVANCING mnemonic-name phrase is not supported for the WRITE statement.
7. The AT END imperative phrase will be ignored. It will not be executed because an AT END condition cannot occur on a work station.
8. If the screen is clear, the first record written to the screen will appear in line 3. Otherwise, the first record will appear after the last line already on the screen.
9. When the last line of the screen has been used, the next record will be written to line 3.

Work Station I/O with Sequential I/O — Example

Figure 6.3 shows an example of work station I/O using the sequential access method and data set organization. The interaction between the program and the operator is as follows:

1. The program issues a prompt for the operator to enter data.
2. The operator enters data and presses the Enter key.
3. The program writes the data back to the display screen.
4. The above steps are repeated until the operator enters *EOF* to indicate end of data.

```
IDENTIFICATION DIVISION.
PROGRAM-ID.     ECHO.
ENVIRONMENT DIVISION.
CONFIGURATION SECTION.
SOURCE-COMPUTER.  IBM-370.
OBJECT-COMPUTER.  IBM-5280.
INPUT-OUTPUT SECTION.
FILE-CONTROL.
    SELECT IN-FILE
      ASSIGN TO WORKSTATION 1920.
    SELECT PRINT-FILE
      ASSIGN TO WORKSTATION 1920.
    SELECT CLEAR-FILE
      ASSIGN WORKSTATION 1920.
DATA DIVISION.
FILE SECTION.
FD  CLEAR-FILE LABEL RECORDS STANDARD
    BLOCK CONTAINS 80.
01  CLEAR-REC.
    05 CLEAR-1       PICTURE X(10).
    05 CLEAR-2       PICTURE X(70).
FD  IN-FILE LABEL RECORDS STANDARD
    BLOCK CONTAINS 80.
01  IN-REC.
    05 FIRST-REC   PICTURE X(10).
    05 SECOND-REC  PICTURE X(70).
FD  PRINT-FILE LABEL RECORDS STANDARD
    BLOCK CONTAINS 80.
01  OUT-REC.
    05  REC-ONE      PICTURE X(10).
    05  REC-TWO      PICTURE X(70).
PROCEDURE DIVISION.
DECLARATIVES.
ERROR-HANDLING SECTION.
    USE AFTER STANDARD ERROR PROCEDURE ON IN-FILE.
ERROR-ROUTINE.
    DISPLAY "I/O ERROR ON IN-FILE".
    CLOSE IN-FILE.
    STOP RUN.
END DECLARATIVES.
```

Figure 6.3. Example of statements used for sequential I/O to a work station (Part 1 of 2)

```
PGM1 SECTION.
BEGIN-PROGRAM.
    DISPLAY "START PROGRAM ECHO".
FILE-OPENING.
    OPEN INPUT IN-FILE.
FILE-OPEN-2.
    OPEN OUTPUT CLEAR-FILE PRINT-FILE.
CLEAR-SCREEN.
    PERFORM CLEAR-WRITE.
    GO TO PROCESS-ECHO.
CLEAR-WRITE.
    MOVE SPACES TO CLEAR-REC.
    WRITE CLEAR-REC.
PROCESS-ECHO.
    MOVE SPACE TO REC-ONE.
    MOVE "TO ENTER, HIT ENTER; TO END, TYPE 'EOF'." TO
    REC-TWO.
    WRITE OUT-REC.
ECHO-1.
    READ IN-FILE INTO REC-ONE.
    IF FIRST-REC = "EOF" GO TO WRAP-IT-UP.
    MOVE SECOND-REC TO REC-TWO.
    WRITE OUT-REC.
    GO TO ECHO-1.
WRAP-IT-UP.
    CLOSE IN-FILE.
    PERFORM CLEAR-WRITE.
    CLOSE PRINT-FILE.
END-IT.
    DISPLAY "END PROGRAM ECHO".
    STOP RUN.
```

Figure 6.3. Example of statements used for sequential I/O to a work station (Part 2 of 2)

STOP Statement

5280 COBOL supports the STOP "literal" statement. When issued, the character string will appear in line 1 (the status line) of the work station screen. The operator can respond by either pressing the Enter key, or the CMD key followed by the End-of-Job key.

If the Enter key is pressed, the COBOL program resumes execution. If the End-of-Job key is pressed, all files opened by the COBOL program are closed and the job ends.

Chapter 7. Compiler Job Procedures and Options

This chapter provides the following information:

- A description of the job statements and the data sets required on the host system for compilation. Separate sections are included for each host compiler.
- How to compile more than one COBOL source module in the same job, described in the section "Multiple Compilations – The *PROCESS Statement".
- A description of the compiler options you can select for each compilation in the section "Host Compiler Options".

Inter-Program Communications and Multiple-Compilations

The inter-program communications facility provides a facility by which one program can communicate with one or more other programs using CALL statements. It is described in detail in the *5280 COBOL Language Reference Manual.*

☞**Note**: *IBM 5280 COBOL requires that you compile all called and calling programs using inter-program communications in the same job step using the *PROCESS statement.*

A linkage editor is not used for IBM 5280 COBOL. The host compiler produces object modules ready for execution on the 5280. A linkage editing step between compilation and execution is neither necessary nor possible; therefore, compilation of all called and calling modules in the same job step is required.

A Note on Compiler Options

The COBOL compiler provides a number of options that control compilation. For example, the options determine the content of your output listing, the types of messages that will be issued, how your statements are numbered in the output listing, etc. You can accept the options provided by default with the compiler. Or you can change the options to suit your needs when you compile a program.

If you are compiling a program for the first time, you may want to review the default options provided by IBM and the other available options. The options are described in the last section of this chapter.

You may want to change the IBM-supplied options. You can do this as follows:

- For OS/VS, in the PARM field of the EXEC statement or in the *PROCESS statement.
- For DOS/VSE, in the *PROCESS statement.

(This page is intentionally left blank.)

OS/VS Compilation

Figure 7.1 shows an in-stream OS/VS job control procedure that can be used to compile COBOL source programs; an explanation of each statement follows the figure.

```
  //procname PROC
1 //COB EXEC PGM=AVC0
2 //STEPLIB DD DSN=dsname,DISP=SHR
3 //SYSLIB DD DSN=dsname,DISP=SHR        COPY LIBRARY
4 //SYSLDOUT DD DUMMY                    LOAD OUTPUT
5 //SYSLDOVL DD DUMMY                    OVERLAY OUTP
6 //SYSPRINT DD SYSOUT=A,DCB=BLKSIZE=1210 LISTING OUTP
  //SYSUDUMP DD SYSOUT=A
7 //SYSUT1 DD UNIT=SYSDA,SPACE=(512,(400,400))
7 //SYSUT2 DD UNIT=(SYSDA,SEP=SYSUT1),
  //             SPACE=(512,(400,400))
7 //SYSUT3 DD UNIT=SYSDA,SPACE=(512,(400,400))
7 //SYSUT4 DD UNIT=SYSDA,SPACE=(512,(400,400))
7 //SYSUT5 DD UNIT=SYSDA,SPACE=(512,(400,400))
  //          PEND
```

Figure 7.1 Job Control Procedure for the 5280 COBOL OS/VS Compiler

You must also specify a //SYSIN DD statement defining the data set that contains the COBOL source program to be compiled. (You can have the source statements instream following a //SYSIN DD * statement if desired.)

The following text describes the statements in figure 7.1.

1 COB is the step name of the compilation job step.

2 STEPLIB defines a step library where the compiler can be located. The use of a step library and the *dsname* are determined by your installation. STEPLIB must be allocated as a partitioned data set.

3 SYSLIB defines the COPY library, which contains source statements and data definitions statements (DDS) you can insert into a source program using COPY. This statement isn't required when compiling programs that don't use COPY. See figure 7.2 for the required parameters if you allocate this data set.

4 SYSLDOUT defines the data set where the compiler places the executable object module for transfer to the IBM 5280. If you specify the compiler options NODECK and NOOBJ (no object module to be built), this statement isn't needed. See figure 7.2 for the required parameters you must use to allocate this data set.

5 SYSLDOVL defines the data set where the compiler places the overlay modules when the program segmentation facility is used. Otherwise, this statement isn't required. See figure 7.2 for the required parameters you must use to allocate this data set.

☞**Note**: *If segmentation is used, don't define the primary program (defined by SYSLDOUT) and the overlay modules (defined by SYSLDOVL) as members of the same partitioned data set. Otherwise, the compiler will stop processing.*

6 SYSPRINT defines where the source statements, maps, and the other information you request through the various compiler options is to be printed. See figure 7.2 for the required parameters you must use to allocate this data set.

7 SYSUT1, SYSUT2, SYSUT3, SYSUT4, and SYSUT5 are utility data sets used by the compiler during execution.

ddname	RECFM	LRECL*
SYSLIB	F or FB	80
SYSLDOUT	F or FB	80 or 128
SYSLDOVL	F or FB	80 or 128
SYSPRINT	FA or FBA	121
SYSIN	F or FB	80
* Records can be blocked. The size (BLKSIZE) can be any multiple of the fixed-length records as determined by the amount of main storage available for buffers. The record length of SYSLDOUT and SYSLDOVL is determined by the DECK option.		

Figure 7.2. Required parameter for allocating COBOL data sets

☞ See Appendix E for estimates on the main and secondary storage required for the compiler and the various data sets.

Figure 7.3 shows an example of some of the statements that can be used when using the COBOL compile procedure. It doesn't show the required JOB statement, which you would code according to the rules set by your installation.

```
1 //CBJOB EXEC AVCOBOL,
  //         PARM.COB='MAP,XREF'
2 //COB.SYSLIB DD DSN=COPY,DISP=SHR
3 //COB.SYSLDOUT DD DSN=LOADLIB(PROGA),DISP=OLD
4 //COB.SYSIN DD DSN=SORCELIB(PROGA),DISP=SHR
  //*
```

Figure 7.3 Example of using a COBOL compile procedure

1 CBJOB causes the compile procedure (named AVCOBOL in this example) to be executed. The PARM statement causes the MAP and XREF options of the compiler to be executed. MAP produces a map of data areas and XREF a cross-reference of statements in the source program. Both of these options are explained later in this chapter.

2 COB.SYSLIB defines a partitioned data set for the COPY library, where the programmer has placed the data definition statements and other COBOL source statements as library members.

3 COB.SYSLDOUT defines a member of a partitioned data set — LINKLIB(PROGA) — for the object module. After execution, the object module can be found in the member PROGA of the partitioned data set LINKLIB.

4 COB.SYSIN defines a member of partitioned data set SORCELIB(PROGA) where the programmer has placed the COBOL source program that is to be compiled.

DOS/VSE Compilation

Figure 7.4 shows an example of the DOS/VSE job statements needed to compile a COBOL source program; an explanation of each statement follows the figure.

```
  // JOB COBOL TEST
  // ASSGN SYS001,150
  // ASSGN SYS002,150
  // ASSGN SYS003,150
  // ASSGN SYS004,150
  // ASSGN SYS005,150
  // ASSGN SYS006,150
  // ASSGN SYS007,150
1 // DLBL WORK1D,'CBL.WORK1',0
  // EXTENT SYS001,DOSR35,1,0,9063,19
1 // DLBL WORK2D,'CBL.WORK2',0
  // EXTENT SYS002,DOSR35,1,0,9082,19
1 // DLBL WORK3D,'CBL.WORK3',0
  // EXTENT SYS003,DOSR35,1,0,9101,19
1 // DLBL WORK4D,'CBL.WORK4',0
  // EXTENT SYS004,DOSR35,1,0,9120,19
1 // DLBL WORK5D,'CBL.WORK5',0
  // EXTENT SYS005,DOSR35,1,0,9139,19
2 // DLBL LDOUT,'CBL.LDOUT'
  // EXTENT SYS006,DOSR35,1,0,9747,19
3 // DLBL LDOVLY,'CBL.LDOVLY'
  // EXTENT SYS007,DOSR35,1,0,9766,19
  // EXEC AYCDO
  *PROCESS
     COBOL SOURCE STATEMENTS
  /*
  /&
```

Figure 7.4. Sample job control statements for a DOS/VSE compilation

The following text describes the statements in figure 7.4.

1 SYS001 through SYS005 are utility data sets used by the compiler during execution. All work files must be on the same device type. The device type can be a 2314, 3330, 3340, 3350, or an FBA device.

2 SYS006 defines the data set where the compiler places the executable load module for transfer to the IBM 5280. If you specify the compiler options NODECK and NOOBJ (no object module to be built), this statement isn't needed. The device type can be a 2314, 3330, 3340, 3350, 3540 (diskette), or an FBA device.

3 SYS007 defines the data set where the compiler places the overlay modules when the program segmentation facility is used. Otherwise, this statement isn't required. The device type can be a 2314, 3330, 3340, 3350, 3540 (diskette), or an FBA device.

☞ Note that DOS does not permit two files to be written to a diskette at the same time. For segmented programs only one of LDOUT and LDOVLY can be written directly to diskette; the other must be written to DASD and then copied to the diskette.

If COPY statements are used in the source program, a COPY library must be defined. THe COPY library will contain the source statements or data definition statements (DDS) to be copied into the program. The COPY library can be the system source statement library, or a private source statement library. The sublibrary is assumed to be *C*.

SYSLIST is assumed to have been assigned to a printer, where source statements, maps, and the other information you request through the various compiler options are to be printed.

Compiler Storage Requirements

The compiler requires a 144K partition with an additional 24K bytes of GETVIS space.

See Appendix F for estimates on the main and secondary storage required for the compiler and the various data sets.

☞**Note: With VSE/Advanced Functions release 2, the default value assigned for GETVIS space is 48K bytes. Because the operating system uses most of this space, you should specify the SIZE parameter on the EXEC statement so that at least 72K bytes of GETVIS space is allocated. For example:**

```
EXEC AVCDO,SIZE=144K
```

The above example assumes that the partition size is 216K bytes. 144K bytes are allocated for compiler space; the remaining 72K bytes are available for GETVIS space.

Multiple Compilations — The *PROCESS Statement

The *PROCESS statement allows you to compile more than one COBOL source program in the same job; *PROCESS statements are required when you use CALL statements in your source program to call other COBOL programs; the called programs must be compiled with the calling program, as shown in the following example:

```
*PROCESS LIST,XREF.
       •
       •
  (first COBOL source module)
       •
       •
*PROCESS.
       •
       •
  (second COBOL source module)
       •
       •
*PROCESS LIST,XREF.
       •
       •
  (third COBOL source module)
       •
       •
```

You can specify a compiler option as a parameter in the *PROCESS statement. Thus, you can vary the options for each of the compilations started by a *PROCESS statement. An option specified in a *PROCESS statement will override the defaults set by IBM and the options, if any, you specify in the job control statements of the host system.

When Using Subprogram Linkage

When using subprogram linkage, place the program which is to receive control initially as the first program of the programs to be compiled.

With Segmented Programs

Only one program (the first program) can be segmented when compiling multiple programs using the *PROCESS statement.

Maximum Number of Programs

A maximum of 10 programs can be compiled in the same job using *PROCESS statements.

*Format and Rules – *PROCESS*

The format of the *PROCESS statement is:

*PROCESS *options-list.*

options-list is one or more of the keywords listed in the compiler options, abbreviations and defaults table in figure 7.5.

The format rules and restrictions for the *PROCESS statement are:

1. The asterisk (*) must appear in column 1 of the input record.
2. The keyword PROCESS follows the asterisk, with no intervening blanks.
3. The options-list follows the keyword *PROCESS, with one or more intervening blanks.
4. The option keywords in the list are separated from each other by a comma, or one or more blanks, or both. The option keywords can appear in any order.
5. The statement should be followed by a period.
6. The *PROCESS statement, with or without options, causes a separate compilation for the source module following the statement. If no options are specified, the IBM defaults or the options specified in the PARM field of the EXEC statement apply for the compilation.
7. Multiple *PROCESS statements are permitted for any compilation, that is, each option keyword could be on its own *PROCESS statement.
8. If contradictory options are submitted (such as LIST and NOLIST) the last one specified is effective.

Host Compiler Options

This section describes the compiler options which you specify before a compilation and which deal with input to the compiler, the compilation activity, and the listings and object modules produced by the compiler.

These options apply to all the host IBM 5280 COBOL compilers.

Defaults and Overrides

You can indicate which options you want in effect during a compilation in three ways:

1. By accepting the default options and making no specifications at all. A list of the compiler options and their defaults is shown in figure 7.5; a detailed explanation of each option is given in the sections that follow.
2. For OS/VS, by specifying the option in the PARM statement of the EXEC statement (as described earlier in this chapter); the PARM specifications override the default options provided by IBM.
3. By specifying the option in the *PROCESS statement (described earlier in this Chapter under "Multiple Compilation — the *PROCESS Statement"); the *PROCESS specifications override both the defaults provided by IBM and, for OS/VS, options coded in the PARM field. By specifying options in the *PROCESS statements, you can vary the options for a series of source modules being compiled in the same job.

List of Options

Figure 7.5 shows the COBOL compiler options, their abbreviations, and their defaults.

Compiler Option	Abbreviation	Compiler Default
CMPAT(SIGNC | SIGNF)	CM(SIGNC | SIGNF)	CMPAT(SIGNC)
DECK | NODECK	D | NOD	NODECK
FIPS(L | LI) | NOFIPS		NOFIPS
FLAG(I | W | E | S | U) | NOFLAG	F(I | W | E | S | U) | NOF	FLAG(I)
FLUSHERR		(not performed)
GONUMBER | NOGONUMBER	GN | NOGN	NOGONUMBER
LINECOUNT(n)	LC(n)	LINECOUNT(56)
LIST | NOLIST		NOLIST
MAP | NOMAP		NOMAP
NUMBER | NONUMBER	NUM | NONUM	NONUMBER
OBJECT | NOOBJECT	OBJ | NOOBJ	OBJECT
OFFSET | NOOFFSET	OFF | NOOF	NOOFFSET
OPTIONS | NOOPTIONS	OPTN | NOOPTN	OPTIONS
QUOTE | APOST		QUOTE
PRINT | NOPRINT		PRINT
PROMPT | NOPROMPT		PROMPT
RUNMSG(n)		RUNMSG(1)
SEQUENCE | NOSEQUENCE	SEQ | NOSEQ	NOSEQUENCE
SOURCE | NOSOURCE	S | NOS	SOURCE
STMT | NOSTMT		STMT
XREF | NOXREF	X | NOX	NOXREF

Figure 7.5 List of options available at compilation

CMPAT(SIGNC) or CMPAT(SIGNF) — CM(SIGNC) or CM(SIGNF)

The CMPAT option determines whether a *C* or an *F* will be generated as the positive sign for signed numeric data: signed numeric computational, numeric display sign leading attached, and numeric display sign trailing attached.

If you specify CMPAT(SIGNC), the positive sign will be represented as a *C*; if you specify CMPAT(SIGNF), the positive sign will be represented as an *F*.

The default option is CMPAT(SIGNC).

DECK or NODECK — D or NOD

By default, the compiler puts out object modules with 128-byte records. This option is designed for those users who require 80-byte records, either because they are required by their host remote job entry subsystem, or for any other reason.

The DECK option, together with the OBJECT option, determines the size of the records of the object module put out by the compiler as follows:

OBJECT Option	DECK Option	Object Module Record Size
NOOBJECT	NODECK	No module produced
NOOBJECT*	DECK	80 bytes
OBJECT	NODECK	128 bytes
OBJECT	DECK	80 bytes

The default option is NODECK.

☞***Note 1**: When NOOBJECT and DECK are specified for the compilation, the compiler changes NOOBJECT to OBJECT, and produces an object module.

☞**Note 2**: When compiling more than one source program in the same job, either DECK or NODECK must apply to all the programs being compiled; likewise, OBJECT or NOOBJECT must apply to all the programs. Specify the desired option in a *PROCESS statement before the first program, or, for OS/VS, in the PARM field of the EXEC statement.

☞**Note 3**: If an overlay program is created with the DECK option, it can only be executed when the data set(s) containing the program has a record length of 80. That is, the program cannot later be copied or reformatted to a different record length. Also, a program created with NODECK can only reside in data sets having a record length of 128.

FIPS or NOFIPS

When you specify this option, the compiler will issue a warning message when it finds COBOL statements that don't conform to certain levels of the Federal Information Processing Standard (FIPS); you specify the level of FIPS, as documented in FIPS PUB 21-1, dated 1975 December 1, the compiler is to use as follows:

FIPS(L) causes the compiler to issue a warning message when it finds language extensions above Low Level COBOL.

FIPS(LI) causes the compiler to issue a warning message when it finds language extensions above Low Intermediate Level COBOL.

The default option is NOFIPS.

FLAG or NOFLAG

With the flag option, you specify the types of diagnostic messages the compiler is to issue after finding an error condition. The default is FLAG(I) (all messages will be issued if you don't specify one of the other FLAG options described below).

FLAG(I) - List all messages.

FLAG(W) - List all messages except informatory messages.

FLAG(E) - List all messages except informatory and warning messages.

FLAG(S) - List only severe and unrecoverable error messages.

FLAG(U) - List only unrecoverable error messages.

NOFLAG - Do not list any messages.

The severity levels of the diagnostic messages are discussed under "Compiler Messages" in Chapter 9.

FLUSHERR

If you want a listing of all the possible diagnostic messages that the compiler can issue, specify the FLUSHERR option. When FLUSHERR is in effect, the source program will not be compiled.

GONUMBER or NOGONUMBER — GN or NOGN

If an error is found during execution, an error code appears in the Status Line on the work station screen. If you specify GONUMBER, the line number in the program where the error was found will appear in position 15-20. The line number will be either the numbers in the sequence field of the input record or the statement numbers assigned by the compiler as determined by the STMT or NUMBER option (described later in this chapter).

NOGONUMBER is the default.

LINECOUNT (n) — LC(n)

With LINECOUNT(*n*), you specify the maximum number of lines to be printed on each page of the compiler listing, including heading lines and blank lines; *n* can be from 10 to 999; the default is 56 lines per page.

LIST or NOLIST

If you specify LIST, the compiler will produce a listing with generated machine code and assembler or pseudo-assembler language statements. By default the compiler produces no listing (NOLIST).

The LIST option will override OFFSET if you specify both for the same compilation.

Use the LIST option when you suspect an error caused by the host compiler and before reporting a problem to IBM. An explanation of the contents of the listings produced can be found in the *5280 COBOL Host Compilers Problem Determination Manual.*

MAP or NOMAP

If you specify MAP, the compiler will print on the compiler output listing a map of the data division data items as shown in 7.5. The default is NOMAP.

Figure 7.6 shows a portion of the listing when MAP is specified:

```
VERSION 1 LEVEL  0           IBM 5280 COBOL 5708-CB1                 PROGRAM-ID: LOADIT          03 SEP, 1980      13:15
                                                       DATA  DIVISION MAP
  STMT#    LVL  SOURCE NAME                            LNTH   AREA      DISP      TYPE        RFMT     DICT#
❶ 17    ❷FD ❸EMPMAS-FILE                            ❹ 88  ❺ F      ❻   0   ❼FD ENTRY    ❽ F     ❾ 14
   18      01   EMPRECD                                 88     U          90      GROUP                    15
   19      02   ACREC                                    2     U          90    ALPHANUM                   16
   20      02   EMPNO                                    5     U          92    ALPHANUM                   17
   21      02   ENAME                                   20     U          97    ALPHANUM                   18
   22      02   STRAD                                   20     U         117    ALPHANUM                   19
   23      02   CTYST                                   20     U         137    ALPHANUM                   20
   24      02   ZIPCD                                    5     U         157    ALPHANUM                   21
   25      02   BEGDT                                    6     U         162    ALPHANUM                   22
   26      02   SOSNO                                    9     U         168    ALPHANUM                   23
   27      02   MARST                                    1     U         177    ALPHANUM                   24
   29      77   STATUS-KEY                               2     U         178    ALPHANUM                   25
   30      77   RKEY                                     5     I          86     NUMERIC                   26
   31      77   DEMPNO                                   5     I          91     NUMERIC                   27
```

Figure 7.6. Sample listing with MAP option

The following text explains each of the columns in the figure.

1 STMT#. These digits give the statement number in the listing of the COBOL program. If STMT is specified as an option, the compiler generates a number. If NUMBER is specified as an option, the number generated is the sequence number specified in columns 1-6 of the source program.

2 LVL. Level (01, 02, 03 etc.) of the data-item.

3 SOURCE NAME. Name of data-item.

4 LNTH. Storage length in bytes of the data-item.

5 AREA. Defines the area in which the data has been allocated as follows:

F — File Control Block (FCB)
U — Uninitialized data area
I — Initialized data area
L — Linkage area

6 DISP. The displacement of the data-item in the area indicated at 5 above.

7 TYPE. Type of data-item: FD entry, group, numeric, alphanumeric, etc.

8 RFMT. The format of the record, *F* for fixed or *V* for variable.

9 DICT#. Entry number in the compiler symbol table.

NUMBER or NONUMBER — NUM or NONUM

If you specify NUMBER, it indicates to the compiler that line numbers are written in the sequence field of the source program; the compiler will use these numbers, instead of compiler-generated numbers, in any error messages issued during compilation and execution. The option NOSTMT will be in effect. NONUMBER is the default.

OBJECT or NOOBJECT — OBJ or NOOBJ

If you specify OBJECT, the default, the compiler produces an object module. If you specify NOOBJECT, the compiler doesn't produce an object module.

This option together with the DECK option, determines the size of the records in the object module put out by the compiler. See "DECK or NODECK" earlier in this chapter.

☞**Note**: When compiling more than one source program in the same job, either OBJECT or NOOBJECT must apply to all the programs being compiled. Specify the desired option in a *PROCESS statement before the first program, or, for OS/VS, in the PARM field of the EXEC statement.

OFFSET or NOOFFSET — OFF or NOOF

If you specify OFFSET, the compiler will print a list of relative offsets in main storage for each procedure division verb in the source program; the default is NOOFFSET.

A listing of offsets is useful in identifying the statement being executed when an error occurs and a listing of the object module (obtained by specifying LIST) is not available.

LIST overrides OFFSET.

OPTIONS or NOOPTIONS — OPTN or NOOPTN

When OPTIONS, the default, is in effect, the assembler will print a list of all the options in effect for the compilation, both the options you explicitly express in a PARMS or *PROCESS statement and the default options.

Figure 7.7 shows a listing when OPTIONS was specified.

```
0OPTIONS IN EFFECT FOR THIS COMPILATION:

      FLAG(I)
      LINECOUNT( 56)
      OBJECT
      OPTIONS
      PRINT
      QUOTE
      RUNMSG(1)
      SOURCE
      STMT

      NOFIPS
      NOGONUMBER
      NOLIST
      NOMAP
      NONUMBER
      NOOFFSET
      NOSEQUENCE
      NOXREF
```

Figure 7.7. Sample listing with OPTIONS specified

PRINT or NOPRINT

When PRINT, the default, is in effect, a listing will be produced according to the other options in effect. NOPRINT suppresses all printing regardless of what other options you specify.

PROMPT or NOPROMPT

This option determines whether or not prompts will appear when a COBOL program is loaded. If you specify prompt, the prompts described under "Prompts for Run-Time Options" in Chapter 8 will appear. If you specify

NOPROMPT, the prompt will not appear; the default options, shown in Chapter 8, will be assumed.

Regardless of whether PROMPT or NOPROMPT is specified, prompts for data set information will appear if required; also, all error messages, ACCEPT, DISPLAY, and STOP verbs will be executed.

☞**Note**: When compiling more than one source program in the same job, either PROMPT or NOPROMPT must apply to all the programs being compiled. Specify the desired option in a *PROCESS statement before the first program, or, for OS/VS, in the PARM field of the EXEC statement.

The NOPROMPT option is designed for use with the job-to-job facility described in Chapter 10. NOPROMPT can reduce or eliminate operator action when running a series of jobs in sequence.

QUOTE or APOST

QUOTE, the default option, specifies that quotes (") instead of the apostrophe ('), be used as the delimiter for literals and for the generation of the figurative constant QUOTE.

RUNMSG (n)

The RUNMSG option determines the language of the prompts generated by the COBOL object program to the operator at the 5280 data station. (These prompts appear on the screen when the program is loaded and are described in Chapter 8). If you don't specify a RUNMSG option, the prompts will be issued in English by default. Specify for *n* one of the following:

Option	Language	Type
1	English	Upper/Lower case
2	English	Upper case only
3	French	National
4	French	Multinational
5	German	National
6	German	Multinational
7	Italian	National
8	Italian	Multinational
9	Spanish	National
10	Spanish	Multinational
11	Japanese	Katakana

☞**Note**: *The language specified by the RUNMSG option applies to all source programs compiled in the same job. If you want object modules to produce prompts in different languages, each related source module must be compiled in a separate job with the appropriate RUNMSG option specified.*

SEQUENCE or NOSEQUENCE — SEQ or NOSEQ

If you have written line numbers in the sequence field of the source program, and specify the option SEQUENCE, the compiler will check the sequence of the numbers. The compiler places an asterisk immediately to the left of each Line number that is not in ascending order.

The default is NOSEQUENCE.

SOURCE or NOSOURCE — S or NOS

SOURCE, the default option, cause the compiler to print the following:

- The COBOL source statements you write
- The COBOL source statements (if any) brought in by COPY statements
- The Data Definition Statements (if any) brought in by COPY statements
- The COBOL source statements generated by the Data Definition Statements (if any)
- Compiler messages (as described in Chapter 9), if any

Figure 7.8 shows a portion of the source code listed when SOURCE is specified:

```
VERSION 1 LEVEL  0         IBM 5280 COBOL 5708-CB1            PROGRAM-ID:                 03 SEP, 1980     13:15
COPY-ID  STMT.#  SEQ.#  A...B.........COBOL SOURCE STATEMENTS..............................IDENTFCN

             1          IDENTIFICATION DIVISION.                                          00000010
             2          PROGRAM-ID. LOADIT.                                               00000020
             4          ENVIRONMENT DIVISION.                                             00000030
             5          CONFIGURATION SECTION.                                            00000040
             6          SOURCE-COMPUTER. IBM-370 WITH DEBUGGING MODE.                     00000050
             8          OBJECT-COMPUTER. IBM-5280.                                        00000060
            10          SPECIAL-NAMES.
            11              CONSOLE IS SCREEN.
            12          INPUT-OUTPUT SECTION.                                             00000070
            13          FILE-CONTROL.                                                     00000080
            14              SELECT EMPMAS-FILE ASSIGN TO DISK                             00000090
            14                 ORGANIZATION IS RELATIVE                                   00000100
            14                 ACCESS MODE IS SEQUENTIAL                                  00000120
            14                 RELATIVE KEY IS RKEY                                       00000110
            14                 FILE STATUS IS STATUS-KEY.                                 00000130
            15          DATA DIVISION.                                                    00000140
            16          FILE SECTION.                                                     00000150
            17          FD  EMPMAS-FILE                                                   00000160
            17              LABEL RECORDS ARE STANDARD.                                   00000170
            18              COPY DDS-EMPRECD.
EMPRECD     18   000001 01      EMPRECD.
EMPRECD     19   000002     02  ACREC    PIC X(00002).
EMPRECD     20   000004     02  EMPNO    PIC X(00005).
EMPRECD     21   000006     02  ENAME    PIC X(00020).
EMPRECD     22   000008     02  STRAD    PIC X(00020).
EMPRECD     23   000010     02  CTYST    PIC X(00020).
EMPRECD     24   000012     02  ZIPCD    PIC X(00005).
EMPRECD     25   000016     02  BEGDT    PIC X(00006).
EMPRECD     26   000018     02  SCSNO    PIC X(00009).
EMPRECD     27   000020     02  MARST    PIC X(00001).
            28          WORKING-STORAGE SECTION.                                          00000770
            29          77  STATUS-KEY PIC X(2).                                          00000780
            30          77  RKEY PIC 9(5) VALUE 1.                                        00000790
            31          77  DEMPNO PIC 9(5) VALUE 0.                                      00000790
            32          PROCEDURE DIVISION.                                               00000800
            33          BEGIN.                                                            00000810
            34              OPEN OUTPUT EMPMAS-FILE.                                      00000820
            35                  MOVE SPACES TO EMPRECD.                                   00000850
            36                  MOVE "F" TO ACREC.                                        00000840
            37              DISPLAY
            37             "EMPLOYEE MASTER FILE WITH 100 MASTER RECORDS BEING CREATED."
            37                    UPON SCREEN.
            38              PERFORM LOAD 100 TIMES.                                       00000960
            39              DISPLAY
            39                "MASTER FILE CREATED.  READY FOR UPDATE RECORDS"
            39                    UPON SCREEN.
            40          FIN.                                                              00001000
            41              CLOSE EMPMAS-FILE.                                            00001010
            42              STOP RUN.                                                     00001020
            43          LOAD.                                                             00001030
            44              ADD RKEY 1000 GIVING DEMPNO.                                  00001040
            45              MOVE DEMPNO TO EMPNO.
            46              WRITE EMPRECD INVALID KEY                                     00001060
```

Figure 7.8. Sample listing with SOURCE specified (source program)

Figure 7.9 shows how the Data Definition Statements appear in the source listing.

```
VERSION 1 LEVEL  0        IBM 5280 COBOL 5708-CB1             PROGRAM-ID: LOADIT        03 SEP, 1980    13:15
                                                   DDS SOURCE LISTING
COPY-ID   CDS #           0...+....1....+....2....+....3....+....4....+....5....+....6....+....7....+....8


EMPRECD   D00001          01A         R EMPRECD          B      DSPATR(CS UL)
          D00002          02A           ACREC        2          DSPATR(ND CA)
          D00003          03A                                   CHECK(BY)
          D00004          04A           EMPNO       5D     01 11CHECK(DR RZ)
          D00005          05A                            O      'EMPLOYEE NUMBER' DSPATR(CA)
          D00006          06A           ENAME      20X     02 11CHECK(DR RB)
          D00007          07A                            O      'EMPLOYEE NAME' DSPATR(CA)
          D00008          08A           STRAD      20A     03 11CHECK(DR RB)
          D00009          09A                            O      'STREET ADDRESS' DSPATR(CA)
          D00010          10A           CTYST      20X     04 11CHECK(DR RB)
          D00011          11A                            C      'CITY, STATE'  DSPATR(CA)
          D00012          12A           ZIPCD       5D     05 11CHECK(DR MF FE)
          D00013          13A                            O      'ZIP CODE'  DSPATR(CA)
          D00014          16A           BEGDT       6D     06 11CHECK(AD DR MF FE)
          D00015          17A                            C      'BEGINNING DATE' DSPATR(CA)
          D00016          18A           SOSNO       9D     07 11CHECK(DR MF FE)
          D00017          19A                            C      'SOCIAL SECURITY NUMBER' DSPATR(CA)
          D00018          20A           MARST       1X     08 11CHECK(DR MF FE)
          D00019          21A                            O      'MARITAL STATUS - M OR S' DSPATR(CA)
```

Figure 7.9. Sample listing with SOURCE specified (Data Definition Statements)

STMT or NOSTMT

STMT, the default option, causes the compiler to place a sequence number by all the statements in the COBOL source program. This number will be used in error messages and source listings. If GONUMBER is specified, this number will appear in the status line on the screen display when error messages are issued.

XREF or NOXREF — X or NOX

If you specify XREF, the compiler will print an alphabetic listing of the procedure division names and data names, and the line numbers of where they appear in the source listing.

Figure 7.10 shows a portion of a listing when XREF was specified:

```
VERSION 1 LEVEL  0          IBM 5280 COBOL 5708-CB1           PROGRAM-ID: LUADIT        03 SEP. 1980     13:
                                                    CROSS REFERENCE LISTING
   ❶SOURCE NAME                            DICT# ❸TYPE          DEFN     REFERENCES
  ACREC                                  ❷16    ALPHANUM      ❹19     ❺36
  BEGDT                                    22    ALPHANUM       25
  BEGIN                                    29    PARANAME       33
  CTYST                                    20    ALPHANUM       23
  DEMPNO                                   27     NUMERIC       31       44        45        49
  EMPMAS-FILE                              14    FD ENTRY       17       14        34        41
  EMPNO                                    17    ALPHANUM       20       45        47
  EMPRECD                                  15       GROUP       18       35        46
  ENAME                                    18    ALPHANUM       21
  FIN                                      30    PARANAME       40       51
  LOAD                                     31    PARANAME       43       38
  MARST                                    24    ALPHANUM       27
  RKEY                                     26     NUMERIC       30       14        44        48
  SCREEN                                   13    MNEMNAME       11
  SOSNO                                    23    ALPHANUM       26
  STATUS-KEY                               25    ALPHANUM       29       14        50
  STRAD                                    19    ALPHANUM       22
  ZIPCD                                    21    ALPHANUM       24
```

Figure 7.10. Sample listing with XREF specified

The following text explains each of the columns in the example.

1. SOURCE NAME. Data-items and PROCEDURE names in alphabetic order.
2. DICT#. Entry number in the compiler symbol table.
3. TYPE. Type of data-item: FD entry, group, numeric, alphanumeric, etc.
4. DEFN. The statement number where item or name was defined.
5. REFERENCES: Statement numbers in the source listing where the item or name is used.

Chapter 8. Guide for COBOL Program Execution

This chapter contains information to be used in preparing for and executing a program written in the IBM 5280 COBOL language.

Manuals and Documentation Needed by the Operator

Manuals and other documentation needed by the operator responsible for loading and executing a COBOL program include the following:

- *IBM 5280 Distributed Data System Operator's Guide*, GA21-0364. This manual contains information on starting and stopping the system, running background programs and sort/merge, and a detailed explanation of the keyboard. (The functions of some keys on the keyboard are different for a COBOL program than as explained in the *Operator's Guide*; these differences are explained in Chapter 1 of this publication.)
- *IBM 5280 Utilities Reference/Operations Manual*, GA21-7788. This manual contains information on the utility programs you will use to allocate and and maintain the data sets you need to run a COBOL program.
- *IBM 5280 Message Manual*, GA21-9354. This manual contains an explanation of the error codes and messages that can appear in the status line (the format of the status line is explained in this chapter) as the program executes.

The individuals responsible for writing the program and for systems operations must provide certain information to the operator who runs the program. This information will vary with each program, but typically, the operator will have to to know the following:

- The volume names and the names of the data sets to be used by the program. If the data sets do not already exist, the operator must know how much space to allocate, and the appropriate volume and owner identifications, if any, required by the program.
- Instructions on the use of the COBOL command keys, if the program makes use of these keys.
- Explanations of error codes or messages put out by the program, error recovery procedures, and any other special instructions on the operations of the program. (The format of the error messages is discussed under "The Status Line and Error Messages" later in this chapter.)
- How to respond to the COBOL prompts, which is discussed in this chapter.

Transferring the Load Module from the Host to IBM 5280

The information in this section is included for planning purposes only.

A COBOL load module can be transferred from the host system to the IBM 5280 system either by diskette or over a communications link. The implementation of each method will vary among IBM customers and is a customer responsibility.

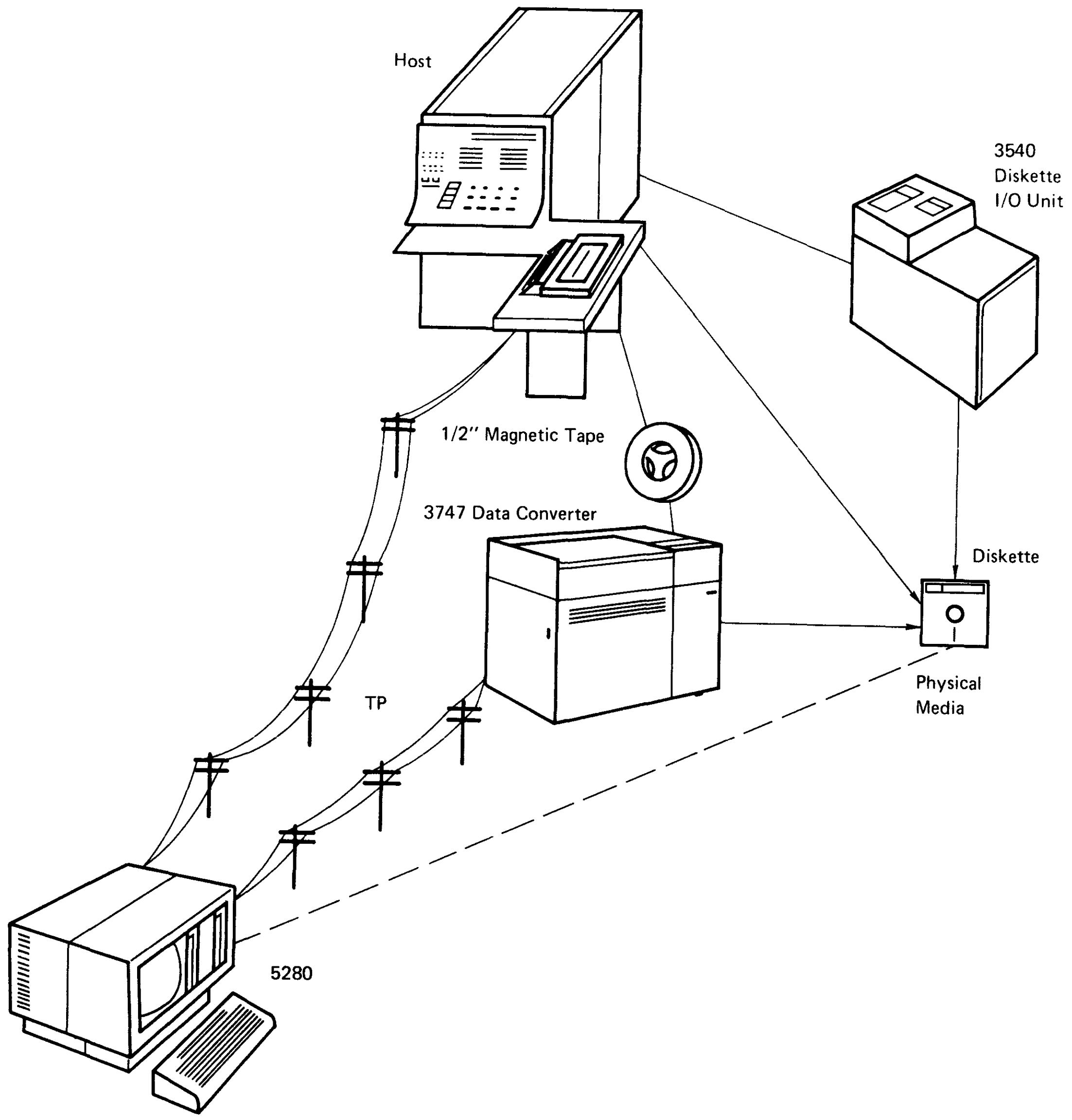


Figure 8.1. Possible methods of host-5280 data exchange

Figure 8.1 gives a conceptual view of three methods of transferring a COBOL load module; they are:

- Over a telecommunications link directly from the host to the 5280. The programming support to do this varies; for example remote job entry (on the host) and the communications utilities (on the 5280) could be used, or user-written routines on the host and 5280. See Chapter 4 for the system requirements for data communications.
- From the host to a diskette device such as the 3540 I/O unit. The diskette could then be hand-carried to the 5280 system. (For information on the diskette formats supported by 5280, see the *System Concepts Manual.*)

- From the host to a tape, and then through a data converter (for example, the IBM 3747 Data Converter) to a diskette. The load modules on the diskette could either be hand-carried to the 5280 system or sent over a telecommunication link.

☞**Note 1**: Ensure that the transparent text mode is specified on both the host and the 5280 when sending object modules over a telecommunications link.

☞**Note 2**: 5280 RJE provides a name for the data set to which the object module is sent. Therefore, you may want to change the name of the data set using the facilities of the 5280 diskette label maintenance (SYSLABEL) utility. *You must change the name when the object modules are segmented. See the section "When Programs Are Segmented" later in this chapter.*

Information on the 5280 communications facilities, including RJE, is described in the *5280 Communications Reference Manual.*

Remote Job Entry Subsystems

If you use a remote job entry subsystem to transfer load modules from the host to your 5280 system, you may have to use the DECK/NODECK option to ensure that the size of the object records conforms with the size required by your subsystem. Some remote job entry subsystems, such as RES, JES2, DOS/VSE POWER, etc., require that the data to be transferred be made up of 80-byte records.

Unless you specify otherwise, the compiler puts out load modules made up of 128-byte records. With the DECK/NODECK option, described in Chapter 7, you can cause the compiler to create load modules with 80-byte records. Use this option if you require 80-byte records for your host subsystem or for any other reason.

Storage Required for Object Programs

The information in this section will help you estimate (1) the storage required for your COBOL object module on the diskette device and (2) the storage required during execution in the partition. These requirements differ, as will be shown.

Storage Required on Diskette

The number of bytes of diskette space needed by your COBOL program is printed on the last page of the compiler listing. Either the OBJECT or LIST option must be in effect during compilation. (These options are described in Chapter 7. Unless you specify otherwise, the OBJECT option by default will always be in effect.)

Main Storage Required for Execution

The main storage required by your program during execution will be *at least* the amount printed on the compiler listing. In addition, you must add space for I/O buffers, which are allocated dynamically during execution.

The amount of storage required for I/O buffers varies with the number of files concurrently opened by your program and the sizes of the records in each file. In allocating I/O buffers for your program, COBOL considers these variables and the optimum performance that can be achieved. Because the variables can differ greatly among users, no simple formula exists for determining the I/O buffer size.

If A Program Is Too Large

If a program is too large for a partition, you should first determine whether or not a larger partition exists in your system. This can be done with the system status utility (SYSSTAT) described in the *Utilities Reference/Operations Manual.*

If a program is too large for any of the partitions available on your system, you can do one of the following:

- Divide the program into segments as described under "Segmentation" in the *5280 COBOL Language Reference*, or
- Increase the partition size using the system configuration program as described in the *System Control Programming Reference/Operation Manual.*

Allocating Data Sets for Object Programs

The data set to hold the object module must be in either the I-exchange or basic exchange format. The size of the I-exchange data set can be either 80 or 128 bytes, depending on the size of the object module as determined by the DECK option at compile time. The size of the basic exchange data set must always be 128 bytes.

Follow the procedures described in the *Utility Reference/Operations Manual* and the *Systems Concepts* manual. Special rules apply when allocating data sets to hold segmented programs, as described below.

When Programs Are Segmented

When programs are segmented, the data set name you specify at allocation is determined by the PROGRAM-ID name of the segmented program. Specify the data set name as follows:

1. If the PROGRAM-ID name is less than or equal to six characters, place the letter *O* after the PROGRAM-ID name. For example, if the PROGRAM-ID is SEND, the data set name for the overlay segments would be SENDO.
2. If the PROGRAM-ID is more than six characters, truncate all characters above six and add an *O*. For example, CONCATENATE would become CONCATO.

☞**Note**: If the 5280 RJE was used to receive the object data set from the host, you will have to change the name of the data set to conform with the above naming conventions using the diskette label maintenance (SYSLABEL) utility.

Allocating Data Sets for Program Files

Before running the program, the operator must ensure the diskette data sets used by your program, if any, are allocated and available. If not, the diskettes must be initialized and the data sets allocated as explained in the *Utilities Reference/Operations Manual.*

(When initializing and allocating diskettes, consider the factors discussed under "Improving Performance" later in this chapter.)

Initializing Diskettes

To initialize the diskette(s) that is to hold the data sets, use the diskette initialization utility (SYSINIT). You will be prompted for information by the utility.

Allocating Data Sets

To allocate the data set(s), use the diskette label maintenance utility (SYSLABEL), which prompts you for the information needed for allocation. Your response to the *exchange type*, the *record size*, the *number of records*, and the *delete character* prompts will depend on how you code your program and other considerations. This is discussed in the sections that follow.

Specifying a Multivolume Indicator

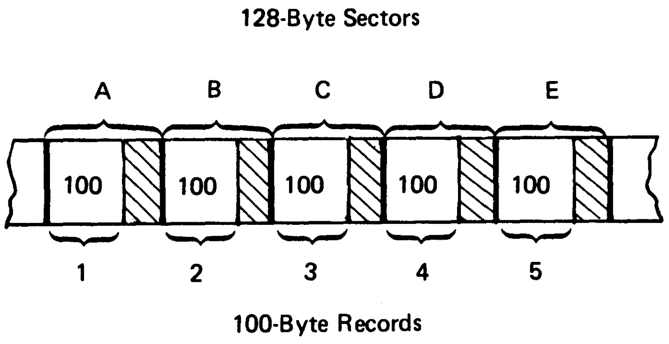
If your data set is located on more than one diskette volume, you must specify a multivolume indicator during allocation. See "Multivolume Record Processing" in Chapter 5 for the conventions to follow in specifying multivolumes.

Determining Exchange Type

You should be aware of the data set *exchange types* you can allocate for the most efficient operation of your program.

SYSLABEL will prompt you to enter one of the following exchange types: basic, *H*, or *I*. The data exchange you select determines the structure of your data set as discussed below.

Unblocked and unspanned (basic and *H*) means that records are not blocked together and that the records must each start on sector boundaries. Each record is one block of data. On the 5280, unblocked and unspanned records cannot be longer than sectors.

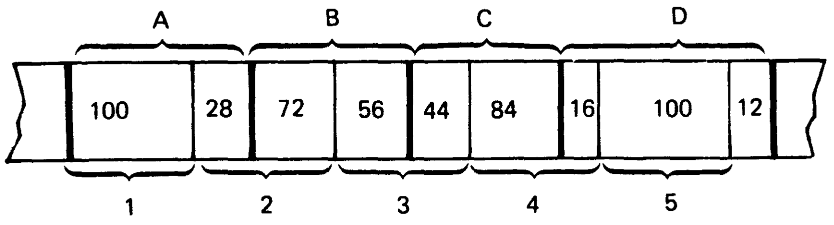


In the preceding example, each 100-byte record starts at the beginning of a 128-byte sector boundary. The remaining 28 bytes in each sector are not available for data storage. Some of the potential storage space is thus wasted.

The basic and *H* exchange types use the unblocked and unspanned data structure.

Blocked and spanned (I) means that records are blocked together, and that sector boundaries are not necessarily related to record positions.

128-Byte Sectors



100-Byte Records

In the preceding example, 100-byte records are placed next to each other with no regard for sector boundaries. No diskette storage space is unused.

The *I* exchange type uses the blocked and spanned data structure. Unless you need to exchange data with another system that accepts only unblocked and unspanned data, the blocked and spanned data structure is recommended.

For detailed information on IBM 5280 data set concepts, the different formats for diskettes, and the requirements for data exchange with other IBM systems using diskettes, see the *System Concepts* manual.

Record Size

The size of the record as defined in the COBOL program and the size of the record allocated must be the same. If you are allocating the index file to be used for key indexed data sets, the record size must be the length of the key plus four.

Number of Records for Indexed Files

This section describes how to determine the number of records to enter when the *specify number of records to be allocated* prompt appears. This information applies only when (1) you allocate a data set for a file with an indexed organization and (2) you are going to create that file with the sequential access method.

In creating the file, COBOL writes every tenth record as a null record, to make future updates more efficient. Therefore, in determining the number of records to allocate, estimate the number of records to be created from your program, and then increase this number by 10%. If a separate data set exists for the index, this applies only to the index data set.

Delete Character

☞**Note 1**: If the COBOL file organization is indexed or relative, and the data exchange type is *I*, you must specify a 5280 delete character in response to the prompt (except as noted in Note 2). Ensure that the character is one that will never appear as the last character of a COBOL record. Otherwise, the record will be deleted by the system.

☞**Note 2**: Do not specify a delete character for the index data set, when, for an indexed file, you've specified separate data sets for the index and for the application records.

You don't have to specify a delete character for the data set containing the application records unless you intend to delete records from that data set.

Improving Performance

Reducing the amount of processing time and other improvements in the performance of a COBOL program depend generally on two factors: (1) coding techniques and considerations and (2) IBM 5280 system considerations.

System Considerations

System considerations include how many partitions are allocated, in which partition the COBOL program is to execute in relation to other programs, and the location of diskette data sets.

Performance could improve if you do the following:

- If only one diskette device is available, place the data sets to be used concurrently as close together as possible on the diskette.
- Preferably, if more than one diskette device is available, place such data sets on separate devices.

For additional guidelines on improving performance in this area, see Chapter 7 in the *System Concepts* manual.

Improved Performance with Sequential Access Method

When updating files with a large number of records, a program will execute faster when using the sequential rather than the random access method. However, if only a few records are to be updated, the random access method will be more efficient.

Improved Performance when Creating Indexed Files

When creating files with an indexed organization, a program will execute faster when using the sequential rather than the random access method.

Improved Performance with SIZE Clause

In processing files with an indexed organization, you can affect performance with the SIZE clause in the File-Control Paragraph. With the SIZE clause, you determine the size of the in-storage index. This clause can be used only when random access is specified.

By increasing the index size, performance may be improved because of less search time to find the desired record. However, you must also consider the additional storage required when increasing the size of the in-storage index.

See "Processing Files with an Indexed Organization" in Chapter 5 for information on how to code the SIZE clause.

Loading the COBOL Program and Responding to Prompts

In loading and executing a COBOL program, the operator performs the following:

1. Loads (IPLs) the system.
2. Loads the COBOL program.
3. Responds to prompts in specifying the COBOL run-time options. These prompts will appear only when the PROMPT option (described in Chapter 7) has been specified during compilation.
4. Responds to prompts, if any, in specifying diskette files required by the program.

5. Responds to prompts, if any, in specifying printer files required by the program.

The following sections show the layout of each prompt and explains the various responses that are possible.

Prompts for Loading a COBOL Program

To load the COBOL program, respond to the following prompt:

```
0 0001        A 16   40

Program name:
Device address:
Partition number:

                  Press ENTER
```

1. Insert the diskette containing the COBOL program.
2. Enter the name of the data set that holds the COBOL program after *Program name.*
3. After *Device address*, enter the address of the device where the diskette containing the COBOL program is inserted.
4. Enter the number of the partition where the program will execute after *Partition number.* (If a partition number is not entered, the number defaults to the number of the partition associated with the keyboard.)
5. Press the Enter key.

Prompts for Run-Time Options

If the PROMPT option (described in Chapter 7) is in effect, the following prompts will appear. Either press the Enter key to accept the default options as displayed, or change the options as desired and press the Enter key.

```
User parameters:  parm
Debug option (1=Debug, 2=Nodebug):  1
UPSI switch values:  00000000
SYSOUT Device (1=Display, 2=Printer):  2   Device address:  8000


                         Press Enter
```

User Parameter

parm represents parameters the operator can enter. The maximum number of characters for *parm* is 50.

The contents of the user parameter are determined when the COBOL program is written. The data to be passed is described in the Linkage Section of the Data Division of the first routine in the COBOL program to receive control after invocation. The following statements show the entries required to describe a ten-byte parameter:

```
LINKAGE SECTION.
01 PARM1.
   02 PARM1-STRING PIC X(10).
```

PROCEDURE DIVISION USING PARM1.

To pass a ten-character parameter string *ABCDEFGHIJ* to the PARM1 data-item linkage section described earlier, the operator would enter *ABCDEFGHIJ*. PARM-STRING would be *ABCDEFGHIJ* when the routine is invoked.

Debug Option

If 1 (for Debug) is entered after *Debug option*, the USE FOR DEBUGGING procedures will be executed in your COBOL program. If 2 (for NODEBUG) is entered, these same procedures will not be executed. An example and an explanation of a USE FOR DEBUGGING procedure is given in Chapter 9 under "Debugging Language".

UPSI Switch

The value the operator enters (a combination of 0s and/or 1s) is assigned the eight switches UPSI-1, UPSI-2. . . UPSI-7 defined in your COBOL program. A *0* indicates the *off* condition; a *1* indicates the *on* condition.

You determine the use of the UPSI switch by the logic in your program. See the *5280 COBOL Language Reference* under the "SPECIAL NAMES PARAGRAPH" for information on how these switches are defined.

SYSOUT Device

The SYSOUT device receives messages generated by DISPLAY statements within the COBOL program, depending on how the COBOL program is coded (see the special rules that apply under "DISPLAY and ACCEPT" in Chapter 6).

Enter a *1* to display all messages on the printer or a *2* to display all messages on the work station screen.

Device Address

Enter the address of the device to which printer files are to be written. This is the device used by files which are specified with ASSIGN TO PRINTER in the associated SELECT clause as described under "Printer I/O" in Chapter 6.

Prompts for Diskette Files

Respond to the following prompt:

```
Enter the following information for file file-name
Dataset name:
Device address:
Owner id:

                    Press ENTER
```

file-name is the first eight characters of the file name you code in the SELECT statement in your COBOL program.

This prompt may or may not appear, depending on how you code the entries in the File-Control paragraph. See the section "File-Control Paragraph" in Chapter 5 for details.

file-name is the first eight (8) characters of the FD name you code for the FD name in your COBOL program except, in some cases, for indexed files.

For indexed files with two data sets (an application record data set and an index data set), *file-name* is the first seven (7) characters of the FD name of the application data set; the eighth character is an asterisk (*). If the FD name for the application data set is less than seven (7) characters, asterisks (*) will pad out *file-name* to a full eight (8) bytes.

☞**Note**: Make sure that the operator responsible for running a program which uses indexed files with two data sets is aware of the *file-name* conventions just described, and the appropriate responses.

Dataset Name

Depending on the way the program is coded, the operator might enter one of the following:

> *dsname*, which is the name of the data set specified when it was allocated with the diskette label maintenance utility or
>
> *volid.dsname*, where *volid* is the name of the volume specified when the diskette was initialized with the diskette initialization utility.

The maximum number of characters that can be specified with either entry is 26.

Device Address

For device address, the operator can enter either the 4-digit physical address (4000, 4400, etc.) or the logical device ID specified when the system was configured. If the logical ID is entered:

- It must be left justified.
- The Alpha (shift) key must be pressed during entry.

owner-ID is from 1 to 14 characters as specified when the diskette was initialized with the diskette initialization utility (SYSINIT).

The operator must fill in *owner-ID* when the volume-protect (accessibility) field has been set. This is done with option 3 (modify volume label) of the diskette label maintenance utility (SYSLABEL).

The operator needn't enter *owner-ID* when the volume is protected and a VALUE OF OWNER-ID clause is specified in the FD entry for the diskette file.

Prompts for Printer Files

The prompts for information on printer files appear in the following format:

```
Enter the following information for file file-name
Printer device address:

                  Press ENTER
```

file-name is the first eight characters of the file name you code in the SELECT statement in your COBOL program.

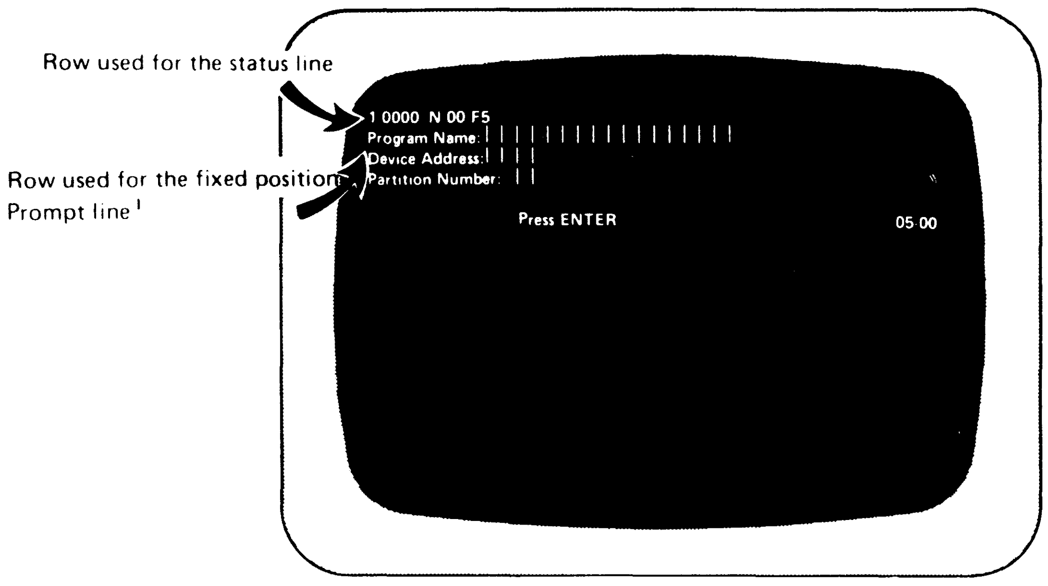
Device Address

For device address, the operator can enter either the 4-digit physical address (4000, 4400, etc.) or the logical device ID specified when the system was configured. If the logical ID is entered:

- It must be left-justified.
- The Alpha (shift) key must be pressed during entry.

The Status Line and Error Messages

Figure 8.2 shows the layout of information as it appears on a 5280 screen.


Row used for the status line
Row used for the fixed position Prompt line ¹

```
1 0000 N 00 F5
Program Name:| | | | | | | | | | | | | | | |
Device Address:| | | |
Partition Number: | |
              Press ENTER                    05 00
```

Figure 8.2. Layout of 5280 screen information

Row 1 of the display is reserved for the status line, which provides information about the operation of the system; row 2 is reserved for fixed position prompts. These prompts are defined with the PMT keyword in a field description statement as described in Chapter 2.

The remainder of the screen can be used as prompt lines by the COBOL applications you write. The number of lines available depends on the size of the screens available in your 5280 system.

Four status line formats are used depending upon the conditions being displayed:

- Normal operations
- Keyboard error
- Device error
- COBOL Execution Error

The format of the status line for normal operations and for keyboard errors is the same regardless of whether the program is written in COBOL, RPG, or assembler.

(See either the *Operator's Guide* or the *Messages Manual* for a description of the status line for normal operations and for keyboard errors.)

The format of the status line for device errors and execution errors differs when generated by a COBOL program. These formats are discussed in the sections that follow.

Device Errors

I/O device errors that occur during program execution are also indicated on the status line. After a device error, the status line is in the following format:

```
----+----1----+----2----+----3----+----4
P AAAA CCCC-LL PPPPPPPP DDDDDDDDDDDDDDDD
```

P (in position 1) is the partition number.

AAAA (in positions 3-6) is the address of the device in the form of a four-digit code (for example 4000, 4400, etc.)

CCCC (in positions 8-11) is the error message code. This code corresponds to a recovery procedure in the *Message Manual.* The operator should write the error code on a piece of paper, or already know the correct error procedure before pressing the Reset key.

LL (in positions 13-14) is the logical device ID.

PPPPPPPP (in positions 16-23) is the name assigned to the COBOL program in the IDENTIFICATION DIVISION of the source program. If multiple COBOL programs are called, it is the name of the main program.

DDDDDDDDDDDDDDDD (in positions 25-40) corresponds to the last 16 positions of the data set name given at allocation.

COBOL Execution Errors

COBOL execution errors usually occur because of faulty logic in the program. Some possible causes of these errors are discussed in Chapter 9 under "Possible Causes of Execution Error Messages".

After a COBOL execution error, the status line is in the following format:

```
----+----1----+----2----+----3----+----4
P      CCCC            NNNNNN IIIIIII
```

P (in position 1) is the partition number.

CCCC (in positions 8-11) is the error message code. This code corresponds to a recovery procedure in the *Message Manual.* The operator should write the error code on a piece of paper, or already know the correct error procedure before pressing the Reset key.

NNNNNN (in positions 25-30) corresponds to the number in the compiler listing of the COBOL program where the error occurred. The number is determined by the STMT or NUMBER options as described in Chapter 7. The number is displayed only if the GONUMBER option (also described in Chapter 7) is specified at compilation.

IIIIIII (in positions 32-39). The characters in these positions, when present, are explained for the related message in the *Messages Manual.*

Chapter 9. Debugging

This chapter contains information to aid you in debugging a COBOL program. Topics covered include:

- Compiler messages, which indicate syntax and other errors found in your program during compilation
- Compiler Abends and their causes
- Program error messages, which are caused by logic error in your program and which are issued during execution of the program on the 5280
- COBOL debugging language, which you can write in your source program as an aid in isolating and correcting program errors

Compiler Messages

The primary means for locating errors in the source program is through the diagnostic messages produced by the compiler. (If you need a list of all the possible messages the compiler can generate, use the FLUSHERR option as described in Chapter 7.)

Compiler messages are generally the result of violating the rules of the COBOL language as defined in the *5280 COBOL Language Reference*. The compiler issues several types of messages in varying degrees of severity.

Types of Messages

The compiler can issue five types of messages:

- Information Messages
- Warning Messages
- Error Messages
- Severe Error Messages
- Unrecoverable Error Messages

Information (I) messages call attention to possible inefficiencies in the object module or give other information generated by the compiler that may be of interest.

Warning (W) messages call attention to possible errors, although the statements to which they refer are syntactically valid.

Error (E) messages indicate an error that the compiler can correct with a high degree of confidence that program execution will be correct.

Severe (S) error messages indicate errors that cannot be corrected with any degree of confidence by the compiler. Execution will almost certainly fail or produce incorrect results.

Unrecoverable (U) error messages indicate errors that caused the compiler to terminate compilation of the program, or that the compiler detected errors that would make the resulting load module unexecutable.

You determine the types of error messages the compiler can issue by the specification you make with the FLAG option at compilation. See Chapter 7 for details. If you don't specify the FLAG option, the compiler will issue all five types of messages described above; the compiler always lists unrecoverable errors regardless of the FLAG specification.)

Acting Upon Compiler Messages

All messages except information messages should be acted upon, even if the compiler has been able to 'fix' the error. The compiler, in making an assumption as to the intended meaning of the erroneous source statement, can introduce a further error which in turn can produce another error, and so on. When this occurs, the compiler issues a number of diagnostic messages that may all be caused either directly or indirectly by one error.

It is recommended that you use the the flag option *I* or *W* as described under *FLAG or NOFLAG* in Chapter 7. Using another FLAG option may suppress an error message needed to debug your program.

Compiler Abends

There are four types of errors that cause the compiler to cease normal processing and end abnormally.

- Source Program Content — A source program may be too large for the compiler to process in its current configuration.
- User Errors — Defining compiler data sets with improper attributes or insufficient space are examples of user errors. Almost all of these conditions are recognized by the compiler and an appropriate diagnostic message is issued. A few user errors may cause further errors that mask the true problem.
- Compiler Logic Errors — An unexpected condition can cause a compiler phase to stop processing or a further type of error to occur such as a program check. Compiler logic errors are sometimes recognized by the compiler and an appropriate diagnostic message issued, but more often the result is an abnormally terminated compilation thought to be caused by a user programming error.
- System Errors — Hardware errors and programming logic errors within the operating system are examples of system errors. Because these types of errors are not recognized by the compiler, the true cause of the error will not be clear. Most hardware errors, however, are processed by special system routines that can produce diagnostic messages to the console and listing file that may enable the operator to correct the error and rerun the job. These messages can be found in the appropriate *Messages* manual for the host system on which your compiler runs.

The compiler sets a return code after compilation to reflect any error conditions. See Appendix D for the return codes and their possible causes. The code set and the action taken for each error is as follows:

- If the error occurs in the compiler initialization phase, an immediate return is made to the operating system and no further output is produced. The return code is 1000 and is added to the internal compiler error code. The compiler return codes are given in Appendix D.
- If the error occurs in the compiler termination phase, return is made to the operating system but additional formatted debugging information is produced. The return code is 2000 and is added to the internal compiler error code.

The compiler return codes can help personnel responsible for isolating problems in the compiler. More information on this subject can be found in the *5280 COBOL Host Compilers Problem Determination Manual.*

Abnormal Termination and SYSDOVL

☞**Note**: An abnormal termination will occur with segmented programs if the primary program (defined by SYSLDOUT) and the overlay modules (defined by SYSLDOVL) are members of the same partitioned data set. If this occurs, make sure two different data sets are defined for SYSLDOVL and SYSLDOUT before recompiling.

Messages at Execution

A number of messages associated with a COBOL program can be issued as the COBOL program executes on an IBM 5280 system. These messages appear in the status line at the top of the screen. They contain a four-character code which refers to an explanation in the *Messages Manual.* The format of the different types of messages that can appear in the status line are described under *The Status Line* in Chapter 8.

Possible Causes of Execution Errors

Some areas where a program logic error will not be found by the compiler, but may cause problems at execution are listed below with a possible remedy.

- INDEX or subscript is out of range. Use the debugging line facility or the USE FOR DEBUGGING declaratives to check values periodically throughout the program.
- REDEFINEs is used with different USAGE clauses. Verify that the USAGE clauses are compatible.
- Logic errors after a group MOVE. Your program may have executed a group MOVE, and the data-items in the group did not have matching PICTURE and USAGE clauses. Verify that the corresponding data-items are compatible. There are no execution-time checks made as to data validity. You can code COBOL language elements, such as ALPHABETIC and NUMERIC class tests, to verify the data before executing the MOVE.
- PICTURE clause arithmetic values exceed the precision of the PICTURE clause, which causes significant digits to be lost. Use ON SIZE ERROR clause to determine when this occurs.

The following conditions can cause unpredictable results:

- A READ which references data-items defined within a variable length record when a shorter length record is read
- Referencing a data-item within a record before the file has been opened
- Using data within a record description after a WRITE or REWRITE verb

Debugging Language

COBOL provides two switches which ease the writing of error-free programs:

- The WITH DEBUGGING MODE switch, which serves as a compile-time switch for debugging statements written into the source program
- The DEBUG/NODEBUG switch, which serves as an object-time switch that activates debugging procedures written into your program

Information on how to use these two switches is covered in the sections that follow. For detailed information, see the section "Debugging" in Part V of the *COBOL Language Reference.*

Debugging Lines — WITH DEBUGGING MODE

A debugging line is any COBOL statement you write in your program with a *D* in column 7. You can use debugging lines to assist in locating logical errors in your program. When you write a WITH DEBUGGING MODE clause in the SOURCE-COMPUTER paragraph of the Configuration Section, the debugging lines are made a part of the object code and will be executed with the object program.

If the WITH DEBUGGING MODE clause is removed, the debugging lines will be treated as comments and will not be executed.

The execution-time options DEBUG or NODEBUG (discussed in Chapter 8) do not affect debugging lines; they affect only the USE FOR DEBUGGING declarative, as described in the next section.

Declarative Procedures — USE FOR DEBUGGING

The USE FOR DEBUGGING declarative allows you to create procedures within your program to examine its internal status during execution. With the USE FOR DEBUGGING declarative, you identify program elements you wish to monitor. COBOL then gives control to your debugging procedure when these elements are referenced during execution. Your procedure also has access to the DEBUG-ITEM special register, which contains information about the conditions causing the activation of the debugging procedure.

You can control the USE FOR DEBUGGING procedures with two switches: the WITH DEBUGGING MODE source clause at compile-time, and the DEBUG/NODEBUG option at execution-time. If you specify WITH DEBUGGING MODE, COBOL compiles the procedures as executable code; otherwise COBOL treats them as comments.

If you specify WITH DEBUGGING MODE at compile time, and the DEBUG option at execution-time, the debugging procedures will be executed; if you specify WITH DEBUGGING MODE at compile time, and the NODEBUG option at execution time, the procedures will be bypassed.

Example - COBOL Debugging Language

The example in figure 9.1 shows the use of debugging lines and a debugging declarative procedure. Here is an explanation of some of the statements (keyed to the example) used:

The WITH DEBUGGING MODE clause of the SOURCE-COMPUTER paragraph at $\boxed{1}$ causes the debugging statements in the program to be compiled.

Each time the paragraph READ-MASTER at $\boxed{3}$ is entered, control is passed to the statements following USE FOR DEBUGGING ON READ-MASTER at $\boxed{2}$. After the statement executes, control is returned to READ-MASTER, where the statements, beginning with SUBTRACT, execute.

You can control whether or not this procedure executes through the debug option when you load the program. See “Prompts for Run-Time Options” in Chapter 8 for more information.

The debugging lines (a D in column 7) at 4 are compiled and execute as any other source statement. The use of debugging lines is another method of tracing program flow during execution. Debugging lines will execute regardless of what you specify as the debug option when the program is loaded.

```
     IDENTIFICATION DIVISION.
     PROGRAM-ID.  UPEMPMAS.
     AUTHOR.  A NAME.
     ENVIRONMENT DIVISION.
     CONFIGURATION SECTION.
1    SOURCE-COMPUTER. IBM-370 WITH DEBUGGING MODE.
     OBJECT-COMPUTER. IBM-5280.
     SPECIAL-NAMES.
         ATTRIBUTE-DATA IS TERMINAL-INFO.
     INPUT-OUTPUT SECTION.
     FILE-CONTROL.
         SELECT SCREEN-FILE-PR3 ASSIGN TO WORKSTATION 1920
           ORGANIZATION IS TRANSACTION
           ACCESS MODE IS SEQUENTIAL
           FILE STATUS IS TUBE-STAT
           CONTROL-AREA IS WSTATION-CONTROL-AREA.
         SELECT EMPMAS-FILE ASSIGN TO DISK
            ORGANIZATION IS RELATIVE
            ACCESS MODE IS RANDOM
            RELATIVE KEY IS RKEY
            FILE STATUS IS DISK-STAT.
     DATA DIVISION.

         •
         •
         •

     PROCEDURE DIVISION.
     DECLARATIVES.
2    DEBUG-SECTION SECTION.
        USE FOR DEBUGGING ON READ-MASTER.
            DISPLAY "READ-MASTER ENTERED".
            DISPLAY "REKEY= " RKEY.
            DISPLAY "EMPNO= " EMPNO.
     IO-ERROR SECTION.
        USE AFTER ERROR PROCEDURE ON SCREEN-FILE-PR3.
     WORK-STATION.
             DISPLAY "ERROR ON WORK STATION I/O".
             DISPLAY "FILE STATUS IS " TUBE-STAT.
             DISPLAY "RUN STOPPED.".
             STOP RUN.
     DISKETTE-IO-ERROR SECTION.
        USE AFTER ERROR PROCEDURE ON EMPMAS-FILE.
     DISKETTE.
             DISPLAY "ERROR ON DISKETTE I/O".
             DISPLAY "FILE STATUS IS " DISK-STAT.
             DISPLAY "RUN STOPPED." .
             STOP RUN.
     END DECLARATIVES.
```

Figure 9.1. Coding example with COBOL debugging language (Part 1 of 2)

```
    EXECUTE SECTION.
        •
        •
        •
        PERFORM UPDATE-MASTER.
3   READ-MASTER.
        SUBTRACT 1000 FROM EMPNO GIVING RKEY.
4  D DISPLAY "EMPNO - 1000 = RKEY IS " RKEY.
        READ EMPMAS-FILE
           INVALID KEY
              MOVE 0 TO CONTINUE
              MOVE 1 TO ERROR-FOUND, INVALID-EMPNO.
        IF CONTINUE = 1
           IF ACREC NOT EQUAL TO "F"
              MOVE 1 TO ACTIVE-RECORD, ERROR-FOUND
              MOVE 0 TO CONTINUE
        ELSE MOVE "A" TO ACREC.
    UPDATE-MASTER.
4  D DISPLAY "UPDATE-MASTER ENTERED".
        MOVE SCREEN-RECORD TO EMPRECD.
        •
        •
        •
```

Figure 9.1. Coding example with COBOL debugging language (Part 2 of 2)

Chapter 10. Job-to-Job Facility

The COBOL job-to-job facility allows you to pass control from your program to another COBOL program or any other program (for example, a DE/RPG program, an assembler program, a system utility program, etc.). You achieve this linkage with a CALL to a COBOL library routine as will be described in this chapter.

Together with the PROMPT/NOPROMPT option described in Chapter 7, this facility permits you to control linkage among several programs with little or no operator action.

COBOL Statements Required

The following is the format of the CALL statement used within a COBOL program to load and execute the next program:

```
CALL "AVCSETX0" USING parameter-1
```

parameter-1 refers to an area you code in the Data Division of your program in the following format:

```
01  parameter-1
    02 device-id          PIC XXXX.
    02 dataset-name       PIC X(25).
```

device-id specifies the diskette device where the next program to be executed resides. You can code either a two-byte logical ID followed by two blanks, or a four-byte physical address in this field.

dataset-name specifies the name of the data set that contains the next program. You can code this field in the following format:

**volume.dsname*

volume can be any volume ID up to six (6) bytes long; code an asterisk (*) before the volume ID name. The volume ID is optional.

dsname is the name of the data set. The number of characters you can specify depends on the exchange type of the data set as follows:

1. For H, I, or basic exchange data sets, the data set name can be up to 8 bytes in length.
2. For E exchange data sets, the data set name can be up to 17 bytes in length. The data set name can be made up of one or more qualifiers of up to 8 alphanumeric characters each; each qualifier must be separated by a period (.) with no intervening blanks. For example: *ONE.TWO.THREE*

   In determining the length, each period counts as 1 byte.

☞**Note**: If parameter-1 contains blanks, the effect is to reset the job-to-job facility to not load another program at termination.

When Using The Job-to-Job Facility

When using the job-to-job facility, you must consider the following:

1. Control is passed to the specified program when your program ends, either normally or abnormally.
2. The next program is always loaded into the same partition as the current program.
3. You can code more than one CALL to the job-to-job facility. This allows the logic of your program to determine which program will be loaded and executed next after, for example, a STOP RUN statement in the current program is executed.
4. In programs with more than one CALL to the job-to-job facility, the last CALL processed determines which succeeding program will be loaded.
5. Be sure to specify the NOPROMPT option described in Chapter 7 when compiling the next job to be run. This ensures that this job can be loaded and executed with little or no operator intervention.

Appendix A. Direct Data Communications Support

This appendix describes how to code data communication programs using direct support provided by IBM 5280 COBOL.

Direct communications support gives more direct control over all communications facilities than the data communications support described in Chapter 4. Direct support is more complicated to use, but provides more flexibility in writing a communication program.

☞**Note 1**: this level of COBOL communications support may be used to produce specialized interface code when necessary. The user is encouraged to use a higher level communications interface for application program development. The level of interface described in this section may not be supported by IBM on other products and thus could cause difficulties in converting to other systems.

☞**Note 2**: Don't intermix CALLs between routines for direct support described in this appendix with the communication routines described in Chapter 4. If the Open routine for direct support is invoked, all subsequent CALLs for COBOL communication support must be made to the other direct support routines.

Before Running a Data Communication Program

Before running a data communication program, you must ensure that the proper communications environment exists for your program. The communications environment is established with two of the IBM 5280 Data Communication Preparation Utilities:

- The Communications Configuration Utility, to which the characteristics of your software and hardware are described; the utility places this information in a *communications configuration record.*
- The Communications Load Utility, which loads the communications access method (CAM) into a background partition.

See the *IBM 5280 Communications Reference Manual*, SC34-0247 for detailed information on these two utilities.

Functions Supported

The following table lists the direct support library routines and the functions they perform.

Routine	Function
TINIT	Defines the environment and initializes line.
TTERM	Terminates connection between transmitting and receiving stations.
TOPEN	Prepares for transmit and receive operations.
TCLOZ	Ends communication operations.
TREAD	Reads a record into a buffer at the receiving station.
TWRT	Writes a record from the sending station to the receiving station.
TCTL	Permits use of BSC line control.

After execution, each routine returns a completion code in a data item you code in your COBOL program as described later in this appendix.

Horizontal and Vertical TAB translations and Data Formatting (specified by HTAB and VTAB in assembler language programs) are not supported. However, you can use any feature which the communication access method supports, even though you cannot specify them in the parameter lists described later in this chapter. Instead, you specify the feature when you run the communication preparation utilities. See the Communications Configuration Utility described in the *Communications Reference Manual* for a list of the features available and information on preparing your communications environment for your program.

TINIT Routine

This routine is called to define a communications environment and initialize the line. This must be the first routine called before any Communications I/O. Any options specified in the parameter list will override the same options you specify in the communications control record created by the IBM 5280 Communications Configuration Utility.

The format of the statement to call the TINIT routine is:

CALL "AVCLINIT" USING *parameter-1 buffer-2*

parameter-1 and *buffer-2* are data-items you code as follows:

```
01 parameter-1
  02 return-code        PIC 9999.
  02 record-length      PIC 9999.
  02 blocksize          PIC 9999.
  02 protocol           PIC AAA.
  02 record-format      PIC AA.
  02 type-1             PIC AA.
  02 type-2             PIC AA.
01 buffer-2   PIC X(n).
```

buffer-2 is where you place the record to be sent, or where you want a record to be received. When sending data under SNA, you should, if required, place the log-on data in the first 80 bytes of this record.

The contents of the *parameter-1* record is as follows:

return-code — contains a code posted by the TINIT routine upon completion. The return-codes and their meanings are explained in Appendix H of the *IBM 5280 Communications Reference Manual*, SC34-0247.

record length — Specify the length of the record to be read.

blocksize — Specify the maximum length of the block to be transmitted. Specify 0, if you want to use the blocksize specified in the communications control record (CCR).

protocol — if system network architecture is used, specify SNA. Otherwise, BSC is assumed.

record-format — for BSC only (this field is ignored for SNA). Specify one of the following:

- FU, for fixed length unblocked
- VU, for variable length unblocked
- FB, for fixed length blocked

- VB, for variable length blocked. If VB is specified, Either IRS or ITB must be specified as delimiters in the communications control record.

If you don't specify any of the above, fixed unblocked (FU) format is assumed.

type-1 — specifies the data set attribute as follows for BSC (for SNA, CM is assumed):

- CM, for READ/WRITE in any order
- SR, for sequential read
- SW, for sequential write
- CN, for conversational: Write 1 message and read many

If you don't specify any of the above for BSC, CM is assumed.

type-2 — Data set attribute 2. For BSC only.

- CB - Expand any blank characters that, when received, are compressed.
- BT - Truncate trailing blanks in data to be sent.

If neither of the above is specified, this option is ignored.

TTERM Routine

This routine is called to terminate the logical connection between the COBOL application program and the host communication program.

The format of the statement to call the TTERM routine is:

`CALL "AVCLTERM" USING` *return-code*

return-code is a data-item you code as follows:

```
01 return-code   PIC 9999.
```

return-code — contains a code posted by the TTERM routine upon completion. The return-codes and their meanings are explained in Appendix H of the *IBM 5280 Communications Reference Manual*, SC34-0247.

TOPEN Routine

This routine is called to establish the beginning of communications transmit and receive operations. It must be called before any read, write or device control.

The format of the statement to call the TOPEN routine is:

`CALL "AVCLOPEN" USING` *return-code*

return-code is a data-item you code as follows:

```
01 return-code   PIC 9999.
```

return-code — contains a code posted by the TOPEN routine upon completion. The return-codes and their meanings are explained in the *IBM 5280 Communications Reference Manual*, SC34-0247.

TCLOZ Routine

This routine is called to end communications operations.

The format of the statement to call the TCLOZ routine is:

CALL "AVCLCLOZ" USING *return-code*

return-code is a data-item you code as follows:

```
01 return-code  PIC 9999.
```

return-code — contains a code posted by the TCLOZ routine upon completion. The return-codes and their meanings are explained in the *IBM 5280 Communications Reference Manual*, SC34-0247.

TREAD Routine

The TREAD routine is called to instruct CAM to read a record from the host into *buffer-2* defined when TINIT is called. TREAD must be repeatedly called until an end-of-data or other appropriate return-code is returned.

The format of the statement to call the TREAD routine is:

CALL "AVCLREAD" USING *parameter-1*

parameter-1 is a data-item you code as follows:

```
01 parameter-1
   02 return-code        PIC 9999.
   02 record-length      PIC 9999.
   02 read-type          PIC X.
   02 read-status        PIC X.
```

The contents of the *parameter-1* record is as follows:

return-code — contains a code posted by the TREAD routine upon completion. The return-codes and their meanings are explained in the *IBM 5280 Communications Reference Manual*, SC34-0247.

record-length — the length of the data read as posted by CAM.

read-type — for BSC only. Minus sign (-) indicates to read the entire block. Otherwise, a record will be read.

read-status — for BSC only. Specifies whether the read is conditional or not. Minus sign (-) indicates to return control immediately with status if data is not available for the read. Otherwise, the read will wait until data becomes available.

TWRT Routine

This routine is called to write a record from *buffer-2*.

The format of the statement to call the TWRT routine is:

CALL "AVCLWRT" USING *parameter-1*

parameter-1 is a data-item you code as follows:

```
01 parameter-1
   02 return-code        PIC 9999.
   02 record-length      PIC 9999.
   02 write-final        PIC X.
```

The contents of the *parameter-1* record is as follows:

return-code — contains a code posted by the TWRT routine upon completion. The return-codes and their meanings are explained in the *IBM 5280 Communications Reference Manual*, SC34-0247.

record-length — Specify the length of record to be written.

write-final — An F indicates the *write final* option. This is the last WRITE in a series. A *write final* causes the CAM to request positive response from the host that the entire series of WRITEs has been successful.

TCTL Routine

This routine permits the use of BSC line control, giving you more control over the data communications exchange.

The format of the statement to call the TCTL routine is:

```
CALL "AVCLCTL" USING parameter-1
```

parameter-1 is a data-item you code as follows:

```
01 parameter-1
  02 return-code        PIC 9999.
  02 control            PIC XXXX.
```

return-code — contains a code posted by the TCTL routine upon completion. The return-codes and their meanings are explained in the *IBM 5280 Communications Reference Manual*, SC34-0247.

control — the device control command. For SNA, the value 0007 is valid, indicating that a signal should be sent to the host. For BSC, the values in the following table are valid.

Value	Meaning
0100	Write Status
0300	End of Transmission
0400	Transmit RVI
0500	Transmit header (SOH-heading-STX)
0600	Transmit header (SOH-heading-ETB)
0700	Transmit header (SOH-heading-ITB)
0800	Transmit header (SOH-heading-STX-ETX)
0001	Set compression
0002	Reset compression
0003	Set transparent mode
0004	Reset transparent mode
0005	Set trace
0006	Reset trace

Appendix B. Status Key Return Codes

Table 1 summarizes the possible codes that can appear in the Status Key. The exact meanings differ according to the file organization (sequential, transaction, relative, or indexed) and the type of I/O verb executed (OPEN, READ, WRITE, etc.). See Tables 1 through 7 for the meanings according to file organization and I/O verb issued.

Table 1: Summary of Status Key Information

First Character of Status Key	Meaning	Second Character of Status Key	Meaning
0	Successful Completion	0	No further information
1	At End (no next logical record)	0	No further information
2	Invalid Key	0	No further information
		1	Sequence error
		2	Duplicate key; duplicate keys not allowed
		3	No record found
		4	Boundary violation (indexed or relative file)
3	Permanent error (data check, parity check, transmission error)	0	No further information
		4	Boundary violation (sequential file)
9	Other Errors	0	No further space (indexed file)
		2	Logic error
		3	Resource not available
		4	No current record pointer for sequential request
		5	Invalid or incomplete file information
		7	Data set shutdown (indexed file)

Relative, Sequential, and Transaction Data Set Organization

Table 2: OPEN Status Codes

Status Key	COBOL Detected Error
00	No error detected
30	See table 7 for detailed explanations.
92	The file is already open Attempt to open a file after previous open failed for non-resource reasons File is locked Second attempt to open a TRANSACTION I/O file.
93	Insufficient main storage for buffers See table 7 for additional causes.
95	COBOL detected invalid file characteristics: ASCII not configured in system; file is declared as ASCII. COBOL record size not equal to record size allocated on diskette by SYSLABEL. For relative files, a delete character was not specified when the data set was allocated. See Table 7 for additional causes.

Relative, Sequential, and Transaction Data Set Organization

Table 3: READ, WRITE, and REWRITE Status Codes

Status Key	COBOL Detected Error
00	No error detected
10	End of file sequential access
22	Duplicate key random access
23	No record found random access
24	Boundary error (relative file) possibly caused by an empty data set.
30	I/O Error See table 7 for additional causes.
34	Boundary error (sequential file)
92	Invalid request. For example: a WRITE or READ attempted on an unopened data set; a WRITE attempted on a file opened for input only. File is not open Second attempt to open a TRANSACTION I/O file Transaction I/O - 2 WRITES in SEQ Invalid sequence of READ/WRITE requests in TRANSACTION I/O. REWRITE, DELETE during sequential access not preceded by successful READ
93	See Table 7 for detailed explanations.
94	Current record pointer is undefined for the READ request (for sequential access mode only).
95	For relative files, a delete character was not specified when the data set was allocated.

Relative, Sequential, and Transaction Data Set Organization

Table 4: CLOSE Status Codes

Status Key	COBOL Detected Error
00	No error detected
30	I/O Error See Table 7 for additional causes.
92	File already closed. CLOSE UNIT issued to a multivolume data set. CLOSE UNIT issued to last volume of multivolume data set.
95	The sequence number of a volume in a multivolume data set is 99. See Table 7 for additional causes.

Indexed Data Set Organization

Table 5: OPEN Status Codes

Status Key	COBOL Detected Error
00	No error detected
30	I/O error. See table 7 for additional causes.
92	The file is already opened. Attempt to open a file after a previous open failed for reasons other than those given under code 93 below. File is locked. ACCESS IS RANDOM, OPEN mode is INPUT or I-O, and file is empty.
93	No space available for buffers. See Table 7 for additional causes.
95	COBOL detected invalid file characteristics: ASCII not configured in system; file is declared as ASCII. COBOL record size not equal to record size allocated on diskette by SYSLABEL. A delete character was not specified when the data set was allocated. Data set is multivolume. Length of index data set records is 4 bytes and ACCESS IS RANDOM; not allowed. See Chapter 5. Record length specified for the index data set is not valid. Must be equal to the length of the record key plus four(4). See Table 7 for additional causes.

Indexed Data Set Organization

Table 6: READ, WRITE, REWRITE, and DELETE Status Codes

Status Key	COBOL Detected Error
00	No error detected
10	End-of-file (for SEQUENTIAL ACCESS only).
21	Key out of sequence
22	Duplicate key
23	No record found
24	Boundary error End-of-extent
30	I/O Error See Table 7 for additional causes.
90	No space for insert
92	Key changed between READ, and a REWRITE or DELETE request (SEQUENTIAL ACCESS only). Invalid request such as READ when opened for OUTPUT REWRITE,DELETE during sequential processing not preceded by successful READ A WRITE request was made to a shared file when the OPEN mode was I/O (update). A delete character was not specified when the data set was allocated.
94	Current record pointer undefined for this READ request (for sequential access mode only).
97	Data set shut down

All File Organizations

Table 7: Detailed Explanations of Status Codes

Status Key	Cause	Reference in *Message Manual**
30	Device not ready after OPEN	3151
	Hardware failure	3201
	Memory overrun	3203
	Deleted sector	3204
	Bad track	3205
	Control address mark	3206
	Media check	3207
	Adaptor machine check	3208
	Defective sector	3209
	Erase mismatch off	3212
	Lost ready when busy	3251
	CRC error	3301
	Byte miscompare	3302
	ID mismatch	3303
	Head not positioned	3304, 3305
	Missing address marker	3306
	Initialization of track failed	3307
	Unexpected CTL address	3308
	Data set may contain defective sectors	3705
93	Data set marked NO SHARE	0115
	Data set marked NO SHARE	0116
	Data set in use (NO SHARE)	0727
	Device marked no share	0733
	Invalid open of unexpired data set	3234, 3410
	Write to protected data set	3422

Status Key	Cause	Reference in *Message Manual**
95	Missing label on first volume	3210
	Invalid diskette type	3213
	Write protect character error	3216
	Invalid record/block format	3217
	Invalid exchange type	3218
	Invalid offset to next space	3219
	Invalid physical record length	3220
	Invalid record attribute	3221
	Invalid record length	3222
	Invalid block length	3223
	Invalid blocking	3224
	Invalid extent	3225
	Invalid volume label format	3226
	Overlapped extents	3227
	Two data sets with same name	3228
	Invalid label—standard version	3231
	Invalid data set name	3232
	Invalid delete character	3233
	DSN must be simple name	3235
	Secure 'data set'	3242
	Invalid data header and/or parameter	3243
	Incorrect header label (not on Cylinder 0)	3244
	Invalid physical record sequence	3247
	Diskette requires ASCII translation table	3432
	No delete character	3433

*** See the *IBM 5280 Message Manual*, GA21-9354 under the message identifier in this column for a detailed explanation of the error and possible error procedures.**

Appendix C. Compiler Limitations

Alphabetic, alphanumeric items	length ≤ 32767 bytes
Alphanumeric edited, numeric edited items	length ≤ 127 bytes
Batch compilations	Max=10 COBOL programs
CALL USING	number of data-items ≤ 15
Data sets	≤ 12
GO TO DEPENDING	number of procedure names ≤ 15
IF statements	nesting depth ≤ 12
Message length for SEQUENTIAL I/O to Work Station	78 characters
Non-numeric literals	≤ 120 characters
Numeric, numeric edited items	≤ 18 digits
PERFORM N TIMES	1 ≤ N ≤ 32767
Printer line length	198 characters per line
PROCEDURE DIVISION USING	number of data-items ≤ 15
Structures	depth ≤ 20 level number ≤ 49
Table size	≤ 32767 bytes
Tables	dimensions ≤ 3

Appendix D. Compiler Return Codes

The following table shows the return-codes given by the compiler after execution, and their meanings:

Code	Meaning
0	No compiler error messages; only informational messages. A normal termination occurred.
4	Warning level messages only or an error was found in an option specification.
8	The compiler issued a conditional error message; execution of the compiled program *may* succeed.
12	A severe error was encountered; execution of the compiled program is doubtful.
16	A terminal error was encountered; no recovery is possible so the compilation was terminated.
20	An error occurred during the abnormal termination process so recovery is impossible.

There are times when the compiler is *not* in control of the circumstances under which it terminates, such as:

- When the system detects an error condition.
- When a compiler-issued instruction receives an unacceptable return-code from the system.

In the above two cases the return-code is formulated as follows:

- If the error is detected during compiler initialization, the return-code is 1000 added to the internal compiler error code (see the following list).
- If the error is detected during compiler termination, the return-code is 2000 added to the internal compiler error code.

Thus, the return-code not only indicates the particular error but also the location in the compiling process where the error was detected. For example, a return-code of 1010 means the failure, during compiler initialization, of an OPEN for the source data set.

The following table contains ABEND codes caused by an error in the data sets defined for your compile procedure. Any ABEND code received other than those listed is an internal compiler error and should be reported to your IBM representative.

ABEND Code	Meaning
8	Unsuccessful SYSLIB open (OS/VS only)
9	Printer open failed (OS/VS only)*
10	Source file open failed (OS/VS only)*
12	Work file open failed (OS/VS only)*
18	Buffer space not available (OS/VS only)* Increase available main storage.
20	Unsuccessful open for SYSLDOUT (OS*) or LDOUT (DOS). Possible reasons for DOS include: 1. Logical unit not assigned or assigned IGNORE 2. Device not ready, not operational or not DASD For OS, this ABEND can also occur if the record length of the data set does not agree with the record length determined by the DECK/NODECK option.
21	Unsuccessful open for SYSLDOVL (OS*) or LDOVLY (DOS). Possible reasons for DOS include: 1. Logical unit not assigned or assigned IGNORE 2. Device not ready, not operational or not DASD For OS, this ABEND can also occur if the record length of the data set does not agree with the record length determined by the DECK/NODECK option.
40	Insufficient GETVIS space for dictionary access table. At least 24K bytes required (DOS only). Increase size of partition.

* For OS/VS, the *ddname* is probably missing or misspelled. Correct the statement and resubmit the job.

Appendix E. Storage Estimates for OS/VS

Operating System Requirements

The OS/VS compiler operates under the control of the VS1 or MVS configurations of the Operating System.

The compiler requires 192K (196,608) bytes of main storage. This assumes minimal block sizes for those data sets where the block size is variable: SYSIN, SYSLIB, SYSPRINT, SYSLDOUT, and SYSLDOVL. Larger main storage may be required with larger blocks, and may result in improved performance.

Storage Estimates

The storage estimates given in this section are intended to supplement the system figures given in the individual Storage Estimates manuals for your specific operating system. Use these manuals to determine the amount of storage needed for the operating system. Then use the figures given here to determine the additional amounts needed for the IBM 5280 COBOL Compiler.

Estimates are given for the amount of auxiliary storage required for compiler data sets, and the amount of additional work space needed for compilation.

Auxiliary Storage Requirements for Compiler and Library Residence

The auxiliary storage required by the 5280 COBOL compiler and its permanent data set is as follows:

Data Set Name	Block Size	Number of Blocks
SYS1.AVCLOAD	6144	100

Work Space and Load Module Output Requirements

The COBOL compiler requires additional work space beyond the dynamic storage needed for execution. The work space needed varies with the number of source records. The figures below estimate the work space the compiler might require to process typical source programs. According to the type and combination of statements involved, however, the storage requirements may vary widely.

DDNAME	Block Size	Number of Blocks *
SYSUT1	512	86 - 360
SYSUT2	512	230 - 772
SYSUT3	512	145 - 572
SYSUT4	512	64 - 293
SYSUT5	512	174 - 578
SYSLDOUT	128	214 - 340

* The numbers given are for two specific programs. The smaller number is for a program with 1,186 source records (753 statements) and a load module size of 27K. The larger number is for a program with 2,773 records (2,600 statements) and a load module size of 43K.

If your program is segmented, you will have additional requirements for SYSLDOVL, which could even exceed those of SYSLDOUT, depending on the size and number of overlay segments.

Sample Program Requirements

If you run the installation verification procedure you will need space for the following data sets until the procedure is finished:

Data Set Name	Block Size	Number of Blocks
SYS1.AVCSAMP	1600	20
SYS1.AVCLDOUT	128	200

Execution-Time Considerations

The amount of 5280 storage must be sufficient to accommodate the load modules to be executed.

The input/output device requirements for execution of the problem program are determined from specifications made in the source program. See Chapter 8 for more information on 5280 storage requirements.

Appendix F. Storage Estimates for DOS/VSE

Operating System Requirements

The DOS/VSE compiler operates under the control of the DOS/VSE Operating System. The compiler requires 192K (196,608) bytes of main storage.

Storage Estimates

The storage estimates given in this section supplement the system figures given in the individual Storage Estimates manuals for your specific operating system. Use these manuals to determine the amount of storage needed for the operating system. Then use the figures given here to determine the additional amounts needed for the IBM 5280 COBOL Compiler.

Estimates are given for the amount of auxiliary storage required for compiler data sets, and the amount of additional work space needed for compilation.

Auxiliary Storage Requirements for Compiler and Library Residence

The auxiliary storage required by the 5280 COBOL compiler in the Core Image Library is: 660 blocks.

Work Space and Load Module Output Requirements

The COBOL compiler requires additional work space beyond the dynamic storage needed for execution. The work space needed varies with the number of source records. The figures below estimate the work space the compiler might require to process typical source programs. According to the type and combination of statements involved, however, the storage requirements may vary widely.

Filename	Block Size	Number of Blocks *
IJSYS01	512	86 - 360
IJSYS02	512	230 - 772
IJSYS03	512	145 - 572
IJSYS04	512	64 - 293
IJSYS05	512	174 - 578
LDOUT/LDOVLY	128	214 - 340

* The numbers given are for two specific programs. The smaller number is the number of blocks for a program with 1,186 source records (753 statements) and a load module size of 27K. The larger number is for a program with 2,773 records (2,600 statements) and a load module size of 43K.

If your program is segmented, you will have additional requirements for LDOVLY, which could even exceed those of LDOUT, depending on the size and number of overlay segments.

Execution-Time Considerations

The amount of 5280 storage must be sufficient to accommodate the load modules to be executed. The load module created is a complete 5280 partition, and its size is printed by the compiler.

The input/output device requirements for execution of the problem program are determined from specifications made in the source program. See Chapter 8 for more information on 5280 storage requirements.

Index

Fold and tape Please do not staple Fold and tape

NO POSTAGE
NECESSARY
IF MAILED
IN THE
UNITED STATES

BUSINESS REPLY MAIL

FIRST CLASS PERMIT NO. 40 ARMONK, NEW YORK

POSTAGE WILL BE PAID BY ADDRESSEE

International Business Machines Corporation
Information Systems Division
Dept 26X/037
2800 Sand Hill Road
Menlo Park, CA 94025

Fold and tape Please do not staple Fold and tape

IBM

International Business Machines Corporation

General Systems Division
4111 Northside Parkway N.W.
P.O. Box 2150
Atlanta, Georgia 30301
(U.S.A. only)

General Business Group/International
44 South Broadway
White Plains, New York 10601
(International)

IBM 5280 COBOL Programmer's Guide (File No. S5280-24) Printed in U.S.A. SL23-0032-1

SL23-0032-1

Printed in U.S.A.