COBOL85

User's Guide

DPS7000/XTA
NOVASCALE 7000

Languages: COBOL

REFERENCE
47 A2 06UL 06

DPS7000/XTA NOVASCALE 7000

COBOL85

User's Guide

Languages: COBOL

June 2002

BULL CEDOC
357 AVENUE PATTON
B.P.20845
49008 ANGERS CEDEX 01
FRANCE

REFERENCE
47 A2 06UL 06

Printed in France

Suggestions and criticisms concerning the form, content, and presentation of this book are invited. A form is provided at the end of this book for this purpose.

To order additional copies of this book or other Bull Technical Publications, you are invited to use the Ordering Form also provided at the end of this book.

Trademarks and Acknowledgements

Preface

Scope and Objectives

This manual provides information on both DPS 7000 COBOL 85 and the DPS 7000 system needed by a programmer to develop working COBOL 85 programs which will execute efficiently in a DPS 7000 system.

The descriptions in this manual use mainly the JCL form. To find the equivalent GCL commands, the reader is invited to consult the *IOF Terminal User's Reference Manual* that describes GCL in detail.

This manual complements the *COBOL 85 Reference Manual* that contains a formal specification of the COBOL 85 Programming Language.

Certain COBOL topics are not discussed in this manual because they are the subjects of separate manuals. These topics are:

- I/O using the UFAS file access system.
- Data base processing using IDS/II.
- Communications and transaction driven programming using TDS and VCAM.
- ORACLE using the PRO*COBOL pre-compiler (supported under IOF).

Intended Readers

Persons concerned with COBOL 85 DPS 7000 implementation: Job Control Language (JCL) and GCOS Command Language (GCL), utilities needed to input, compile, link, execute and debug COBOL 85 programs in a DPS 7000 environment:

- System engineers,
- COBOL programmers.

Prerequisites

The reader of the manual is assumed to be familiar with the COBOL language and with the basic functions of the GCOS operating system.

Structure

The first part (Chapters 1, 2, 3, and 4) describes the transformation of a coded COBOL source program into a working load module.

- Chapter 1 describes methods of introducing source programs into the system and maintaining source programs in a disk library.
- Chapters 2 and 3 describe the compilation and linkage of COBOL programs.
- Chapter 4 describes debugging techniques and gives hints on how to deal with abnormal program terminations.

The second part discusses some of the COBOL program's interfaces with the system; programming techniques are described which lead to efficient use of the system.

- Chapter 5 discusses the representation of data in memory.
- Chapter 6 discusses calling and called separately compiled programs.
- Chapter 7 discusses segmentation for Virtual Memory Management.
- Chapter 8 describes various programming techniques for reducing the size and increasing the execution speed of COBOL programs.
- Chapter 9 discusses various general aspects of file usage.
- Chapter 10 describes the standard record formats accepted by Data Management.
- Chapter 11 describes the use of unit record files.
- Chapter 12 contains a number of miscellaneous programming topics.
- Chapter 13 discusses the main improvements and changes brought by the COBOL 85 language and the COBOL 85 compiler.
- Appendix A contains sample COBOL and LINKER listings.
- Appendix B lists the diagnostic messages of the COBOL compiler.

The manual is completed with a general index of terms.

Bibliography

GCOS 7 Manuals

The following manual may be used as background material:

System Overview *47 A2 04UG*

The following manuals should be referred to in conjunction with the present manual:

System Calls From COBOL *47 A2 04UL*
COBOL 85 Reference Manual *47 A2 05UL*

JCL Reference Manual *47 A2 11UJ*
JCL User's Guide *47 A2 12UJ*

GCL Programmer's Manual (V7, V8, V9) *47 A2 36UJ*

Library Maintenance Reference Manual *47 A2 01UP*
Library Maintenance User's Guide *47 A2 02UP*

Messages and Return Codes Directory *47 A2 10UJ*

IOF Terminal User's Reference Manual Part 1 (V5) *47 A2 21UJ*
IOF Terminal User's Reference Manual Part 2 (V5-VBO)) *47 A2 22UJ*
IOF Terminal User's Reference Manual Part 2 (V5-FBO) *47 A2 23UJ*
IOF Terminal User's Reference Manual Part 3 (V5) *47 A2 24UJ*
IOF Terminal User's Reference Manual Part 4 (V5) *47 A2 25UJ*

IOF Terminal User's Reference Manual Part 1 (V6) *47 A2 31UJ*
IOF Terminal User's Reference Manual Part 2 (V6) *47 A2 32UJ*
IOF Terminal User's Reference Manual Part 3 (V6) *47 A2 33UJ*
IOF Terminal User's Reference Manual Part 4 (V6) *47 A2 34UJ*

IOF Terminal User's Reference Manual Part 1 (V7, V8, V9) *47 A2 38UJ*
IOF Terminal User's Reference Manual Part 2 (V7, V8, V9) *47 A2 39UJ*
IOF Terminal User's Reference Manual Part 3 (V7, V8, V9) *47 A2 40UJ*

Text Editor User's Guide *47 A2 05UP*
Full Screen Editor User's Guide *47 A2 06UP*

Catalog Management User's Guide *47 A2 35UF*

The following manuals will only be of interest to the COBOL programmer if he is using the facilities described therein:

UFAS-EXTENDED User's Guide *47 A2 04UF*
GAC-EXTENDED User's Guide *47 A2 12UF*

Data Management Utilities User's Guide *47 A2 26UF*

IDS/II User's Guide *47 A2 12UD*
Full IDS/II Reference Manual Volume 1 ..... *47 A2 05UD*
Full IDS/II Reference Manual Volume 2 ..... *47 A2 06UD*

Networks: Overview and Generation (V6) *47 A2 71UC*
Networks: Operations Reference Manual (V6) *47 A2 72UC*
Networks: DSAC User's Guide (V6) *47 A2 75UC*
Networks: AUPI User's Guide (V6) *47 A2 76UC*

Networks Overview (V7, V8, V9) *47 A2 92UC*
Networks Generation (V7, V8, V9) *47 A2 93UC*
Networks User's Guide (V7, V8, V9) *47 A2 94UC*

MCS User's Guide *47 A2 32UC*

TDS COBOL Programmer's Manual (V5) *47 A2 03UT*
TDS COBOL Programmer's Manual (V6) *47 A2 21UT*
TDS COBOL Programmer's Manual (V7, V8, V9) *47 A2 33UT*
TDS C Programmer's Guide *47 A2 07UT*

How to Deal with the Year 2000 ..... *47 A2 23UG*

Sort/Merge User's Guide *47 A2 08UF*

LINKER User's Guide *47 A2 10UP*

Program Checkout Facility User's Guide *47 A2 15UP*

IOF Programmer's Manual *47 A2 05UJ*

FORMS User's Guide (V6/V7) *47 A2 15UJ*

System Administrator's Manual (V5/V6) *47 A2 10US*
System Administrator's Manual (V7) *47 A2 41US*
System Administrator's Manual (V8/V9) *47 A2 54US*

ORACLE7 Installation Guide *47 A2 11UR*
ORACLE7 Guide to Processors and Utilities *47 A2 12UR*

Syntax Notation

The following notation conventions are used in this manual when describing the syntax of JCL/GCL and COBOL:

ITEM	An item in upper case is a literal value, to be specified as shown. The upper case is merely a convention; in practice you can specify the item in upper or lower case.
item	An item in lower case is a non-literal. A user-supplied value is expected. In most cases it gives the type and maximum length of the value: `char12` a string of up to 12 characters `name31` a name of up to 31 characters `dec10` decimal integer value of up to 10 digits `file78` a file description of up to 78 characters `volume18` a volume description, up to 18 characters
<u>ITEM</u>	An underlined item is a default value. It is the value assumed if none is specified.
bool	A Boolean value which is either 1 or 0. A Boolean parameter can be specified by its keyword alone, optionally prefixed by "N". Specifying the keyword alone always sets the value to 1. Prefixing the keyword with "N" always sets it to 0.
{ }	Braces indicate a choice of items. Only one of these items can be selected. When presented horizontally, a vertical bar separates the items as follows: `{ item | item | item }`
[ ]	Square brackets indicate that the enclosed item is optional. An item not enclosed in square brackets is mandatory.
( )	Parentheses indicate that a single value or a list of values can be specified. A list of values must be enclosed by parentheses, with each value separated by a comma or a space.

...	Ellipses indicate that the item concerned can be specified more than once.
+ = $ * / - .	Literal characters to be specified as shown.
- - - -	All parameters or commands below a dashed line do not appear in the help menus.

EXAMPLE 1:

```
[ VOLUME = { * | () | (vol18 ...) } ]
```

This means you can specify:

- Nothing at all (VOLUME=* applies)
- VOLUME=* (the same as nothing at all)
- VOLUME=FSD001:MS/D500 for a single volume
- VOLUME=(FSD001:MS/D500,FSD002:MS/D500) for a list of volumes
- VOLUME=() for no volumes

❑

EXAMPLE 2:

```
[ ACCNTSPACE = { [+]dec5 | -dec5 } ]
```

This means you can specify:

- Nothing at all
- ACCNTSPACE=10 to increase the value by 10
- ACCNTSPACE=+10 to increase the value by 10
- ACCNTSPACE=-10 to decrease the value by 10

❑

EXAMPLE 3:

```
[ AUTOADD ={ bool | 1 } ]
```

This is a Boolean parameter whose default value is one. You can specify:

- Nothing at all (AUTOADD=1 applies)
- AUTOADD=1 or simply AUTOADD
- AUTOADD=0 or simply NAUTOADD

❑

Delivery Conditions

Acknowledgement

This acknowledgement has been reproduced from the CODASYL COBOL Journal of Development, 1984 as requested in that publication, prepared and published by the CODASYL Programming Language Committee.

"Any organization interested in reproducing the COBOL report and specifications in whole or in part, using ideas from this report as the basis for an instruction manual or for any other purpose, is free to do so. However, all such organizations are requested to reproduce the following acknowledgement paragraphs in their entirety as part of the preface to any such publication. Any organization using a short passage from this document, such as in a book review, is requested to mention "COBOL" in acknowledgement of the source, but need not quote the acknowledgement.

COBOL is an industry language and is not the property of any company or group of companies, or of any organization or group of organizations.

No warranty, expressed or implied, is made by any contributor or by the CODASYL COBOL Committee as to the accuracy and functioning of the programming system and language. Moreover, no responsibility is assumed by any contributor, or by the committee, in connection therewith.

The authors and copyright holders of the copyrighted material used herein

FLOW-MATIC (trademark of Sperry Rand Corporation), Programming for the Univac (R) I and II, Data Automation Systems copyrighted 1958, 1959, by Sperry Rand Corporation; IBM Commercial Translator Form No. F 28-8013, copyrighted 1959 by IBM; FACT, DSI 27A5260-2760, copyrighted 1960 by Minneapolis-Honeywell,

have specifically authorized the use of this material in whole or in part, in the COBOL specifications. Such authorization extends to the reproduction and use of COBOL specifications in programming manuals or similar publications."

Table of Contents

1. Creating and Maintaining Source Programs

2. Compiling

3. Linking

4. Executing

5. Representation of Data

6. Calling and Called Programs

7. Segmentation

10. Standard Record Formats

11. Using Unit Record Files

12. Miscellaneous Topics

13. COBOL 85 Extensions and Changes

Table of Graphics

Figures

Tables

1. Creating and Maintaining Source Programs

The COBOL compiler accepts input from a sequential file (usually an input enclosure) or a library member. Input enclosures and library members are both subfiles, but input enclosures are handled by the user as sequential files. The compiler can also read input from other sequential files, e.g., tape files.

An input enclosure must be part of a batch job. A library member, however, may be created or updated during a batch job, or during an interactive step run via IOF (Interactive Operation Facility). If a library member is created, it may be updated later using LIBMAINT (Library Maintenance).

The use of input enclosures, libraries and files for source programs is discussed in these paragraphs.

1.1 Input Enclosures

The use of an input enclosure as direct input to the compiler is shown in this example.

```
$JOB ...
CBL SOURCE = *PROG1, CULIB = RES.CULIB;
$INPUT PROG1;
000100 IDENTIFICATION DIVISION.
000200 PROGRAM-ID. PROG1.
       .
       .
$ENDINPUT;
$ENDJOB;
```

In this example the input enclosure is held in SARF format (TYPE = DATA is the default option on the INPUT JCL statement). SARF format is described in Chapter 10.

It is recommended that where possible the same name be used throughout program development for the following:

- Input-enclosure-name.
- Program-name in the PROGRAM-ID paragraph of the Identification Division.
- Compile-unit-name (taken by the compiler from the program name).
- Load-module-name.

This minimizes any confusion that may arise from having several names for the same program at various stages of development. In the above example the program name and input-enclosure-name are both PROG1. The compiler will generate a compile unit of the same name as the program-name even if the input enclosure name is different.

1.2 Source Libraries

Using an input enclosure in the above manner means that the source program must be re-read from the input stream each time the compilation is executed. To avoid this the source program may be loaded into a library. The program may then be repeatedly updated and compiled without being re-read from the input stream. The use of libraries is discussed in these paragraphs (for more details refer to the *Library Maintenance User Guide*).

1.2.1 Creating a Library Member from the Input Stream

This example shows the use of the utilities LIBALLOC and MAINTAIN_LIBRARY (LIBMAINT) to create a library named SL.LIB containing source language members PROG1 and PROG2. An input enclosure is used containing MOVE commands and the source programs. This input enclosure is read by MAINTAIN_LIBRARY.

```
$JOB ...
LIBALLOC SL, (SL.LIB, SIZE = 5),MEMBERS = 100;
LIBMAINT SL, LIB = SL.LIB, COMFILE = *SLENC;
$INPUT SLENC;
MOVE COMFILE:PROG1,TYPE = COBOLX;
000100 IDENTIFICATION DIVISION.
000200 PROGRAM-ID. PROG1.
.
.
//EOD
MOVE COMFILE:PROG2, TYPE = COBOLX;
000100 IDENTIFICATION DIVISION.
000200 PROGRAM-ID. PROG2.
.
.
//EOD
$ENDINPUT;
$ENDJOB;
```

The LIBALLOC utility creates a library, SL.LIB, with a size of 5 cylinders (this utility need not be used if the library already exists). The MOVE commands create the two library members, PROG1 and PROG2, each containing one of the programs in the input enclosure.

The TYPE = COBOLX option in the MOVE commands indicates that the sequence number and card identifier fields will not be included in the library member text. The use of this option is discussed in a later paragraph.

The following paragraphs explain how to create a library member using the EDIT under IOF. Note that the EDIT command can be used in a similar way to create a library member from cards in a batch job. See the *Text Editor User's Guide* for details.

1.2.2 Creating a Library Member Interactively

IOF may be used to create a source language library member during an interactive session. IOF does not use input enclosures. Instead the source language is entered using EDIT or FSE (Full Screen Editor).

The following example illustrates the use of the EDIT command at an interactive terminal. Program PROG1 is entered via EDIT, and then written to an existing source library SL.LIB. Sequence numbers are generated in the SSF headers (RENUMBER) and the program is printed (PRINT). A detailed description of this process is given below:

```
S: MAINTAIN_LIBRARY SL, LIB = SL.LIB;
>>> 09:28 LIBMAINT 40.04 21
C: EDIT;
R: A
I: IDENTIFICATION DIVISION.
I: PROGRAM-ID. PROG1.
   .
   .
   .
I: /
R: W (CBX) PROG1
R: /
C: RENUMBER PROG1;
C: PRINT PROG1;
   10 IDENTIFICATION DIVISION.
   20 PROGRAM-ID. PROG1.
   :
C: /
<<< 09:30
S:
```

The prompt S: is output by the system. Following this prompt you can enter any system level GCL command. When the MAINTAIN_LIBRARY command is entered, you get a welcome banner (>>> etc.), containing the time, followed by the C: prompt which invites you to enter a MAINTAIN_LIBRARY (LIBMAINT) command. You then enter the EDIT command after which you get the R: prompt. This invites you to enter an EDIT request. You enter an append data request (A) after which you get the I: prompt. This invites you to enter input data until the escape character / is encountered. You then enter the source program. When you enter the / character, you again get the R: prompt and the system waits for another request. The response "W (CBX) PROG1" requests that the source program just entered are written to a library member named PROG1. The / request then terminates the EDIT session. You return to the C: prompt where you can enter a new command.

The (CBX) option used in the W request is equivalent to the TYPE = COBOLX option in the MOVE command discussed earlier. Using this option the line sequence numbers are not entered (but the indicator area is entered). They must be generated after input by using the command RENUMBER PROG1, as shown in the example. This command generates a sequence number in the SSF header of each record (see Chapter 10) but does not insert a sequence number into the COBOL text.

The new member is then printed using the PRINT PROG1 command, and LIBMAINT execution is terminated by a / command. LIBMAINT outputs the time (e.g. 09:30) immediately before terminating. The system then outputs an S: prompt and you may enter further GCL commands at the system level.

1.3 Updating the Source Member

A source program can be updated in a batch job using the LIBMAINT command UPDATE. This command allows you to insert, replace or delete specific lines that are identified by line sequence numbers. If more complex alterations are to be made to the member (e.g. search for a character string and replace by another string), you should use the EDIT command.

The UPDATE command is normally used with an input enclosure containing program updates. Since input enclosures are not used in IOF, the EDIT command should be used when updating source programs interactively (it may also be used in batch mode).

The FSE editor may also be used to update source programs.

Full descriptions of the use of EDIT, FSE, and the UPDATE command of LIBMAINT are given in the relevant manuals.

1.4 COBOL Reference Format

The COBOL reference format describes the line of COBOL text in terms of character positions in a record on an input medium. The ANS standard is shown in the table "COBOL Reference Format" below.

COBOL Reference Format

Characters	Used for
1 to 6	Sequence number area
7	Indicator area
8 to 11	Area A
12 to the end of the record	Area B

The length of area B depends upon the actual record length of the storage medium used for the program and whether the optional eight-character card identifier area is included. (The traditional card identifier area is not part of the ANS standard and is therefore not mentioned in the COBOL reference format). On all storage media except input enclosures the sequence number area can optionally be excluded from the record. The use of the sequence number area and the card identifier area can be controlled by specifying the language type of the COBOL program, and by using the CARDID or NCARDID parameters of the CBL GCL/JCL statement.

The use of the COBOL reference format with the language types DATA, COBOL, COBOLX, and DATASSF for punched cards, library member records and interactive terminal lines is discussed in these paragraphs.

NOTE:

The meaning of the term "record format" will be limited to that used in Chapter 10, Standard Record Formats. The term "text format" will be used in these paragraphs to refer to the format of the COBOL source text (with or without sequence number area or card identifier area) in a library member record.

Horizontal tabulation (T) and backspace (B) characters can be inserted in the COBOL source text. The effect of these characters is as follows.

T in columns 1-6: skip to column 8 (i.e., the next character in the source text is treated as if it were in column 8).

T in columns 7-11: skip to column 12.

B in columns 7-8: Backspace to column 7.

If a **T** occurs elsewhere in the source text, it is treated as a space.

FOR EXAMPLE:

```
            Col. 1   5       14
                 .   .        .
                 .   .        .
Input text   .    T01 A-NAMETPIC X.

            Col. 1   8       11      20
                 .   .       .        .
                 .   .       .        .
Treated as:           01     A-NAME PIC X.
```

A **T** beyond column 11 can be used to format output on a terminal.

❑

1.4.1 Input Enclosure Record Format

The lines of a program contained in an input enclosure are in one of the formats shown in the table "Punched Card Format" below (unless the CONTCHAR option is used in the $INPUT JCL statement - see below).

Punched Card Formats

With Card Identifier Area		Without Card Identifier Area	
Columns	Used for	Columns	Used for
1 to 6	Sequence number area	1 to 6	Sequence number area
7	Indicator area	7	Indicator area
8 to 11	Area A	8 to 11	Area A
12 to 72*	Area B	12 to 72*	Area B
73 to 80*	Card identifier area		

* When the source program is not on punched cards, if there is no card identifier area, Area B extends to the last characters in each record, else, the card identifier area is the last eight characters of each record.

The sequence number area must always be present. However, the card identifier area may be excluded, in which case area B extends to column 80.

The way in which a program is processed from an input enclosure is controlled by the TYPE parameter of the $INPUT JCL statement:

```
              { DATA    }
TYPE =        { COBOL   }
              { DATASSF }
```

The Stream Reader will copy each record in the input enclosure to a temporary subfile of the system file SYS.IN according to the rules shown in the table "Format of SYSIN Records" below. Subfiles in the system file SYS.IN are known as "SYSIN subfiles".

Format of SYSIN Records

TYPE parameter in $INPUT JCL statement	Format of record in SYSIN (see Chapter 10)	Columns copied to SYSIN
DATA	SARF	1 to 80
COBOL	SSF	7 to 80
DATASSF	SSF	1 to 80

If the input enclosure is read directly by the compiler (via SYSIN) the last eight columns will be ignored, unless the NCARDID option is specified (see Chapter 2). If however, the source program is moved to a library by the LIBMAINT utility, columns 73-80 should be removed from each record (see below), in which case the compiler will treat the last eight columns of the resulting record as COBOL text.

An input enclosure containing a source program to be moved to a library will contain LIBMAINT commands in addition to the source program. In order to preserve the first 6 columns of such commands, the TYPE parameter of the $INPUT JCL statement should not specify COBOL. It is recommended that the TYPE parameter be omitted from the $INPUT JCL statement, in which case a TYPE of DATA will be assumed.

The text format of library member records is discussed in these paragraphs. The effects of using DATA, COBOL and DATASSF in the TYPE, parameter of the $INPUT statement are summarized on the left-hand side in Figure 1-1.

The line formats shown above do not apply when the CONTCHAR option of the $INPUT JCL statement is used. This option requests the Stream Reader to concatenate the data held on several cards wherever a continuation character (-) is encountered as the last non-blank character on a card. When the option is used, area B extends throughout every continuation card up to column 80 of the last continuation card in a record (or column 72 if the card identifier area is present). The sequence number area, indicator area and area "A" only occur in the first card of a record. See the *JCL Reference Manual* for more details.

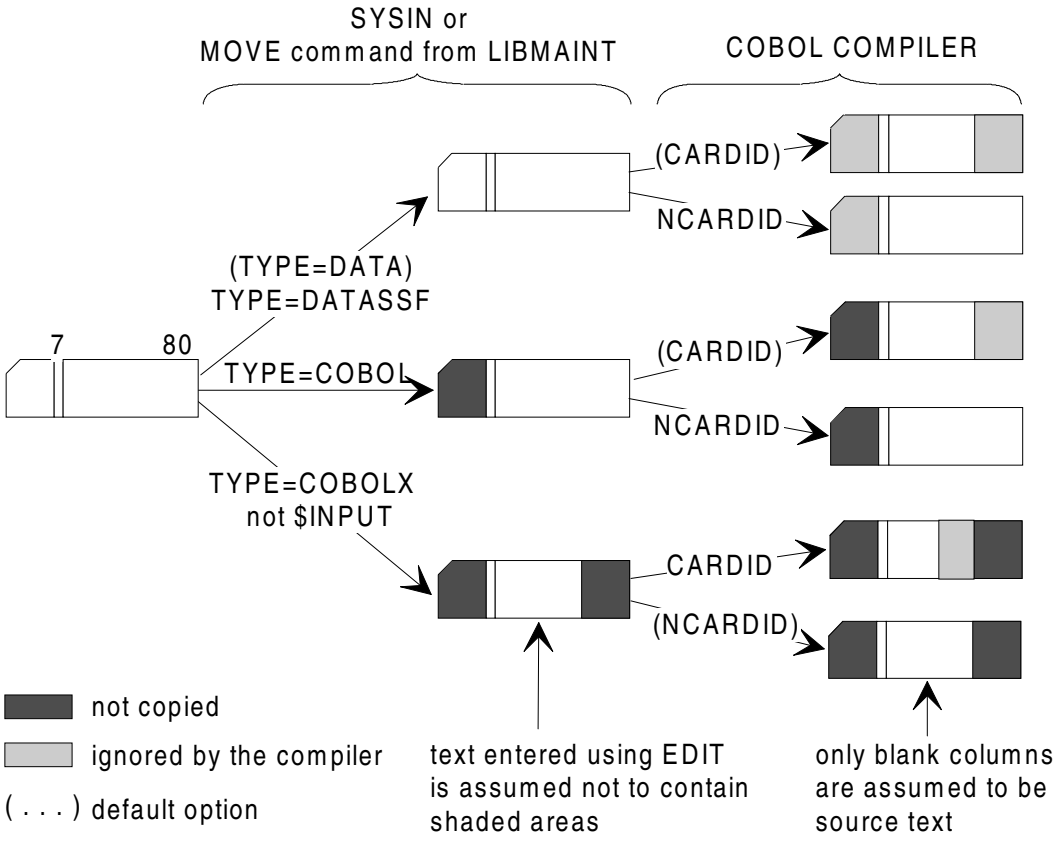


Figure 1-1. Effects of Using TYPE and CARDID Parameters

1.4.2 Library Member Text Format

COBOL text in a library member record is held in one of the formats shown in the table "Library Member Record Formats" below. (Library member records are variable length, therefore the last character position given for area B or the card identifier area is a maximum value.)

Library Member Record Formats

With sequence number area and card_id area (language type DATASSF)		With card_id area only (language type COBOL)		Neither area (language type COBOLX)	
Character positions	Used for	Character positions	Used for	Character positions	Used for
1 to 6	sequence number area	-	-	-	-
7	indicator area	1	indicator area	1	indicator area
8 to 11	Area A	2 to 5	Area A	2 to 5	Area A
From position 12 to 8 positions before the end of the record (i.e. area B can extend up to character position 247)	Area B	From position 6 to 8 positions before the end of the record (i.e. area B can extend up to character position 247)	Area B	From position 6 to the end of the record (i.e. area B can extend up to character position 255)	Area B
Last 8 characters	Card identifier	Last 8 characters	Card Identifier		

The text format of a library member is specified when the member is created by the TYPE parameter of the MOVE command or the W request of the EDIT command (see TYPE = COBOLX and W (CBX) PROG1 in the above examples). The values which can be specified in the MOVE command and W request and which are applicable to COBOL programs are shown in the table "Language Types of COBOL Programs" below.

Language Types of COBOL Programs

Type parameter of MOVE command (language type)	Type parameter of W Request
DATASSF	DAT
COBOL	COB
COBOLX	CBX

The effect of using these parameters is shown above and is summarized on the left-hand side in Figure 1-1.

It is recommended that the same language type be used for all COBOL library members at your installation. This avoids any confusion that might arise concerning the text format of programs that are to be updated. The recommended language type is COBOLX. It has these advantages:

- The sequence number area is removed from the COBOL text (it is stored in the SSF header). This means that you do not have to space over a redundant sequence number area when updating the program at an interactive terminal.
- The card-identifier area is removed. This area is redundant after the program has been stored in a library. The retention of a card-identifier area in library member text can lead to confusion when updating the program (e.g. when the SUBSTITUTE request of EDIT is used).
- All trailing spaces to the right of the COBOL text are suppressed in each library member record (this is not the case withDATASSF). This fact, together with the suppression of the sequence number area and card-identifier area (if relevant) results in a compact record that occupies a minimum of disk space.

If, as recommended, the COBOLX language type is used for the library member, the language type must not be specified in the $INPUT JCL statement of the input enclosure to be read by LIBMAINT (TYPE = DATA will be assumed). COBOLX must be specified in the TYPE parameter of the MOVE command or in the W or Z request of the EDIT command.

In the case of the EDIT command source lines (updates or original programs) should be entered in exactly the same text format as they are to be held in the library member record. That is, if a language type of COBOLX is being used, the first character in the line is the indicator area, and area "B" extends to the end of the line; the program or updated text should be written to the library member with a W or Z request which specifies language type CBX.

It is recommended that a sequence number be present if the source program is input from the stream reader. This number will be included in the SSF header when the program is moved to a library member by LIBMAINT, if the language type of the member is COBOL or COBOLX. The compiler, unless asked not to, will check that these numbers are in non-descending sequence and will report any descending sequences.

The programmer can also refer to sequence numbers in the SSF headers when updating a library member using the EDIT or UPDATE commands. If required, the source program can be given a set of sequence numbers on input by means of the NUMBER option of the MOVE command. However, if this is done the compiler will not be able to check the original sequence of the card deck (this should be done via the CHECK parameter of the MOVE command). An existing library member can be given a new set of sequence numbers at any time by means of the RENUMBER command.

1.4.3 Interactive Terminal Line Format

The format of a line of COBOL text entered under the EDIT command at an interactive terminal is the same as that shown in the table "Library Member Record Format" except that the size of area B is determined by the number of characters entered on one line (including continuation lines if the continuation character (-) is used).

1.5 Source Files

In addition to the library members mentioned above, you can store source programs on sequential files. These files can be generated using the CREATE or LIBMAINT utilities and can be read directly into the compiler by using the INFILE parameter of the COBOL JCL statement.

On the other hand the program may already exist on a sequential file. For example, the program may have been written to a sequential file by a program generator, or the program may have been dumped from a library to magnetic tape at a different installation for compilation at your installation. These files can normally be read directly into the compiler using the INFILE parameter.

2. Compiling

This chapter describes the use of the COBOL compiler.

The compilation of a COBOL program is requested by the CBL JCL statement or by the CBL GCL command.

The syntax is very similar though, in GCL, the user may be menu-driven and ask for help texts.

Not to be confusing, we will restrict ourselves here to JCL form. The equivalent GCL will be found in the *IOF Terminal User's Reference Manual*.

2.1 Job Control Language

The extended JCL statement CBL is used to execute the COBOL compiler. The compiler will normally generate a compile unit and listing. The compile unit can be stored in a temporary or in a permanent library. The linking and execution of the program must be requested by the user in subsequent job steps. The listing can also be stored in a temporary or permanent library or file.

The figure below shows the format of the CBL statement. Note that the parameters, which are underlined in this figure, are the default values assumed by the compiler when no alternative parameter is chosen. For example, if the NCKSEQ parameter is not specified in the CBL statement the CKSEQ parameter will be assumed by the compiler. Default parameters are therefore redundant in the CBL statement. However, they may be used in conjunction with the COMFILE parameter when serial compilation of a set of source programs is being carried out. In such a case they may be used to override the parameters in the CBL statement. See "The Alter Facility", below.

CBL Statement Format

```
CBL
{ SOURCE = *input-enclosure-name                          }
{     comfile = (sequential-input-file-description)       }
{ SOURCE = member-name                                    }
{     [{LIB = (output-library-description)  }]            }
{     [{INLIB  = (input-library-description)}]            }
{     [{INLIB1 = (input-library-description)}]            }
{     [{INLIB2 = (input-library-description)}]            }
{     [{INLIB3 = (input-library-description)}]            }
{     [COMFILE = (sequential-input-file-description)]     }
{                                                         }
{ SOURCE = (member-name, member-name ... )                }
{     [{LIB = (output-library-description)  }]            }
{     [{INLIB  = (input-library-description)}]            }
{     [{INLIB1 = (input-library-description)}]            }
{     [{INLIB2 = (input-library-description)}]            }
{     [{INLIB3 = (input-library-description)}]            }
{                                                         }
{ SOURCE = (star-name, star-name ... )                    }
{     {LIB = (output-library-description)  }              }
{     {INLIB  = (input-library-description)}              }
{     {INLIB1 = (input-library-description)}              }
{     {INLIB2 = (input-library-description)}              }
{     {INLIB3 = (input-library-description)}              }
{                                                         }
{ INFILE = (sequential-input-file-description)            }
{     [COMFILE = (sequential-input-file-description)]     }
{                                                         }
{ COMFILE = (sequential-input-file-description)           }
{     [INLIB = (input-library-description)]               }

[ COVLIB = ( output-library-description ) ]

[ DDLIB1 = ( input-library-description ) ]
[ DDLIB2 = ( input-library-description ) ]
[ DDLIB3 = ( input-library-description ) ]

[ CULIB = ( output-library-description ) ]

[ DICLIB = ( output-library-description ) ]

[{ PRTFILE = ( print-file-description )   }]
[{ PRTLIB = ( print-library-description ) }]
```

```
[        { NSTD                                                    }]
[        { ANSI                                                    }]
[        {{HIGH        }                                           }]
[        {{H           } [{-DBG }] [{-RW }] [{-COM }] [{-SEG }]}]
[LEVEL = {{INTERMEDIATE} [{-DBG1}] [{    }] [{-COM1}] [{-SEG1}]}]
[        {{I           } [{-DBG2}] [{-RW1}] [{-COM2}] [{-SEG2}]}]
[        {{MINIMUM     }                                           }]
[        {{M           }                                           }]

[        { OBJA  } ]
[ CODE = {       } ]
[        { OBJCD } ]

[ DSEGMAX = digits4[K]]
[ PSEGMAX = digits4[K]]
[ ISEGMAX = (digits4[K] digits3 [digits4[K]]) ]

[                  [{NS  }] ]
[ TEMP = [digits2] [{    }] ]
[                  [{BIN }] ]

[{ CARDID }]    [{  CASEQ }]    [{  CKSEQ }]    [{ CKUKFIL }]   [{COBOL85 }]
[{NCARDID }]    [{        }]    [{        }]    [{         }]   [{        }]
[{DCARDID }]    [{ NCASEQ }]    [{ NCKSEQ }]    [{ NCKUKFIL }]  [{COBOL74 }]

[{ CODAPND}]    [{ DCLXREF}]    [{ DDLIST }]    [{  DEBUG }]
[{        }]    [{BDCLXREF}]    [{        }]    [{        }]
[{NCODAPND}]    [{NDCLXREF}]    [{NDDLIST }]    [{ NDEBUG }]

[{ DEBUGMD}]    [{DIAGIN  }]    [{  DMAP  }]    [{ EXPSIZE}]
[{NDEBUGMD}]    [{DIAGAFT }]    [{        }]    [{        }]
[{DDEBUGMD}]    [{DIAGBEF }]    [{ NDMAP  }]    [{NEXPSIZE}]

[{ LFATAL }]    [{   LIST }]    [{  MAP   }]    [{  OBJ   }]
[{        }]    [{  NLIST }]    [{        }]    [{        }]
[{ LOBSERV}]    [{ NCLIST }]    [{ NMAP   }]    [{ NOBJ   }]
                [{CMTLIST }]

[{ OBSERV }]    [{OOBSERV }]    [{  PMAP  }]    [{ REFMDCK}]
[{NOBSERV }]    [{        }]    [{        }]    [{        }]
[{ OBSBEF }]    [{OFATAL  }]    [{ NPMAP  }]    [{NREFMDCK}]
[{ OBSAFT }]

[{ SILENT }]    [{  SUBCK }]    [{ SUBOPT }]    [{ WARN   }]
[{        }]    [{        }]    [{        }]    [{ NWARN  }]
[{NSILENT }]    [{ NSUBCK }]    [{  NOPT  }]    [{ WARNBEF}]
                                                [{ WARNAFT}]

[{  XLN   }]    [{  XREF  }]
[{        }]    [{ BXREF  }]
```

```
[{  ILN   }]    [{ NXREF  }]

[ SIZEOPT = (size-parameters) ]
[ STEPOPT = (step-parameters) ]

[         { (sequential-output-file-description) } ]
[ WORK1 = {                                       } ]
[         { (work-file-description )              } ]

[         { (sequential-output-file-description) } ]
[ WORK2 = {                                       } ]
[         { (work-file-description )              } ]

[ WORK3 = (work-file-description ) ] ;
```

As the JCL CBL statement is an extended JCL statement it must not appear inside a step enclosure. The following example illustrates the use of this statement:

```
$JOB...
LIBALLOC CU, (CU.LIB, SIZE = 5), MEMBERS = 100;
CBL SOURCE = *PROG1, CULIB = CU.LIB;
$INPUT PROG1;
000100 IDENTIFICATION DIVISION.
000200 PROGRAM-ID. PROG1.
       .
       .
$ENDINPUT;
$ENDJOB;
```

The LIBALLOC CU statement is used to create a library, CU.LIB, with a size of 5 cylinders. Normally, the library already exists and this utility need not be used. The compiler will read the source program from the input enclosure PROG1 (via SYSIN) and will store the compile unit in the compile unit library CU.LIB.

The following paragraphs describe the parameters that may be used in the CBL statement. Note that the following symbolic names used in the figure "CBL Statement Format" refer to standard parameter groups which are described in the *JCL Reference Manual*:

```
sequential-input-file-description
sequential-output-file-description
input-library-description
output-library-description
print-file-description
print-library-description
work-file-description
```

See the *JCL Reference Manual* for all information concerning these standard parameter groups.

2.1.1 SOURCE, INFILE, COMFILE, LIB, INLIB and INLIBn Parameters

These parameters are used to specify the name and location of the program or programs to be compiled. COMFILE is also used for specifying modifications to the source program. See "The Alter Facility", below (note that this has no connection with the COBOL ALTER statement). A series of programs may be compiled during a single execution of the compiler. See "Serial Compilation", below.

At least one of the parameters SOURCE, INFILE and COMFILE must be specified in a CBL statement. All of the remaining parameters are optional. SOURCE may appear in the same CBL statement as COMFILE, and INFILE may appear in the same statement as COMFILE. However, SOURCE and INFILE may not appear in the same statement.

The simplest use of these parameters occurs when the source program is held in an input enclosure. In this instance the following statement will suffice:

```
CBL SOURCE = *input-enclosure-name;
```

where input-enclosure-name is the name of an input enclosure contained in the same job.

If the source program is held in a library, the name of the member and the name of the library are both specified in the CBL statement as follows:

```
CBL SOURCE = member-name LIB = (output-library-description);
```

or

```
CBL SOURCE = member-name INLIB = (input-library-description);
```

However, one or more libraries can also be specified in a separate LIB JCL statement as follows:

```
LIB SL  INLIB1  =  (input-library-description)
        [INLIB2  =  (input-library-description)
        [INLIB3  =  (input-library-description)]];

CBL SOURCE = member-name;
```

The LIB JCL statement defines a "search path" for the compiler. The compiler will search for the source program specified by member-name first in the INLIB1 library, then in the INLIB2 library and finally in the INLIB3 library. The first member found will be compiled; any others of the same name will be ignored. Note that the LIB JCL statements shown in this chapter do not contain all possible parameters. See the *Library Maintenance Reference Manual* for further details.

If source programs of the same name occur in more than one of the libraries included in the LIB JCL statement, the library to be used can be specified by the INLIBn parameter of the CBL statement. In this case the normal search path is overridden by the INLIBn parameter. The statement format is as follows:

```
LIB SL  INLIB1  =  (input-library-description)
       [INLIB2  =  (input-library-description)
       [INLIB3  =  (input-library-description)]];

                                { inlib1 }
CBL SOURCE = member-name;       { inlib2 }
                                { inlib3 }
```

The three methods of specifying a member-name and library described above may also be used when a series of source programs is to be compiled in a single execution of the compiler. In this case the SOURCE parameter must specify a series of member-names.

EXAMPLE:

```
CBL SOURCE  =  (member-name [, member-name]... ),
    LIB     =  (output-library-description);
```

or

```
CBL SOURCE  =  (member-name [, member-name]... ),
    INLIB   =  (input-library-description);
```

The chosen search path (given by the LIB or INLIB parameter, or by the LIB JCL statement) may be modified for individual source programs by using the COMFILE parameter. The COMFILE parameter specifies an input enclosure, library member or sequential file containing commands that control the compilation of a series of source programs. The use of the COMFILE parameter is described in detail in "The Alter Facility", below.

❑

The following JCL statements illustrate the use of COMFILE:

```
CBL SOURCE  =  member-name,
    LIB     =  (input-library-description),
    COMFILE =  *input-enclosure-name;

$INPUT input-enclosure-name;
.
.
COMFILE commands
.
.
$ENDINPUT;
```

Source programs may also be read from a sequential file on disk or magnetic tape (this may, for example, be a tape file written by the LIBMAINT utility using the OUTFILE option). The INFILE parameter is used for this purpose as follows:

```
CBL INFILE = (sequential-input-file-description);
```

The file specified in the INFILE parameter can contain one or several source programs. The COMFILE parameter can be used together with the INFILE parameter to specify which source programs in the file are to be compiled.

As an alternative to specifying a list of member names in the SOURCE parameter a range of member names can be specified using the "star convention" (same as the star convention used by the LIBMAINT utility). The following statement format is used:

```
LIB SL  INLIB1  =  (input-library-description)
       [INLIB2  =  (input-library-description)
       [INLIB3  =  (input-library-description)]];

CBL SOURCE = (star-name, star-name . . . )
{lib = (output-library-description)     }
{inlib = (input-library-description)    }
{inlib1                                 } ;
{inlib2                                 }
{inlib3                                 }
```

Note that if the library to be used is not specified in the CBL statement (i.e., no library search is carried out), INLIB1 is assumed to be specified in the CBL JCL statement. The LIB JCL statement can be omitted if an input-library-description is included in the LIB or INLIB parameter. Using the star convention all the library member names in the specified library having certain common characteristics can be excluded from compilation. Conversely, all names having certain common characteristics can be selected for compilation. The star convention works in exactly the same way as the LIBMAINT star convention. For a description of the star convention, see the *Library Maintenance Reference Manual*. However when the FROM= and/or the TO= phrases are used, the star-name including these phrases must be enclosed between apostrophes. The COMFILE parameter cannot be used if the star convention is used. The parentheses in the SOURCE parameter are necessary only when there are more than one star-name, or when the star-name begins with an asterisk.

Both the SOURCE and INFILE parameters can be excluded from the CBL statement. If this is done the COMFILE parameter must be used in conjunction with an input enclosure or sequential file containing commands which specify the members to be compiled. COMFILE is discussed in "The Alter Facility", below.

2.1.2 CARDID, NCARDID and DCARDID Parameters

These parameters control the treatment of the last eight character positions of COBOL text in each record of the source program. CARDID causes the compiler to ignore the last eight character positions (i.e., they are treated as a card identifier area). NCARDID causes the compiler to treat the last eight character positions as COBOL text (i.e., no card identifier area exists, area B extends to the end of the record).

If DCARDID is specified, or if none of the card identifier parameters is specified, the compiler assumes the following default values:

- CARDID where the language type is COBOL or DATASSF.
- NCARDID where the language type is COBOLX.

The effects of using CARDID, NCARDID or DCARDID with the language types DATASSF, COBOL and COBOLX is shown in the table "The Effects of Using CARDID, NCARDID, and DCARDID" below and is summarized in the right-hand side of Figure 1-1.

Table 2-1. The Effects of Using CARDID, NCARDID and DCARDID

Language type of member	Effect of using CARDID	Effect of using NCARDID	Effect of specifying DCARDID or no parameter
DATASSF or COBOL	The compiler ignores the rightmost eight character positions.	The compiler treats the rightmost eight character positions as COBOL text.	Same as CARDID
COBOLX	The compiler ignores the rightmost eight character positions. WARNING: This will result in incorrect compilation because the card identifier area has already been removed by the LIBMAINT utility.	The compiler treats the rightmost eight character positions as COBOL text. The Card identifier area has already been removed by the LIBMAINT utility.	Same as NCARDID

Note that CARDID and NCARDID do not apply to text copied via the COPY statement. The treatment of the last eight character positions of such text depends entirely upon the language type of the copied text (i.e., DCARDID is assumed).

The CARDID, NCARDID and DCARDID parameters should normally be omitted from the CBL statement. CARDID and NCARDID should be used only to compensate for any errors in the loading or the updating of a source library member. These errors will become apparent if the compiler fails to process the end of a source line. Such errors can occur when storing a library member (e.g. using the EDIT Z request) if the original language type is not used. Such errors should be corrected by LIBMAINT.

2.1.3 CASEQ and NCASEQ Parameters

CASEQ requests that all lower-case letters, which are not included in a non-numeric literal, be processed as if they were upper-case letters (default parameter).

NCASEQ requests that lower-case letters be different from upper-case letters, except for the words whose spelling is the same as that of a reserved word (in other words, reserved words may be written in lower-case or upper-case letters, or both). Thus the user-word "abC" is different from the user-word "aBC", whereas "move" is the same reserved word as "MOVE".

Except when used as currency symbols, the lower-case letters a,b,c,d,e,l,p,r,s,v,x and z in picture character strings are always taken to be their corresponding upper-case letters, irrespective of whether the NCASEQ parameter is used. The remaining lower-case letters are never processed in picture character strings as upper-case letters, even when the CASEQ parameter is used.

2.1.4 CKSEQ and NCKSEQ Parameters

The NCKSEQ parameter requests the compiler not to carry out any sequence check on the input source lines. If the CKSEQ parameter is specified, the compiler will check that the line numbers are in non-descending sequence (default parameter) if the source program is in SSF format or in ascending sequence otherwise. The check is done on the line number in the SSF header if the program is in SSF format (TYPE = COBOL, COBOLX or DATASSF) or on the line number in the source line if the program is in SARF format.

2.1.5 CKUKFIL and NCKUKFIL Parameters

The NCKUKFIL parameter requests the compiler not to emit, in the prolog of a program which contains one (or several) FD clause(s), some usual code which would cause the program to abort at Run time under TDS if a file is not described in STDS subfile. There is No ChecK about such UnKnown FILes: they may be absent from the STDS subfile, and from the JCL of the TDS.

Two restrictions exist when using the NCKUKFIL option:

- the PROGRAM-ID must not contain the INITIAL clause (Severity 3 error)
- the program (TPR) must not OPEN/CLOSE itself any file (Severity 2 error, fatal at Run time only if an attempt to READ/WRITE a file occurs after CLOSE and reOPEN of this file)

The default option is CKUKFIL : all COBOL files MUST be described in STDS subfile.

2.1.6 COBOL85 and COBOL74 Parameters

The COBOL85 parameter specifies that the source programs are written in the language described in the DPS 7 *COBOL 85 Reference Manual* and must be compiled as such.

The COBOL74 parameter specifies that the source programs are written in the language described in the DPS 7 *COBOL 74 Reference Manual* and must be compiled as such.

2.1.7 CODAPND and NCODAPND Parameters

CODAPND requests that, an attempt be made to put the last code segment of priority 0 (or the only code segment, if there is only one) in the same segment as the linkage segment. NCODAPND means that the linkage segment will contain no generated code (default parameter). Note that small, single segment programs make the most efficient Transaction Processing Routines (see the TDS manuals) as this reduced the amount of disk activity required for program loading.

2.1.8 CODE Parameter

The CODE parameter specifies the class of the target computer for which code will be generated. The different classes are:

- Class A: DPS 7/x5/x07 and 64/DPS
- Class C: DPS 7/x0, x17, x27, DPS 7000/x0, x2
- Class D: DPS 7/1017, 1027, 1x07, DPS 7000/4xx, DPS 7000/8xx.

If CODE=OBJA is used, the program can be run on a class A or C computer.

If CODE=OBJCD is used, the program can be run on a class C or D computer.

When the COBOL compiler runs under GCOS 7-V3A, the default value is OBJA.

When the COBOL compiler runs from GCOS 7-V3B, the default value is OBJCD.

A program compiled with CODE=OBJA is not executable on a class D computer if floating-point data are processed.

A program compiled with CODE=OBJCD and executed on a class A computer may stop with the exception: "ILLEGAL FIELD INSTRUCTION", when processing floating point data (USAGE COM-9, COMP-10 or COMP-15).

For program compiled with code=OBJA, running in a class D Hardware or vice versa, which does not contain floating point data but contains exponentiation, the executing speed may be reduced by a factor 100.

The LIST command of the LIBMAINT CU processor may be used to get information on the compatibility class of a compile unit. It must be interpreted as follows:

CU class	A-C compatible	C-D compatible
0 or none	Yes	yes
1	Yes	no
2	No	yes
3	No	no

2.1.9 COVLIB Parameter

The COVLIB parameter requires the use of the test coverage facility and specifies the library to be used to store the resulting statistics.

The test coverage facility is a new GCOS 7 COBOL 85 feature which gives a means to build sets of tests to verify newly developed programs, following E. Miller's method.

The test coverage facility is described in the Chapter 13 of the present manual.

2.1.10 CULIB Parameter

The CULIB parameter specifies the library in which the resulting compile unit is to be stored.

If a library other than TEMP is specified, it must have been allocated previously by, for example, the LIBALLOC utility, unless the output-library-description specified in the CULIB parameter contains the SIZE parameter (see the *Library Maintenance Reference Manual*). If TEMP is specified, the compile unit will be written as a member of a temporary library. The member-name given to the compile-unit will be the same as the program-name in the PROGRAM-ID paragraph of the source program.

If the CULIB parameter is omitted this is equivalent to CULIB = TEMP.

When linking compile units produced with no CULIB parameter, or with CULIB = TEMP, the compile unit library TEMP should be present in the library search path that precedes the LINKER statement (e.g., LIB CU, INLIB1=TEMP, INLIB2=...). However, if TEMP is the only input compile unit library, no LIB CU is required to define the search path.

2.1.11 DCLXREF, BDCLXREF and NDCLXREF Parameters

The DCLXREF or BDCLXREF parameter produces a cross-reference listing in declaration order. When DCLXREF is specified, all names declared in the compiled source appear in the cross-reference listing; when BDCLXREF is used, only those, which are referenced in the compiled source, are present. The format of this listing is described in "Cross-Reference Listing (Declaration Order)", below.

NDCLXREF means that no such cross-reference listing is required (default parameter).

2.1.12 DDLIB1, DDLIB2 and DDLIB3 Parameters

DDLIB1, DDLIB2 and DDLIB3 libraries are used in this order to find the subfile whose name is the schema name or the sub-schema name specified in the DB entry of the Sub-Schema Section. The search is done in the order specified. The first subfile found must be an IDS2 schema or sub-schema compiled by DDLPROC.

2.1.13 DDLIST and NDDLIST Parameters

The DDLIST parameter specifies that a listing of the records specified by the Sub-Schema Section is to be produced. The Data Base Record Listing, if produced, includes all specified Data Base Records with their subordinate items.

NDDLIST means that no such Data Base Records Listing is required (default parameter).

2.1.14 DEBUG and NDEBUG Parameters

The DEBUG parameter requests the compiler to build a table of all the source names in the program with an indication of name type (data-name, paragraph-name etc.) and the generated segment addresses. This table is stored in the compile unit. The program may, after linking, be executed under the control of the Program Checkout Facility. See Chapter 4.

NDEBUG is the default parameter assumed if DEBUG is not specified.

If DEBUG is not specified the Program Checkout Facility may only be used with effective addresses.

2.1.15 DEBUGMD, NDEBUGMD and DDEBUGMD Parameters

DEBUGMD means that the compilation is done as if the WITH DEBUGGING MODE clause were present in the Environment Division, even though it is absent.

NDEBUGMD means that the compilation is done as if the WITH DEBUGGING MODE clause were absent in the Environment Division even though it is present.

DDEBUGMD means that the presence or absence of the WITH DEBUGGING MODE clause in the Environment Division is meaningful. That is, it operates as specified in the *COBOL 85 Reference Manual* (default parameter).

2.1.16 DIAGIN, DIAGAFT and DIAGBEF Parameters

DIAGIN specifies that all error messages are embedded in the Alter and Source Listings (default parameter). However, warnings and/or observations may be presented before or after the source listing depending on the presence of the parameters WARNBEF, WARNAFT, OBSBEF and OBSAFT in the CBL JCL statement.

DIAGAFT and DIAGBEF specify that alter error messages are embedded in the Alter Listing, but that other (purely COBOL) error messages are listed after or before, respectively, the Source listing. DIAGAFT and DIAGBEF override WARNAFT, WARNBEF, OBSAFT and OBSBEF.

2.1.17 DICLIB Parameter

This parameter is only valid when COBOL74 is active.

The DICLIB parameter specifies the library in which the information required for the direct input to the Data Dictionary is to be stored.

2.1.18 DMAP and NDMAP Parameters

The DMAP parameter cannot be used if MAP is also used.

The DMAP parameter produces a data map and procedure definition listing. The format of these listings is specified in "Map Listings and Cross-Reference Listings", below.

NDMAP means that no such listings are required (default option).

2.1.19 DSEGMAX and PSEGMAX Parameters

These parameters permit the user to specify (in units of 1024 bytes) the preferred maximum size of data and procedure segments in the object program. If these parameters are not specified, the maximum segment size is that specified (in bytes) in the MAXIMUM DATA SEGMENT SIZE and/or MAXIMUM PROCEDURE SEGMENT SIZE phrases in the OBJECT-COMPUTER paragraph of the Environment Division. These phrases are not part of the ANS standard. If neither is specified, the compiler assumes a default of 4K bytes (K = 1024). The use of DSEGMAX and PSEGMAX is discussed in detail in Chapter 7, Segmentation. The maximum value for PSEGMAX is 64K, and the maximum value for DSEGMAX is 4096K.

2.1.20 EXPSIZE and NEXPSIZE Parameters

EXPSIZE indicates that the compiler is permitted to increase the working set used during part of the compilation. NEXPSIZE indicates that the working set must not be increased. The "working set" is the amount of main memory allocated to the compiler at the start of compilation.

EXPSIZE and NEXPSIZE are obsolete and should no longer be used.

2.1.21 ISEGMAX Parameter

This parameter permits the user to direct INITIAL program data allocation in the stack segment. It specifies the maximum size (in units of 1024 bytes) of the first area, the maximum number of additional areas (default is zero) and the maximum size (in units of 1024 bytes) of any additional area (default is 16K). If this parameter is not specified, the maximum number of stack areas and their maximum sizes (in bytes) are those specified in the MAXIMUM INITIAL DATA SEGMENT SIZE clause of the OBJECT-COMPUTER paragraph of the Environment Division. This clause is not part of the ANS standards. If neither is specified, the compiler assumes a default of 1 area of 64K (K = 1024). Stack space will be asked for at object time for each area separately using the AGT instruction. The maximum value for the size of the stack area is 4096K.

In addition to ISEGMAX parameter, the stack size must be specified in the LINKER command by the STACK3 command.

The maximum number of additional areas and the maximum sizes of these areas are not currently significant. The STACK3 command being of the form STACK3=(INITSIZE=n) where n is the maximum size that the stack may reach dynamically.

2.1.22 LEVEL Parameter

The LEVEL parameter specifies that the compilation level is DPS 7 (NSTD), Full ANSI 85 Standard (ANSI), high level subset (HIGH or H), intermediate level subset (INTERMEDIATE or I) or minimum level subset (MINIMUM or M), and includes level 1 of Debug (-DBG1), full Debug (-DBG or -DBG2), Report Writer (-RW or RW1), level 1 of communication (-COM1), full Communication (-COM or -COM2), level 1 of Segmentation (-SEG1), full Segmentation (-SEG or -SEG2). All features beyond the specified level are flagged as fatal errors (****), unless the LOBSERV parameter is used, in which case they are flagged as observations (*). If there are such flagged features, and the LOBSERV parameter is not used, no object code is then produced. The LEVEL parameter is not accepted if the level specified is above the maximum level specified for the installation. Unless modified by Field Engineering, the default level is ANSI 85, and the maximum level for the installation is NSTD.

The COBOL facilities, which are available in each level of the compiler, are listed in an appendix of the *COBOL 85 Reference Manual*.

2.1.23 LFATAL and LOBSERV Parameters

LOBSERV requests that, when COBOL features are used which are beyond the specified compilation level, the features be flagged as observations rather than fatal errors. The default is LFATAL (fatal errors). See "LEVEL Parameter" above.

2.1.24 LIST, NLIST, CMTLIST and NCLIST Parameters

NLIST specifies that the source program listing is not to be produced. However, unless DIAGAFT or DIAGBEF has been specified, the lines for which an error message is to be produced will be printed. NCLIST means the same as NLIST but only applies to lines included in the source program as the result of a COBOL COPY statement. LIST means that the complete program will be listed, including copied lines (default parameter). CMTLIST means that the listing of the source program restricted to comment lines will be produced.

2.1.25 MAP and NMAP Parameters

The MAP parameter cannot be used if DMAP or PMAP are also used.

The MAP parameter produces a data map and procedure definition listing (unless one of the cross-reference listings has been requested), a procedure map listing and a perform/alter bucket listing. The format of these listings is specified in "Map Listings and Cross-Reference Listings", below.

NMAP means that no such listings are required (default option). Note that the Cross-reference Listing produced by the DCLXREF parameter contains all of the information in the Data Map Listing. The XREF parameter produces the same information in alphabetic order.

2.1.26 OBJ and NOBJ Parameters

The compiler normally generates a compile unit in the library specified in the CULIB parameter (or, by default, in a temporary library). If NOBJ is used, no compile unit is output. The summary page printed at the end of the compilation listing indicates whether a compile unit has been produced.

OBJ is the parameter assumed if NOBJ is not specified.

2.1.27 OBSERV, OBSBEF, OBSAFT and NOBSERV Parameters

The NOBSERV parameter suppresses all observation diagnostic messages in the program listing. However, the number of observation messages and the line numbers on which they occur are printed in the compilation summary page and in the JOR. If errors of this type are detected by the compiler and neither warning, nor serious, nor fatal errors are found, the JCL status value will be set to 100 (SEV1) at the end of the compilation. See "JCL Status", below.

OBSERV means that observations must appear in the listing. The positions of the observations depend on which of DIAGIN, DIAGBEF and DIAGAFT is active. If DIAGIN is active, the positions of the observations depend on WARNBEF or WARNAFT, if specified. OBSERV is the default parameter.

OBSBEF and OBSAFT specify that purely COBOL observations (not alter messages) are listed before or after, respectively, the Source listing; OBSBEF and OBSAFT are overridden by DIAGBEF and DIAGAFT or, if neither DIAGBEF nor DIAGAFT is specified, by WARNBEF and WARNAFT.

OBSERV, OBSBEF and OBSAFT are overridden by NWARN.

2.1.28 OFATAL and OOBSERV Parameters

OFATAL requests that obsolete COBOL features be flagged as fatal errors rather than observations. The default is OOBSERV (observations).

2.1.29 PMAP and NPMAP Parameters

The PMAP parameter cannot be used if MAP is also used.

The PMAP parameter produces a procedure map listing and a perform/alter bucket listing. The format of these listings is specified in "Map Listings and Cross-Reference Listings", below.

NPMAP means that no such listings are required (default option).

2.1.30 REFMDCK and NREFMDCK Parameters

The REFMDCK parameter indicates that reference modifications must be protected against access outside the referenced data item boundaries. The default value for this parameter is NREFMDCK (protection is not provided). If REFMDCK is specified, when an attempt is made to access a data item beyond its boundaries through a reference modification, the run unit is aborted with the return code "F0001807->USER 0,LNERR".

2.1.31 PRTFILE Parameter

This parameter requests that the compilation listing be appended to a permanent SYSOUT file for printing or processing at a later stage by, for example, the WRITER utility or any text handling program or utility. Otherwise, the listing is printed at the end of the job and no permanent copy is kept. For example, the user could specify output to tape:

```
$JOB...
CBL
  SOURCE = *COBSOURCE,
  CULIB = CU.LIB,
  MAP,
  XREF,
  PRTFILE = (COBFILE, DEVCLASS = MT/T9/D1600, MEDIA = ATAPE);
.
.
```

In this case, only the JOR will be printed at the end of job execution. (Note that the JOR is unaffected by the PRTFILE parameter.)

If the PRTFILE parameter is used, the compiler adds the program listing to the SYSOUT file in append mode. The PRTLIB parameter, on the other hand, replaces any previous listing of the same name (see below).

When serial compilation occurs, and the PRTFILE parameter is used, all listings are stored in a single file.

If the SYSOUT file is full, the compilation terminates with the following message in the JOR:

```
CBL01.ERROR WHILE COMPILING program-id. LISTING FILE EXHAUSTED
```

The size of the file should be increased and the compilation should be started again.

2.1.32 PRTLIB Parameter

This parameter is similar to PRTFILE except that the listing will be stored in a member of the library specified in the PRTLIB parameter. If several programs are compiled in series when the PRTLIB parameter is used, the listing for each program will be stored in a separate library member. Each library member will be given the program-name specified in the PROGRAM-ID paragraph of the source program, suffixed by "_L". It replaces any member of the same name. If the library is not large enough to contain the listing, error message CBL01 is printed in the JOR. See "PRTFILE Parameter", above.

2.1.33 SILENT and NSILENT Parameters

The SILENT parameter specifies that, when compiling in an interactive environment under IOF, nothing will be printed at the terminal, except the first and last banners, the source identification, the fatal (****) errors (compiler errors, compiler restrictions,...), and the compilation summary. The source lines with the corresponding errors will be printed on the listing only.

NSILENT means that the errors will be printed on the terminal with the corresponding erroneous lines. Only the first 20 lines with the most serious errors are listed on the terminal (default parameter).

2.1.34 SIZEOPT Parameter

The SIZEOPT parameter group can be used to specify one or more of the parameters included in the SIZE JCL statement (see the *JCL Reference Manual*).

This parameter is obsolete and unused now.

2.1.35 STEPOPT Parameter

The STEPOPT parameter group can be used to specify one or more of the parameters included in the STEP JCL statement (see the *JCL Reference Manual*). However, the following must not be included in the STEPOPT parameter group for COBOL:

- load-module-name;
- TEMP, SYS or input-library-description;
- the ALL option of the DUMP parameter;
- the OPTIONS parameter.

2.1.36 SUBCK and NSUBCK Parameters

The NSUBCK parameter is used to indicate that array bound protection is not required. Array bound protection is a mechanism in the system that traps any attempt to access a data item beyond the upper or lower limits of the table which contains the data item. The default value for this parameter is SUBCK (protection is provided). If an attempt is made to access a table data item beyond the bounds of the table when array bound protection is provided, an exception 17-02 will occur.

The NSUBCK parameter is also used to by-pass the checking of the boundaries of a variable length data-item.

The NSUBCK parameter should be used with care. The removal of array bound protection may produce unpredictable results in the event of an accidental boundary violation. This can make it very difficult to find the program error that caused the boundary violation.

The SUBCK or NSUBCK parameter can be used only if LEVEL=NSTD is also used.

2.1.37 SUBOPT and NOPT Parameters

The SUBOPT parameter requests the compiler to optimize subscripted and indexed references to reduce the time taken to execute such references. Note that under certain circumstances, optimization will result in the removal of array bound protection. The NOPT parameter requests no optimization (default parameter). See Chapter 8, "Efficiency", for more information on the use of SUBOPT.

2.1.38 TEMP Parameter

The TEMP parameter permits the user to specify the maximum number of significant digit positions to be kept in intermediate results of arithmetic expressions. When NS is specified, the number of significant digits is not always guaranteed when faster computations may be achieved. When BIN is specified, intermediate results are in binary floating point format.

If this parameter is not specified, the TEMP clause of the Default Section of the Control Division applies (this clause is not part of the ANS Standard: the TEMP clause or the TEMP parameter can be used only if the LEVEL=NSTD parameter is used). If neither is specified, the compiler assumes a default of 18 significant digits, unless the compilation level is NSTD, in which case 30 is assumed.

2.1.39 WARN, WARNBEF, WARNAFT and NWARN Parameters

The NWARN parameter suppresses all warning and observation diagnostic messages in the program listing. However, if errors of this type are detected by the compiler, the number of each type of error and the line numbers on which they occur are printed in the compilation summary page and in the JOR, and the severity value is set to SEV1 or SEV2, unless serious or fatal errors are found (see " JCL Status" below).

WARN is the default parameter. It specifies that warnings and observations (if NOBSERV is not present) are listed. The position of the warnings and observations depends on which of DIAGIN, DIAGBEF or DIAGAFT is active. If DIAGIN is active, the position of observations depends on OBSBEF or OBSAFT if specified.

WARNBEF and WARNAFT specify that purely COBOL warnings and observations (if NOBSERV is not specified) are listed before or after, respectively, the Source listing. WARNBEF and WARNAFT override OBSBEF and OBSAFT, they are overridden by DIAGBEF and DIAGAFT.

Note that warning messages can be important because some of these messages will become fatal in the next release. Such warnings are immediately followed by the message "WILL BE FATAL (***) NEXT RELEASE" and are marked with a severity 5 in the list of error messages (Appendix B).

2.1.40 WORK1, WORK2 and WORK3 Parameters

The compiler performs approximately one input and one output operation for each source line compiled. These I/O operations fall into the following categories.

- 20% refer to missing compiler segments in backing store (system disks)
- 25% refer to libraries (source, compile unit, listing)
- 55% refer to work files in backing store (system disks)

It can be seen that 75% of all I/O operations occur on system disks. The WORKn parameters can be used to reduce the I/O activity on the system disks. This can be done either by requesting that the work files be placed on other disks or by increasing the block sizes of the work files.

Using the WORKn parameters also reduces the risks of backing store saturation that may arise from a high level of multi-programming or from the compilation of very large programs. When the backing store cannot accommodate the default work files, a fatal error message is printed:

```
* * * * 9-56 BACKING STORE IS FULL. USE WORK FILES FOR LARGE PROGRAMS
```

In such cases it may be advisable either to reduce the total machine load, or to use temporary work files, reserved for the compilation by the WORKn parameters.

If the WORKn parameters are to be used, it is recommended that only WORK1 and WORK2 be used and that WORK3 be left on backing store.

If the values specified for RECSIZE and BLKSIZE in the WORKn parameters are larger than the default sizes, the number of I/O operations during compilation will be reduced. However, this will increase the memory required for I/O buffers and will thus reduce the effective size of the working set available to the compiler. This will, in turn, slow the compilation of larger programs and increase the number of I/O operations during compilation phases that are especially dependent upon available memory.

WORK1 and WORK2 each require a capacity of up to 150 bytes per source line; WORK3 requires up to 300 bytes per source line.

Example of the use of temporary work files:

```
CBL
  WORK1  =  (MYFILE1, FILESTAT = TEMPRY, SIZE = 20)
  WORK2  =  (MYFILE2, FILESTAT = TEMPRY, SIZE = 20)
  SOURCE =  MYPROGRAM;
```

In this example, the compiler will use MYFILE1 and MYFILE2 that will each be allocated temporarily to the job with a size of 20 cylinders on a RESIDENT disk pack. If this size is not large enough for the compilation, for any of the files, it will be increased automatically by units of 1 cylinder.

The SIZE parameter is not mandatory in this example. In fact, for temporary files, a default value of 4 cylinders is taken, and files that happen to be full are incremented by units of 1 cylinder.

The advantage of specifying a SIZE is that the compilation will not be started if space is not available on the RESIDENT disk pack to contain the work files. Failing the specification of SIZE, the compilation will start with 4 cylinders for each file, and a shortage of available compilation space could be dared. The user can also request that the work files be put on another disk pack by specifying DEVCLASS and MEDIA in the WORKn parameter. In that case, if SIZE is not specified, the compilation will start with one cylinder for each file. If the work files specified are too small a fatal error message is printed:

```
* * * 9-55 WORKn IS FULL
```

Permanent work files can be used for WORK1, WORK2 and/or WORK3 if desired. They should be preallocated, for instance on a VBO volume, in the following way.

```
PREALLOC  external-file-name

          BFAS = (LINKQD = (TYPE = NONE
                            BLKSIZE = digits5
                            RECSIZE = digits5
                            RECFORM = FB
                            NDLREC))

          FILESTAT = CAT
          DEVCLASS = device-class
          GLOBAL   = (MEDIA = volume-list
                      SIZE = digits5);
```

Standard values for both BLKSIZE and RECSIZE are 6434 for WORK1 and WORK2, and 2046 for WORK3. Larger values may be used to further reduce the i-o activity. The DEVCLASS, MEDIA and SIZE parameters are explained in the *Data Management Utilities User's Guide* under the PREALLOC utility. The above example preallocates a cataloged file, but uncataloged files may also be used.

The use of permanent work files enables to save the cost of dynamic formatting. It has another advantage: only one compilation using a given set of work files can be active at a given time when SHARE=NO is specified in the WORKn parameter; this is a way of queuing compilations submitted from an IOF terminal. The SHARE parameter is explained in the *JCL Reference Manual*.

The WORK1 and WORK2 work files can be sequential output files. If sequential files are used, they must have been preallocated in the same way as permanent work files are allocated, i.e. as described above, except that the BFAS parameter of the PREALLOC JCL statement must be: "BFAS = (SEQ = (...))", all other parameters being identical (Refer to the *Data Management Utilities User's Guide*). When using sequential files for WORK1 and/or WORK2 the same files must not be used by compilations running concurrently; thus, if the same sequential files are used for several compilations, SHARE=NO must be specified in the WORK1 or WORK2 parameters to prevent two compilations from running at the same time.

2.1.41 XREF, BXREF and NXREF Parameters

The XREF or BXREF parameter produces a cross-reference listing in alphabetic order. When XREF is specified, all names declared in the compiled source appear in the cross-reference listing; when BXREF is specified, only those that are referenced in the compiled source are present. The format of this listing is described in "Cross Reference Listing (Alphabetic Order)", below. NXREF means that no such cross-reference listing is required (default parameter).

2.1.42 XLN and ILN Parameters

ILN specifies that all source line numbers that appear in the compilation listings (i.e. in cross-references, errors,...) refer to internal line numbers. When ILN is specified, the internal line number is present, followed by the external line number on each source line in the program listings. XLN specifies that all source line numbers that appear in the compilation listings refer to external line numbers. When XLN is specified, the internal line numbers appear in the program listing only if the DEBUG parameter is specified or if the WITH DEBUGGING MODE is used or if DML (Data Manipulation Language of IDS/II) is used. XLN is the default parameter for SSF source programs; ILN is the default parameter for non-SSF source programs. See "Program Listing" below.

2.1.43 JCL Status

At the end of the compilation, subsequent job processing may be determined by testing with the JUMP JCL statement a severity value set by the compiler or by the system. Severity values are printed in the JOR. The possible values are shown in the table below.

Table 2-2. Severity Values Set by the Compiler

Severity Value	Status	Flag	Meaning
SEV0	0		no error
SEV1	100	*	observation
SEV2	1000	**	warning
SEV3	10000	***	serious error
SEV4	20000	****	fatal error
SEV5	50000		Compilation killed by operator (TJ)
SEV6	60000		Abort requested by system (exception)

The following example shows how the severity value may be tested to decide whether to link a program that has just been compiled:

```
$JOB...
CBL
SOURCE = *COMPST;
JUMP ABNOR, SEV,GT,2;
LINKER
COMPST
COMFILE = *LKCOB;
ABNOR:...
.
.
```

See "User JCL Status", Chapter 12, for more details.

2.1.44 Libraries Referred to in the COPY Statement

Text to be included in a COBOL program via the COPY statement is stored as a normal library member. The COPY statement must specify the name of the library member in which the text is stored. As a result of a COPY statement the entire contents of the specified library member will be included in the program.

When compiling a program, which includes a COPY statement, the library from which the text is to be copied must be specified either in the CBL JCL statement or in an earlier LIB JCL statement. For example:

```
LIB SL, INLIB1 = MY_LIBRARY;
CBL SOURCE = MY_MEMBER;
```

CAUTION:
The LIB JCL statement must not be confused with the LIB parameter of the CBL JCL statement; the LIB JCL statement specifies a search path, whereas the LIB (or INLIB) parameter in the CBL JCL statement specifies the only library in which the main source text is searched for; in addition, the latter is also included in the search path for COPY statements, as explained below.

If the LIB statement is not used, then the CBL statement must include a LIB=(output-library-description) or INLIB=(input-library-description) parameter in addition to a SOURCE or COMFILE parameter. For example:

```
CBL SOURCE = MY_MEMBER, INLIB = MY_LIBRARY;
```

The COPY statement may contain an optional OF/IN LIB or OF/IN INLIB.

If the LIB JCL statement is used, and the LIB=(output-library-description) or INLIB=(input-library-description) parameter is not included in the CBL statement, the COPY statement may optionally contain OF/IN INLIB1/INLIB2/INLIB3. In this case the text to be copied is in the library specified in the LIB JCL statement. For example:

```
COBOL:
    COPY MY-TEXT OF INLIB2.

JCL:
    LIB SL INLIB1 = MY_LIBRARY_A
           INLIB2 = MY_LIBRARY_B
           INLIB3 = MY_LIBRARY_C;
```

In this example MY-TEXT is copied from library MY_LIBRARY_B. The other libraries specified in the LIB statement could also contain a member called MY-TEXT, but these libraries would be ignored.

If the OF/IN INLIBn option is omitted the compiler will search the libraries specified in the LIB JCL statement. INLIB1 will be searched first, then INLIB2 and then INLIB3.

If the LIB JCL statement is used, and the LIB=(output-library-description) or INLIB=(input-library-description) parameter is included in the CBL JCL statement, the COPY statement may optionally contain OF/IN LIB or INLIB, or OF/IN INLIB1/INLIB2/INLIB3. In this case the text will be copied from the specified library of the CBL or LIB JCL statement. If the OF/IN option is omitted, the compiler will search first in the library specified as LIB (or INLIB) in the CBL statement and will then search in the library specified as INLIB1 in the LIB JCL statement followed by INLIB2 and INLIB3.

For example:

```
COBOL:
    COPY MY-TEXT.
    COPY OTHER-TEXT OF LIB.
    COPY NEXT-TEXT OF INLIB2.

JCL:
    LIB SL INLIB1 = MY_LIBRARY_A
           INLIB2 = MY_LIBRARY_B
           INLIB3 = MY_LIBRARY_C;
    CBL SOURCE = MY_MEMBER, LIB = MY_LIBRARY;
```

In this example, MY-TEXT is searched for in MY_LIBRARY, MY_LIBRARY_A, MY_LIBRARY_B and MY_LIBRARY_C in that order. OTHER-TEXT is searched for in MY_LIBRARY and NEXT-TEXT is searched for in MY_LIBRARY_B.

Alternatively, an actual library name can be specified in the OF/IN clause. This library name must also be specified in the library description of the LIB or INLIB parameter of the CBL JCL statement or in the INLIBn parameters of the LIB JCL statement. The libraries whose names would be LIB, INLIB, INLIB1, INLIB2 or INLIB3 may not be referenced by their actual name.

Note that the LIB, INLIB, or INLIBn parameter in the CBL JCL statement and the R LIB/INLIB/INLIBn request in conjunction with the COMFILE parameter of the CBL statement do not affect the search path used for copied text. See "The Alter Facility", below, for details of the R request.

2.2 The Alter Facility

The alter facility allows the user to compile modified source programs without actually modifying the source library, file or input enclosure. The alter facility should be distinguished from the COBOL ALTER statement. The alter facility is in no way connected with the ALTER statement, and the two should not be confused.

The modifications to be applied to a program are specified in an input enclosure, library member or sequential file known as a "command file". The command file comprises the following:

- a COMPILE command
- editor requests

For example, one could submit a compilation of MY_PROGRAM where the lines 12 and 16 would be deleted, without changing the source library. The submission deck could be as follows:

```
LIB SL, INLIB1 = (MY_LIBRARY,...);
CBL COMFILE = *MY_ALTER;
$INPUT MY_ALTER;
COMPILE;
R MY_PROGRAM
12D
16D
Q
$ENDINPUT;
```

The command file for a batch compilation is normally specified in the COMFILE parameter of the CBL JCL statement. However, if there is no COMFILE parameter, and the first line of source starts with the word COMPILE (possibly preceded by at most 6 blanks), the file is considered to be the command file. Note that a command file must be in SARF format, or in SSF format with a language type DATASSF. The command file for an interactive compilation can be entered directly to the compiler via the IOF terminal. In this case, neither COMFILE nor SOURCE nor INFILE must be specified in the CBL JCL statement. See "Interactive compilation", below.

The COMPILE command consists of the word "COMPILE", optional CBL JCL parameters and a mandatory semi-colon (";"). The command may occupy more than one line but a parameter may not be split between two lines.

The allowed parameters are as follows:

CARDID	NCARDID	DCARDID	
CASEQ	NCASEQ		
CKSEQ	NCKSEQ		
COBOL85	COBOL74		
CODAPND	NCODAPND		
DCLXREF	NDCLXREF	BDCLXREF	
DDLIST	NDDLIST		
DEBUG	NDEBUG		
DEBUGMD	NDEBUGMD	DDEBUGMD	
DIAGIN	DIAGAFT	DIAGBEF	
DMAP	NDMAP		
EXPSIZE	NEXPSIZE		
LFATAL	LOBSERV		
LIST	NLIST	NCLIST	CMTLIST
MAP	NMAP		
OBJ	NOBJ		
OBSERV	NOBSERV	OBSBEF	OBSAFT
OOBSERV	OFATAL		
PMAP	NPMAP		
REFMDCK	NREFMDCK		
SILENT	NSILENT		
SUBCK	NSUBCK		
SUBOPT	NOPT		
WARN	NWARN	WARNBEF	WARNAFT
XLN	ILN		
XREF	NXREF	BXREF	

These parameters override default as well as explicit JCL parameters.

The COMPILE command may also contain the SKIP parameter, in which case no other parameter is permitted. This parameter means that the next program in the source file or member is not to be compiled. See " Serial Compilation", below.

Permissible editor requests are a subset of the standard EDIT requests, namely:

A	append
C	change
D	delete
I	insert
N	no request
Q	or / quit
R	read member
S	substitute
"	comment

A part from Q or /, and R, and apart from the format of the input lines in case of interactive compilation, the above editor requests work in exactly the same way as in the EDIT command of LIBMAINT. The differences for Q and R are explained below. The differences concerning the format of input lines are explained under "Interactive Compilation" below.

The Q or / request is not mandatory for the COBOL compiler when running a batch compilation. It can then be omitted, but the programmer may wish to include the Q or / request so that the command file can later be input to LIBMAINT without modification.

The R request identifies the program to be compiled. Its format is as follows:

```
   [ [ { LIB    } ]                ]
   [ [ { INLIB  } ]                ]
 R [ [ { INLIB1 } ] : member-name ]
   [ [ { INLIB2 } ]                ]
   [ [ { INLIB3 } ]                ]
```

This request is different from the R request of the LIBMAINT EDIT command. The R request, when used, must be the first request after a COMPILE command, and there cannot be any other such request before the next Q (or /) request, or before the next COMPILE command, if any.

INLIB1, 2 or 3 specifies that the member whose name is member-name, is to be searched for in the library specified in the LIB JCL statement under the INLIB1, 2 or 3 parameters respectively. LIB or INLIB are synonymous; they mean that the member is to be searched for in the library specified in the LIB=(output-library-description) or INLIB=(input-library-description) parameter of the CBL JCL statement, whichever is used. In the absence of these keywords, member-name is searched for first in the LIB or INLIB library specified in the CBL JCL statement, then in the libraries specified in the LIB JCL statement, according to the implied searching rules. If member-name is absent, the program to be compiled is the next in the current member or file, or the first in the specified member or file when it is the first "R" request. See "Serial Compilation", below.

It is also possible to specify the source member to be processed in the COMPILE command. Thus:

```
COMPILE member-name [parameter]...;
```

is equivalent to the sequence:

```
COMPILE [parameter]...;
R [member-name]...;
Q
```

Member-name must immediately follow the word COMPILE and it cannot be any of the allowed parameters. No request will be asked for the compilation of the (first) program in member-name.

To prevent the first program in a source member from being compiled, the user must specify the member-name followed by the parameter SKIP in the COMPILE command. For example, the following command and request sequence will skip the first program and compile the second program in MEMBER-1:

```
COMPILE MEMBER-1 SKIP;
COMPILE;
R
Q
```

The address forms which may be used in editor requests are:

```
regular expression
^       (first line)
$       (last line)
.       (current line)
SSF line number
```

possibly modified by an expression of the form:

```
+ relative-number-of-lines
```

Address ranges with the addresses separated by commas or semi-colons are allowed. Compound addresses are allowed except in address ranges.

Addresses must be given in such an order that they refer to successive lines of the source. They cannot refer to lines inserted as the result of an A, C or I request.

When successive programs of a member (see Serial Compilation, below) are referred to, the "^" address refers to the first line of the first program.

Upper and lower-case letters are equivalent in the COMPILE command and as request identifiers.

2.3 Serial Compilation

The compiler can compile a series of programs during a single execution. Two levels of "seriality" are available.

The lower level is as follows. There may be several programs in a single member, or in a file. They must each be terminated by a line containing the character string "END COBOL" only, somewhere in area A or B (the last "END COBOL" line in a member or file is not mandatory). The "END COBOL" line may be omitted after a program which terminates with an END PROGRAM header. Each program is compiled in its turn.

The upper level may be at the level of the SOURCE parameter of the CBL JCL statement. This parameter may specify more than one member. In that case, all programs contained in the first specified member are compiled, then, all programs contained in the second specified member, and so on.

The upper level may also be specified in the COMPILE command and any associated R request in the command file. Thus:

```
CBL SOURCE = (MEMBER-1, MEMBER-2...)...
```

could also be written:

```
CBL COMFILE = *ALTER...
$INPUT ALTER;
COMPILE;
R MEMBER-1
COMPILE;
R MEMBER-2
$ENDINPUT;
```

Note that SOURCE or INFILE can be used in the same CBL JCL statement as COMFILE.

The command file is necessary if the source program is to be modified before compilation. For example:

```
$INPUT ALTER;
COMPILE;
R MEMBER-1
^,$S/MOVE/ADD/
COMPILE;
R MEMBER-2
"NO MODIFICATION APPLIED TO MEMBER-2
$ENDINPUT;
```

If MEMBER-1 contains more than one program, say 3, only the first one will be compiled with the above command file. If all of them are to be compiled, the command file should be as follows:

```
$INPUT ALTER;
COMPILE;
R MEMBER-1
COMPILE;
R
COMPILE;
R
COMPILE;
R MEMBER-2
$ENDINPUT;
```

If the second program of MEMBER-1 was to be skipped (not compiled) the command file becomes:

```
$INPUT ALTER;
COMPILE;
R MEMBER-1
COMPILE SKIP;
COMPILE;
R
COMPILE;
R MEMBER-2
$ENDINPUT;
```

Obviously, when the R request does not specify a member-name, or when the SKIP parameter is used with the COMPILE command, there must be another program in the current member. This means that the name of the current member must be established by an "R" request (as shown in the previous examples).

However the name of the first member may also be specified in the CBL JCL statement:

```
CBL SOURCE = MEMBER-1, COMFILE = *ALTER...
$INPUT ALTER;
COMPILE;
R
"COMPILE THE FIRST PROGRAM OF MEMBER-1
COMPILE;
R
.
.
```

In the same way, a source file may also be specified:

```
CBL INFILE = (SOURCE,...), COMFILE = *ALTER...
$INPUT ALTER;
COMPILE;
R
"COMPILE THE FIRST PROGRAM OF SOURCE
COMPILE SKIP;
COMPILE;
R
"COMPILE THE THIRD PROGRAM OF SOURCE
.
.
```

When a command file is specified, the serial compilation is controlled by the contents of that command file. Therefore, the possible SOURCE parameter of the CBL JCL statement must not specify serial compilation.

The MOVE function of LIBMAINT can store more than one subfile in a sequential file. This sequential file may then be submitted to the compiler. The contents of each subfile will be handled as if it was separated from the next subfile by END COBOL. Each subfile can itself contain programs separated and/or terminated by END PROGRAM headers or by END COBOL.

A serial compilation can be restarted, if necessary, simply by changing the JCL to exclude those programs which have already been compiled successfully. The compiler will not restart in the middle of a partially compiled program.

2.4 Interactive Compilation

The compiler can be executed from an IOF terminal. The normal CBL JCL statement can be used for this purpose. Serial compilation can be specified in the normal way and the alter facility can be used. The compiler executes as in batch mode with the following exceptions.

The parameters COMFILE, SOURCE and INFILE can all be omitted from the CBL JCL statement. In this case, the command file must be input directly to the compiler via the IOF terminal. IOF will issue a prompt "C: " which invites the user to enter a COMPILE command. The prompt "R: " then invites the user to enter an editor request. Depending upon the request entered, an input prompt may be displayed inviting the user to enter input data. The "R: " and input prompts are issued repeatedly by IOF until the user enters a Q or QUIT request. The Q request will cause the "C: " prompt to be issued again so that a further COMPILE command can be entered. The QUIT request will cause the "S: " prompt to be issued so that the user can enter the next JCL statement. The entry of a command file via an IOF terminal is illustrated in the figure "Sample Interactive Compilation". The use of IOF is also discussed in "Creating a Library Member Interactively", Chapter 1.

When neither the parameter SOURCE nor the parameter COMFILE are present but the parameter LIB is present in a CBL statement entered at an IOF terminal, an implicit source member exists. The implicit source member name is of the form "username_EDT", in which username is the name of the user given when starting the IOF session. If the library specified in the LIB parameter contains a member whose name is the implicit source member name, every COMPILE command that does not explicitly specify a member name is considered specifying the implicit source member name.

So, if the user name is MY-NAME and the library MY-LIBRARY contains a member called MY-NAME_EDT, the (first) program contained in that member will be compiled by:

```
COMPILE;
```

Contrary to the way lines are input under the EDIT command of LIBMAINT, and under the COMPILE command when making a batch compilation, the input prompt, within an interactive compilation, is not "I: ", but ".0001 " or "00001 ", ".0002 " or "00002 ", ... , with a number incremented as the lines are entered. In addition, lines are entered starting from "column 8" (Area A), and without identification area, regardless of the actual language type of the source text in library.

If a character is to be entered in the indicator area (conceptual "column 7"), one can depress "backspace" followed by the character "*", "/", "-", or "D", or better, one can simply type the character as the first character of the input (followed by a blank if the character is the debugging line mark "D"). If this character is the first character of the line (i.e. not preceded by a "backspace"), it will be put in "column 7" by the compiler; if it is "-", or if it is "D" followed by two blanks (instead of one), the next input character will be put in "column 12" (Area B); in the other cases, the next input character will be put in "column 8" (Area A). If no character is to be entered in "column 7", and the text starts in "column 12" (Area B), only one blank is needed at the beginning of the input line. The compiler will then add 3 extra blanks to make the line start in Area B.

For example the following input:

```
C: COMPILE;
R: R MY-PROGRAM
R: 15A
.0001 *****
.0002 PAR.
.0003 COMPUTE THREE = O
.0004 -NE + TWO
.0005 D + ZE-RO
.0006 /
R: /
```

will insert after 15 of MY-PROGRAM the following text:

```
*****
PAR.
    COMPUTE THREE = O
-   NE + TWO
D   + ZE-RO
```

with the necessary sequence number and/or identification areas, as required by the language type of MY-PROGRAM.

When the COMFILE parameter is used in the COBOL JCL statement typed at the IOF terminal, the compiler executes as if the COMFILE parameter was not specified, except that the "C: " and "R: " prompts are shown as "C.." and "R.." pseudo-prompts, and that the input is taken from the COMFILE file, the same way as in a batch compilation (as opposed to entered by the user). More specifically, the pseudo-prompt to input text is "I..", and the input text must take into account the language type of the source text, i.e. it must contain sequence number and/or identification areas as required, and the Area A must be 4 characters long.

The printer output is stored in the TEMP source library rather than in a standard SYSOUT subfile, if neither the PRTFILE nor the PRTLIB parameter is used. If the user wishes the printer output to be actually printed, a WRITER JCL statement with the MB=program-name_L parameter must be used. Before it is printed, the output can be examined using SCANNER or LIBMAINT.

Diagnostic messages are output on the IOF terminal as well as in the program listing if the NSILENT parameter is active. If the SILENT parameter is specified, only compiler banners, source identifications, and compilation summary are displayed upon the IOF terminal. If a given error message is output more than once for a particular compilation, the explanatory text part of the message is not repeated on the terminal for the second and subsequent occurrences of the message, except for those variable parts of the message which are shown enclosed between "...". The compiler will suppress the output of observation (*), warning (**), serious (***), and fatal (****) messages when 20 lines with diagnostics have been output on the terminal. If more than 20 source lines contain diagnostics, the lines with fatal diagnostics (****) are first selected, then the lines with serious diagnostics (***) are selected, then lines with warnings (**), and finally lines with observations (*); the 20 first selected lines are then printed on the IOF terminal, obviously in the order of the source listing.

The parameters DCLXREF, BDCLXREF, DDLIST, DIAGAFT, DIAGBEF, NLIST, NCLIST, CMTLIST, MAP, BXREF, XREF, OBSAFT, OBSBEF, WARNAFT, WARNBEF, XLN and ILN have no effect upon the output to the terminal. These parameters only affect the printer output. All messages sent to the JOR by the compiler are also sent to the terminal, including the compilation summary CBL02.

If the Break key on the terminal is pressed, the system prompts 3 question marks (???). If a "slash" (/) is returned to the prompt, the current compilation is terminated immediately, and the next compilation (in case of serial compilation) is started. If "QUIT;" is returned to the "???" prompt, the current compilation or series of compilations is terminated immediately.

Sample Interactive Compilation

```
S: CBL LIB=DOC;
>>>12:38 COBOL V-01.99-5
   APR 15, 1993 12:38:24 X4339.5
C: COMPILE;
R: R FIND-DAY
   PROCESSING OF FIND-DAY:DOC
R: /DATA/S/DIVISION/&./
R: /01  DITWEEK-TAB/,/SUNDAY/C            BLABLA
                              ?
*  End of line assumed to be comment
.0001  COPY DAYS
.0002  REPLACING == PIC X(8) == BY == PIC X(10) ==.
.0003 /
R: /
    37.2.3             02 FILLER COMP-1 SYNC.
                         1      2
*    1  2-199 A 1 byte type 2 FILLER item was allocated to align this
              synchronized item (see the COBOL 85 Reference Manual).
**** 2  3-41  This feature is a NON STANDARD feature, not included
              in the current compilation level.


        87               MOVE SPLIT-DATE TO SHORT-DATE
                                            1
**   1  5-148 The receiving item may be truncated on the right.
**   1  5-264 Sending and receiving fields overlap.
*    1  5-184 This is a group move and operands do not have
              the same size.

       114             MOVE Y TO DAY-OF-THE-WEEK.
                                 1
**   1  5-156 Possible left truncation.


   SUMMARY FOR FIND-DAY: ****=1 **=3 *=3 NO CU PRODUCED.
```

```
   APR 15, 1993 12:41:24 X4339.5
C: COMPILE LOBSERV;
R: R CALENDAR
   PROCESSING OF CALENDAR:DOC
R: /
        14         FD  PRT LABEL RECORDS OMITTED.
                       1
*    1  3-254 This feature is temporarily kept in the COBOL
              Standard. It will disappear from a future version
              of the Standard.

       125         01    L COMP-2.
                         1
*    1  3-41  This feature is a NON STANDARD feature, not included
              in the current compilation level.


       126         01    O PIC LX(3000) DEPENDING ON L.
                                 1
*    1  1-159 This feature ('L' in PICTURE character-string) is a
              NON STANDARD feature, not included in the current
              compilation level.

       163             MOVE I TO CUR-MONTH.
                                  1
**   1  5-156 Possible left truncation.

   SUMMARY FOR CALENDAR: **=1 *=3 OBSOLETE=1 CU PRODUCED.

   APR 15, 1993 12:42:00 X4339.5
C: /
<<<12:42

S: CBL SOURCE=(CALENDAR FIND-DAY) LIB=DOC;
>>>12:46 COBOL V-01.99-5
   APR 15, 1993 12:46:48 X4339.7 PROCESSING OF CALENDAR:DOC
        14         FD  PRT LABEL RECORDS OMITTED.
                       1
*    1  3-254 This feature is temporarily kept in the COBOL
              Standard. It will disappear from a future version of
              the Standard.

       125         01    L COMP-2.
                         1
**** 1  3-41  This feature is a NON STANDARD feature, not included
              in the current compilation level.
```

```
.i.QUIT request;

???QUIT;

   SUMMARY FOR CALENDAR: PROCESSING TERMINATED BY USER.

<<<12:47
S: CBL SOURCE=(CALENDAR FIND-DAY) LIB=DOC LOBSERV;
>>>12:50 COBOL V-01.99-5
   APR 15, 1993 12:50:24 X4339.9 PROCESSING OF CALENDAR:DOC
        14         FD  PRT LABEL RECORDS OMITTED.
                           1
*    1  3-254 This feature is temporarily kept in the COBOL
              Standard. It will disappear from a future version of
              the Standard.

       125         01    L COMP-2.
                           1
*    1  3-41  This feature is a NON STANDARD feature, not included
              in the current compilation level.

       126         01    O PIC LX(3000) DEPENDING ON L.
                                 1
*    1  1-159 This feature ('L' in PICTURE character-string) is a
              NON STANDARD feature, not included in the current
              compilation level.


       163             MOVE I TO CUR-MONTH.
                                  1
**   1  5-156 Possible left truncation.


   SUMMARY FOR CALENDAR: **=1 *=3 OBSOLETE=1 CU PRODUCED.


   APR 15, 1993 12:50:24 X4339.9 PROCESSING OF FIND-DAY:DOC
        14         WORKING-STORAGE SECTION.
                    1
**   1  3-3   Period expected after the previous word.
*    1  3-154 Syntax checking is resumed at this point.

       87              MOVE SPLIT-DATE TO SHORT-DATE
                                          1
**   1  5-148 The receiving item may be truncated on the right.
**   1  5-264 Sending and receiving fields overlap.
*    1  5-184 This is a group move and operands do not have
              the same size.
```

```
       114              MOVE Y TO DAY-OF-THE-WEEK.
                                    1
**   1  5-156 Possible left truncation.


   SUMMARY FOR FIND-DAY: **=4 *=2 CU PRODUCED.
<<<12:50
```

2.5 Compiler Limits

The COBOL compiler has the limits shown below.

Table 2-3. Compiler Limits

Variable	Limit
Number of lines of source COBOL	more than 500000
Number of user-names (data, FILLER, paragraphs)	more than 200000
Size of numeric item (ANS standard is 18)	30 digits
Size of numeric literal (ANS standard is 18)	30 digits
Size of non-numeric literal in object program (ANS standard is 160)	256 char.
Size of a code segment	65000 bytes
Size of an edited item	256 bytes
Number of files per program	500 or less depending on number of object segments
Number of files per run-unit	512
Number of files in MULTIPLE FILE TAPE clause	253
Number of ALTERNATE RECORD KEYs per file	15
Length of a data item (elementary or group)	4194304 bytes
Length of record in a file	32768 bytes
Length of a record key or alternate record key	256 bytes
Value of a relative key	16777215 (*)
value in WRITE ADVANCING	255
Number of data names (non FILLER) in a record description containing level 66 entries	about 600

* UFAS-EXTENDED allows a relative key value up to (2**32 - 1) = 4294967295. A specific revision of the COBOL 85 compiler can generate programs which are able to access the whole range of UFAS-EXTENDED relative key values. However, such programs cannot access relative files which are not UFAS-EXTENDED. Therefore, after any program declaring UFAS relative files is recompiled using this specific revision of the COBOL 85 compiler, non-UFAS-EXTENDED relative files may have to be reallocated as UFAS-EXTENDED even though they do not extend beyond the current limit of 16777215 for a relative key value.

Limits which are in excess of the ANS standard are available only if the LEVEL=NSTD parameter is included in the CBL JCL statement; otherwise the ANS standard limits apply.

2.6 Object Code

The following Procedure Division extract shows 4 COBOL statements on lines 22, 24, 30 and 31.

```
20      PROCEDURE DIVISION.
21      DEBUT.
22          OPEN INPUT F1.
23      LEC.
24          READ F1 AT END GO TO FIN.
        .
        .
29      FIN.
30          CLOSE F1.
31          STOP RUN.
```

The corresponding procedure map extract (in line number order) gives:

```
22 2:0000C    24 2:0007A    24 2:00096    -- -------
-- -------    30 2:000CE    31 2:0010A    -- -------
```

Thus the first COBOL statement starts at address C. There is an object code that occurs before the first COBOL line which is known as the "Prologue". The Prologue is executed at the start of the program and carries out certain housekeeping functions. Each COBOL statement included in the Procedure Division is compiled into object code together with a note of its source line number. If a line of source COBOL contains more than one statement there will be several sets of compiled code for that line. In this case the procedure map contains several entries for the line (line 24).

The object code generated from statements such as OPEN, CLOSE, OCCURS DEPENDING ON uses subroutines which are generated at the end of the main object code in what is called the "Epilogue". These subroutines are usually appended at the end of the last code segment with a priority of zero. However, if exponentiation with TEMP IS 13, or with COMP-9, COMP-10 and COMP-15 data items is used, the associated subroutines will be in the code segment containing this code.

2.7 Printer Output

The following paragraphs describe the printer output produced by the compiler. The output is described in the order in which it is produced, under the following headings:

- Banner page
- Program listing
- Map and cross-reference listings
- Summary page

To enable the listing below to fit in this manual, some non-significant characters were suppressed and some lines were folded.

```
*****************************************************************************************************
*****************************************************************************************************
**** GCOS7                                                                                      ****
****                          C  O  B  O  L                                                     ****
****                                                  VERSION: 01     DATED: DEC 01, 1987 ****
*****************************************************************************************************
*****************************************************************************************************



PROGRAM: FIND-DAY

USER: MARTIN

PROJECT: CB85

DATE: MAR 23, 1988

TIME: 18:32:24

COMPILER VERSION: INTERNAL V-01

USER OPTIONS: SOURCE=FIND-DAY COMFILE LIB=0 LIB PRTLIB LEVEL=NSTD DCLXREF XREF

ACTIVE OPTIONS: OBJ, NDEBUG, WARN, OBSERV, NMAP, DCLXREF, XREF, LIST, CKSEQ, CARDID, CASEQ,
                EXPSIZE, DIAGIN, NCODAPND, NOPT, NDEBUGMD, LFATAL, OOBSERV, XLN, NSILENT,
                NDDLIST, NREFMDCK, COBOL85, PSEGMAX=4096(BYTES), DSEGMAX=4096(BYTES),
                ISEGMAX=(65536(BYTES), 0, 16384(BYTES)), TEMP=30, CODE=OBJA.

COMPILATION LEVEL: NOT STANDARD
```

```
COMPILER INPUT:
  ALTER FILE
    ALTER-DAYS IN COBL.DOC (H_ALTER)
    CD=01/23/78 CT=10:35:24 MD=01/23/78 MT=10:35:24 SL=DAT MN=00 NM=ALTER-DAYS
  SOURCE FILE
    FIND-DAY IN COBL.DOC (H_INLIB)
    CD=01/23/78 CT=10:35:24 MD=03/23/88 MT=14:54:36 SL=DAT MN=14 NM=FIND-DAY
  COPY FILE (COPY STATEMENT ON LINE 37.0, COPIED TEXT ON 12 LINES)
    DAYS IN COBL.DOC (H_INLIB)
    CD=01/23/78 CT=10:35:24 MD=01/23/78 MT=10:35:24 SL=DAT MN=00 NM=DAYS
```

2.7.1 Banner Page

A sample banner page is shown in the figure above. The information appearing on this page is discussed in the following paragraphs.

Program

This shows the program-name taken from the PROGRAM-ID paragraph of the source program. If there is no PROGRAM-ID paragraph in the program the compiler assigns a name to the program based upon the current date and time:

yyddd_hhmmss_ttt

where:
- yy is year
- ddd is day
- hh is hour
- mm are minutes
- ss are seconds
- ttt are thousandths of a second

This program-name is used in forming the printer output when a permanent SYSOUT file or library is being used or when the printer output is stored in the TEMP source library.

USER and PROJECT

These items show the USER and PROJECT specified in the $JOB statement, or those specified at logging time if the compilation is started under IOF.

Date and Time

These items show the system date and time at which the compilation was done.

Compiler Version

This shows the version of the compiler being used and the patches which have been applied to it. For example:

```
COMPILER VERSION: COBOL V-01
```

If patches have been applied to the compiler the relative number of the latest series of patches applied appears after the version number. For example:

```
COMPILER VERSION: COBOL V-01.6
```

If one or more patches prior to the latest patch have not been applied, then the number of non-applied patches follows the latest patch number. For example:

```
COMPILER VERSION: COBOL V-01.6-2
```

In such a case, the numbers of the missing patches are listed in the following way:

```
COMPILER VERSION: COBOL V-01.6-2
COMPILER VERSION ADDITIONAL INFORMATION: 001 004
```

where 001 and 004 are the numbers of the missing patches.

USER OPTIONS and ACTIVE OPTIONS

USER OPTIONS lists the parameters specified by the user in the CBL JCL statement. ACTIVE OPTIONS lists all of the compiler parameters in use, either default and/or user-specified.

Compilation Level

This shows the level of COBOL which is expected by the compiler and is derived from the LEVEL parameter in the CBL JCL statement. The possible values are: NOT STANDARD, ANSI, HIGH, INTERMEDIATE or MINIMUM.

Optional Modules

This shows the levels of optional modules which are expected by the compiler and is derived from the LEVEL parameter in the CBL JCL statement. This entry is present when the level of COBOL is HIGH, INTERMEDIATE or MINIMUM only. The possible values are any combination of REPORT WRITER, COMMUNICATION HIGH or COMMUNICATION LOW, DEBUG HIGH or DEBUG LOW, and SEGMENTATION HIGH or SEGMENTATION LOW. When the LEVEL parameter specifies no optional modules, OPTIONAL MODULES: NONE is printed.

Compiler Input

This identifies the file and subfile from which the source program was read. The command file (see "The Alter Facility", above) is also identified if it is used. If the source program contains a COPY statement, the file from which text is copied is also identified.

A two, three or more line identification is printed for each type of file. The first line identifies the type of file: ALTER FILE, SOURCE FILE, ALTERED SOURCE FILE, COPY FILE, or OBJECT SUBSCHEMA. If the file is a COPY FILE, the line numbers of the copied lines are also specified. For example:

```
COPY FILE (COPY STATEMENT ON LINES 32.0, COPIED TEXT ON 12 LINES)
```

The second line gives the file-name, subfile-name (if a subfile is being used) and the internal-file-name used by the compiler, in the following way:

```
subfile-name IN file-name (internal-file-name).
```

The third line appears only if the file or subfile is in SSF format or is an object schema or sub-schema. It gives information taken from the type 101 control record. It contains the following list of mnemonics and values:

CD =	creation date
CT =	creation time
MD =	date last modified
MT =	time last modified
SL =	source language type (DAT, CBL, CBX, DD)
MN =	modification number. This is zero for a new file or subfile and is increased by one for each update.
NM =	name (normally the same as the subfile name.)

The extra lines appear only if the file or subfile is in SSF format and was obtained as the result of a SOURCE command of the Dictionary Maintenance Services. Each extra line gives information on the Dictionary object from which the source in the file or subfile, or part thereof, is generated from. The first element of the line is the identification of the Dictionary base. It is followed by the following mnemonics and values:

MD = date last modified

TP = type of the object within the Dictionary (RECORD, FILE, ...)

NM = identification of the object within the Dictionary.

Program Listing

The program listing may be printed in one, two or three sections.

- Alter listing,
- Source listing,
- Error listing.

The alter listing is a listing of the contents of the command file (note that this listing has no connection with the ALTER statement). The alter listing is always produced when a command file is used. It is printed before the source listing.

The source listing is printed if there is no NLIST parameter specified in the CBL JCL statement. The listing incorporates any alterations specified in a command file and (if there is no NCLIST parameter in the CBL JCL statement) any text referred to in COPY statements. The effects of REPLACE statements or of a REPLACING clause in a COPY statement are shown. That is, the specified words are listed in their new form.

The internal line numbers of alter listings and source listings are distinguished in the following ways:

- - Alter listing. The internal line numbers in an alter listing are always prefixed by "A.". For example A.101, A.102, A.103.
- Source listing. The internal line numbers of source listings are not prefixed.

The above prefixes are also used in the summary of errors on the summary page of the compiler listing.

Sample Alter Listing

```
FIND-DAY      ALTER LISTING

   A.1 -----> C: COMPILE;

   A.2        R: R FIND-DAY

   A.3        R: /DATA/S/DIVISION/&./

   A.4        R: /01  DITWEEK-TAB/,/SUNDAY/C            COMMENT

                                          1
* 1  1-44   Text follows the 'A', 'C', 'I' or 'Q' command on the
            line.
            Text is ignored.

   A.5        I:   COPY DAYS
   A.6        I:     REPLACING == PIC X(8) == BY == PIC X(10) ==.
   A.7        I: [F


FIND-DAY          SOURCE LISTING
    XLN  TEXT  7-10-------20-------30-------40-------50-------60-------70--....<-

         1         IDENTIFICATION DIVISION.
         2        *                      THIS ROUTINE, STARTING FROM A DATE, GIVES
         3        *                      THE DAY IN THE WEEK CORRESPONDING TO THE
         4        *                      DATE
         5         PROGRAM-ID. FIND-DAY.
         6        *
         7         ENVIRONMENT DIVISION.
         8         CONFIGURATION SECTION.
         9         SOURCE-COMPUTER. DPS7                        .
        10         OBJECT-COMPUTER. DPS7.
        11        *
       *12         DATA DIVISION.
        13        *
        14         WORKING-STORAGE SECTION.
        15        *                       TEMPORARIES
        16         01  X PICTURE 9(10).
        17         01  Y PICTURE 9(5).
        18        *                      TOTAL NUMBER OF DAYS PRECEDING THE MONTH
        19        *                      (SHOWN BY ITS ORDINAL NUMBER IN THE LIST)
        20        *                      IN THE YEAR
```

```
 21         01  PREC-D-TAB.
 22             02  FILLER PIC 999 VALUE   0.
 23             02  FILLER PIC 999 VALUE  31.
 24             02  FILLER PIC 999 VALUE  59.
 25             02  FILLER PIC 999 VALUE  90.
 26             02  FILLER PIC 999 VALUE 120.
 27             02  FILLER PIC 999 VALUE 151.
 28             02  FILLER PIC 999 VALUE 181.
 29             02  FILLER PIC 999 VALUE 212.
 30             02  FILLER PIC 999 VALUE 243.
 31             02  FILLER PIC 999 VALUE 273.
 32             02  FILLER PIC 999 VALUE 304.
 33             02  FILLER PIC 999 VALUE 334.
 34         01  PREC-D-TAB-RED REDEFINES PREC-D-TAB.
 35             02  PRECEDING-DAYS PIC 999 OCCURS 12.
 36        *                      TABLE GIVING THE NAME OF THE DAYS IN THE
 37        *                      WEEK
  .             COPY DAYS
  .               REPLACING == PIC X(8) == BY == PIC X(10) ==.
 ..1         01  OTHER-UNUSED.
 ..2             02 FILLER PIC X.
 ..3             02 FILLER COMP-1 SYNC.
 ..4         01  DITWEEK-TAB.
*..5             02  FILLER                       PIC X(10)
*..5                                 VALUE "LUNDI   ".
 ..6             02  FILLER PIC X(10) VALUE "MARDI   ".
 ..7             02  FILLER PIC X(10) VALUE "MERCREDI".
 ..8             02  FILLER PIC X(10) VALUE "JEUDI   ".
 ..9             02  FILLER PIC X(10) VALUE "VENDREDI".
..10             02  FILLER PIC X(10) VALUE "SAMEDI  ".
..11             02  FILLER PIC X(10) VALUE "DIMANCHE".
 -46         01  DITWEEK-TAB-RED REDEFINES DITWEEK-TAB.
  47             02  DAY-IN-THE-WEEK PIC X(10) OCCURS 7 TIMES.
  48        *                     AREA FOR DATE SPLITTING INTO YEAR, MONTH,
  49        *                      AND DAY
  50         01  SPLIT-DATE.
```

Sample Error Listing

```
FIND-DAY          ERROR LISTING

   * 1  2-199 ON LINE 37.0.3 COLUMN 15
              A 1 byte type 2 FILLER item was allocated to align
              this synchronized item
              (see the COBOL 85 Reference Manual).

  ** 1  5-148 ON LINE 87 COLUMN 33
              The receiving item may be truncated on the right.

  ** 1  5-264 ON LINE 87 COLUMN 33
              Sending and receiving fields overlap.

   * 1  5-184 ON LINE 87 COLUMN 33
              This is a group move and operands do not have
              the same size.

  ** 1  5-156 ON LINE 114 COLUMN 22
              Possible left truncation.
```

No source listing will be produced when the NLIST parameter is specified in the CBL JCL statement and no COPY text will be printed when the NCLIST parameter is specified. An error listing is printed whenever the NLIST and/or DIAGAFT or DIAGBEF or WARNAFT or WARNBEF or OBSAFT or OBSBEF parameters are active.

The error listing presents messages generated by the compiler under two forms. In the first form, the message appears together with the relevant line of source program. In the second form, no source line is listed; instead, the line number and column number of the corresponding source code is printed with the message. The first form appears only when the NLIST parameter is specified. The second form appears only when either of DIAGAFT, DIAGBEF, WARNAFT, WARNBEF, OBSAFT or OBSBEF is active.

When DIAGBEF or DIAGAFT is specified, the error listing contains messages in the second form.

When DIAGIN is specified:

- If WARNAFT or WARNBEF is specified, the error listing contains warnings and observations (if NOBSERV is not specified) in the second form; if NLIST is specified, this is preceded (if WARNAFT is specified) or followed (if WARNBEF is specified) by serious and fatal error messages in first form.
- If WARN is specified and either OBSAFT or OBSBEF is specified, the error listing contains observations in the second form; if NLIST is specified, this is preceded (if OBSAFT is specified) or followed (if OBSBEF is specified) by serious and fatal error messages and warnings in first form.
- In the other cases, the error listing is produced only when NLIST is specified, then it contains messages in the first form.

The layout of the source listing is described in the following paragraphs.

Headings

The first two lines of heading are self-explanatory. The meaning of the third line of heading is as follows:

ILN	Internal line number, used by the compiler to identify lines of source code. This is independent of the external line number (XLN). ILN is present only if the parameter ILN or the parameter DEBUG is present in the CBL JCL statement.
XLN	External line number, taken from the source input file.
TEXT	The first column of source code starts under the T of TEXT. Columns 7, 10, 20, 30, 40, 50, 60, and 70 are marked along the line. The specified column appears under the least significant digit of each number. Columns 73 to 80 are marked by periods, followed by <- in columns 81 and 82 to mark the end of the traditional 80-column line.

Source Lines

The components of a line of source code are as follows:

ILN	The ILN starts at one for the first line printed from the source input file and increases by one for each subsequent line from this file, including lines which are copied into the source program using the COPY statement or are included as a result of an A, I or C request in a command file. The ILN is present only if the parameter ILN or the parameter DEBUG or DEBUGGING MODE or DML is specified in the CBL JCL statement.
XLN	This is the line number taken from the SSF header on each source record in the input file. If the input file is not in SSF format, the XLN is taken from the first 6 character positions of the record. A single period to the left of the external line number indicates that the line has been included as the result of an A, I or C request of the command file. A double period to the left of the external line number indicates that the line has been included as the result of a COPY statement. An asterisk to the left of the external line number indicates that the line has been modified as the result of an S request of the command file, or as the result of the application of a REPLACE statement or of the REPLACING clause of a COPY statement. An asterisk also indicates the different parts of a line split up by a COPY statement. A minus sign to the left of the external line number indicates that lines have been deleted from the source before that line as the result of a C or D request in the command file (lines deleted at the end of the program are not shown).
TEXT	Columns 1 to 6 of the text may contain a line sequence number. This number corresponds to the sequence number which may be punched in columns 1 to 6 if the program is input to the system on cards. There will be no such sequence number if the source input file has a language type COBOL or COBOLX.

In the source listing, whenever a line is not in 80 column format (with source from columns 8 to 72) an arrow <- is printed in the two columns following the card identifier area if one of the following conditions is met:

- The line contains a non-numeric literal that continues on the next line, or
- the line is shorter than 80 characters and it contains a non blank character in the identification area, or
- the line is longer than 80 characters and it contains a non blank character at a column number greater than 72.

Identification Division headers of contained programs are flagged with a number followed by an arrow pointing to the right in the first column of the listing line. The number is the nesting level of the contained program (e.g. 2-> for the Identification Division header of a contained program a second level of nesting). End Program headers of contained programs are flagged the same as their Identification Division header but the arrow is pointing to the left (e.g. 2<-).

Diagnostic Error Messages

Diagnostic messages are generated when the compiler detects incorrect or inconsistent code in the source program. (A complete list of error messages is given in Appendix B). Diagnostic messages are also generated when the compiler detects incorrect or inconsistent requests in the alter file. These messages normally appear embedded in the source listing. However, if the DIAGAFT parameter is specified in the CBL JCL statement, all errors will be listed after the source listing, except for the errors in the command file which are always embedded in the Alter Listing. If the DIAGBEF parameter is specified in the CBL JCL statement, all errors will be listed before the source listing, except for the errors in the command file which are always embedded in the Alter Listing.

The format of a diagnostic message is as follows:

```
aaaa o p-nnn message-text
```

where:

aaaa	can be one, two, three or four asterisks, indicating the severity of the message. * is an observation, ** is a warning, *** is a serious error and **** is a fatal error.
o	is a number from 1 to 9 indicating the order of the message, when it refers to a specific piece of text in the line.
p-nnn	is the number given to the error which has occurred.
message-text	is a plain English explanation of the error.

Each part of the message is described in turn in the following paragraphs.

An observation message (one asterisk) indicates the action taken by the compiler where this may not be clear from the source code. Observation messages can be suppressed by specifying the NOBSERV or NWARN parameter in the CBL JCL statement. The following example contains an observation message:

```
14       FD F1
15           DATA RECORD A1 A2.
16       01 A1 PIC X(80).
17       01 A2 PIC X(80).
18       01 AO PIC X(80).
            1
1 3-191 Record description assumed to be data record
         for the preceding FD.
```

A warning message (two asterisks) indicates a possible error. The source statement is compiled but the results may be different from those intended by the programmer. Warning messages can be suppressed by specifying the NWARN parameter in the CBL JCL statement. The following example contains a warning message:

```
20           WORKING-STORAGE SECTION.
21           01 AA PIC X(8).
22           01 BB PIC X(6).
23           PROCEDURE DIVISION.
24           P1.
25                MOVE AA TO BB.
                           1
** 1 5-148 This receiving item may be truncated on the right.
```

A serious error message (three asterisks) indicates a major error in the program. The compiler continues to check the source code but does not generate a compile unit. The message "NO CU PRODUCED" will be printed on the summary page of the compiler listing. The following example contains a serious error message:

```
10           FD F1 LABEL RECORD OMITTED.
11           01 AI               PIC X(80).
20           WORKING-STORAGE SECTION.
21           01 RECEP            PIC X(4).
22           PROCEDURE DIVISION.
23           S1 SECTION 1.
24           P1.
                WRITE A1.        (A1 instead of AI)
                      1
*** 1 6-2 Item not declared.
* 1 5-164 Syntax checking discontinued.
```

The "Syntax checking discontinued" observation message indicates that the compiler has interrupted analysis from this point, until it encounters a recognizable sequence (point, verb, paragraph, section, etc.). The compiler then resumes its analysis indicating "Syntax checking resumed".

A fatal error message (four asterisks) indicates that an error has occurred which:

- prevents the compilation from continuing its analysis. It may be a system error (e.g. unable to read a source file), a compiler error (e.g. irrecoverable difficulty), a compiler limit (e.g. too many operands in the REPLACING phrase of a COPY statement), a user error (e.g. absence of the specified source member from the specified input library or libraries); or
- prevents the compilation from generating object code (e.g. use of a feature not included in the level of compilation explicitly or implicitly specified by the LEVEL parameter of the CBL JCL statement).

The following example contains a fatal error message:

```
52 520            03 DATA-ITEM PIC X VALUE 'X'.
                                          1
*    1 1-100   The apostrophe is used instead of the quote to
               delimit literals in this program.
**** 1 1-26    This feature is a NON STANDARD feature, not
               included in the current compilation level.
```

In this example the compilation has been requested without the LEVEL=NSTD parameter in the CBL JCL statement. Should this parameter have been included, only the first message would have been issued.

There may be several error messages for the same line of source code. In this case, the error order number, which is printed after the asterisks in the message, is used to relate the messages to the errors in the source line. This number is also printed under each error in the source line as shown in the following example:

```
     565 586    PERFORM L TEST AFTER UNTIL A9 EQUAL TO B9.<-
                1         2                    3
*    1 5-165   Syntax checking resumed.
**** 2 5-162   This feature is HIGH LEVEL feature, not
               included in the current compilation level.
***  3 6-2     Item not declared.
*    3 5-164   Syntax checking discontinued.
```

The error under PERFORM results from an error on a previous line.

The error under TEST results from LEVEL=I used in the CBL JCL statement. The error under EQUAL results from a truncation of the line by the compiler. The program was created with TYPE=COBOL which implies the presence of a card identifier area. Therefore, the last eight characters are ignored and the compiler recognized EQUA instead of EQUAL. Hence the error messages. The arrow at the end of the line indicates that the line terminates after the termination period.

2.7.2 Map Listings and Cross-Reference Listings

Depending on the parameters specified by the user in the CBL JCL statement, some or all the following memory map listings and cross-reference listings may be produced:

- data map and procedure definition listing
- cross-reference listing (declaration order)
- cross-reference listing (alphabetic order)
- procedure map listing
- perform/alter bucket listing.

These listings are always printed in the order shown above. However, if a data map/procedure definition listing and cross reference listing are both requested by the user, they are combined in a single listing (i.e., only the cross-reference listing is produced).

The parameters which must be specified in the CBL JCL statement to obtain map and cross-reference listings are as follows.

- MAP produces a data map/procedure definition listing, procedure map listing and perform/alter bucket listing.
- DMAP produces a data map and procedure definition listing.
- PMAP produces a procedure map listing and perform/alter bucket listing.
- DCLXREF or BDCLXREF produces a cross-reference listing (declaration order).
- XREF or BXREF produces a cross-reference listing (alphabetic order).

Each of the above listings is described in the following paragraphs.

Data Map and Procedure Definition Listing

The data map and procedure definition listing is divided into several sections separated by dotted lines. There is a section for each program comprised in the compiled source. The first section concerns the externally compiled program (i.e. not including contained programs). Following the first section, sections related to contained programs (a section per contained program) appear in source order. All these sections have the same structure as described hereafter.

The first line of a section gives the name of the program with the indication PROGRAM-NAME under the header DESCRIPTION, followed in the first section by the indication SEPARATELY COMPILED. Following the program name, the data map and procedure definition listing comprises a list of all the identifiers in the Data Division, printed in the order in which the identifiers are defined.

For each identifier the following information is printed:

- level number (if applicable)
- name
- parameter number (if applicable)
- memory address (if applicable)
- description
- line number of the line in which the identifier is defined

A sample data map and procedure definition listing is shown in the figure below. The contents of this listing are described below.

The parameter-number is listed under the heading PN. It is used only for identifiers which are included in the USING clause of a Procedure Division header or their subordinate, redefining or renaming data items, and specifies the position of the parameter within the USING clause.

The memory address of the object generated for each data item is shown under the ADDRESS heading. For data allocated in static segments, the address is of the form isn:sra, where isn is the internal segment number (decimal) and sra is the address (hexadecimal) relative to the start of the internal segment (for an explanation of segment numbers see "Linking", Chapter 3).

For data allocated in the first area of the stack segment, the address is of the form ?BR1.offset where offset is the hexadecimal value that must be added to the contents of the base register 1 to get the address of the data item. For data allocated in the second, third,... area of the stack segment, the address is of the form ?BR1.base->offset where base is the hexadecimal value that must be added to the contents of the base register 1 to get the address of an ITS, offset is the hexadecimal value that must be added to the contents of that ITS to get the address of the data item. For based data items, i.e., those declared in the Linkage Section which are not parameters, the address is of the form base->offset, where base is the address of an ITS in the form described above for data allocated in static segments or in the stack segment, offset is the hexadecimal value that must be added to the contents of this ITS to get the address of the data item. For parameters, only an offset is shown under the ADDRESS heading; it is the offset (hexadecimal) within the data item which is referenced in the Procedure Division header.

Under DESCRIPTION there is an identification of the type of object to which the identifier refers. This description may be one of the following:

- GROUP indicates a group item composed of subordinate group or elementary items.
- BINARY indicates a usage BINARY data item.
- BIT indicates a usage BIT data item.
- DISPLAY indicates an elementary item with (usually by default) USAGE IS DISPLAY.
- COMP, COMP-1, etc. indicate items defined with one of the COMPUTATIONAL options.
- INDEX indicates an index data item.
- PAC-DEC indicates a usage PACKED-DECIMAL data item.
- POINTER indicates a POINTER data item.

- INDEX-NAME indicates an index-name declared by use of the INDEXED BY clause in a table description.
- ALPHABET-NAME indicates an alphabet-name.
- CONDITION-NAME indicates a level 88 condition-name.
- MNEMONIC-NAME indicates a mnemonic-name.
- SECTION-NAME indicates a section-name.
- PARAGRAPH-NAME indicates a paragraph-name.
- INDEXED indicates an INDEXED file.
- RELATIVE indicates a RELATIVE file.
- SEQUENTIAL indicates a SEQUENTIAL file.
- ...-DYNAMIC where ACCESS MODE IS DYNAMIC clause is used.
- ...-RANDOM where ACCESS MODE IS RANDOM clause is used.
- SORT-MERGE indicates a sort-merge-file (SD appears under LN).
- INPUT indicates an input CD (CD appears under LN).
- OUTPUT indicates an output CD (CD appears under LN).
- I-O indicates an input-output CD (CD appears under LN).
- ...-INITIAL indicates an initial CD.
- SCHEMA-NAME indicates a schema-name.
- AREA-NAME indicates an area-name.
- SET-NAME indicates a set-name.
- DB-KEY indicates a db-key-name.
- REPORT-NAME indicates a report-name (RD appears under LN).
- REPORT-HEADING indicates a report heading report group.
- PAGE-HEADING indicates a page heading report group.
- CONTROL-HEADING indicates a control heading report group.
- DETAIL indicates a detail report group.
- CONTROL-FOOTING indicates a control footing report group.
- PAGE-FOOTING indicates a page footing report group.
- REPORT-FOOTING indicates a report footing report group.
- ELEMENT indicates a report group item.

When applicable, the right hand part of the DESCRIPTION contains a simplified description of the explicit or implicit PICTURE clause. For those PICTURE clauses which are not of a form like 9(n)V9(n), X(n), A(n), ... , the description may be one of the following:

- SIMPLE-FLOAT indicates a COMP-9 floating point item.
- DOUBLE-FLOAT indicates a COMP-10 floating point item.
- QUADR.FLOAT indicates a COMP-15 floating point item.
- FIX BIN (15) indicates a COMP-1 binary item.
- FIX BIN (31) indicates a COMP-2 binary item.
- FLOAT DEC indicates a usage display decimal floating point item.
- NUM-EDT indicates a numeric edited item whose length is shown between parentheses.
- AN-EDT indicates a alphanumeric edited item whose length is shown between parentheses.

Under the heading DEF the line number at which the identifier is defined is shown. The line number is the internal line number if the parameter ILN is specified in the CBL JCL statement; it is the external line number if XLN is specified.

The data map and procedure definition listing also contains a list of all paragraph-names and section-names (NAME) together with the line numbers at which these names are defined (DEF). The line number is the internal line number if the parameter ILN is specified in the CBL JCL statement; it is the external line number if XLN is specified. This list appears at the end of the data map listing and is in the sequence in which the paragraph or section names are defined. See the figure below for an example. The type of name is indicated under DESCRIPTION by PARAGRAPH-NAME or SECTION-NAME. This list is used in conjunction with the procedure map listing. The procedure map listing contains memory addresses and line numbers. These line numbers can be related to the containing paragraph or section using this listing.

When externally compiled programs are referenced, the map and procedure definition listing terminates with a section that lists these programs. Each line of this last section has the same form as the first line of the first section.

Cross-Reference Listing (Declaration Order)

A cross-reference listing in declaration order contains all the information included in a data map listing. In addition to this, for each identifier, there is a list of line numbers for those lines which refer to the identifier. The line numbers are internal line numbers if the parameter ILN is specified in the CBL JCL statement; they are external line numbers if XLN is specified. An example is given in the figure below.

This listing is printed in the same sequence and has the same format as the data map listing, except that the additional information is printed under the heading REF (+ WHEN MULTIPLE, * WHEN CHANGED). More than one reference to the same identifier on a single line is shown by a plus sign following the line number. When an identifier is used to specify a receiving item, an asterisk (*) is printed to the right of the line number.

Cross-Reference Listing (Alphabetic Order)

The cross-reference listing in alphabetic order contains all the information in the cross-reference listing in declaration order except that the lines are sorted into alphabetic identifier order. An example is given in the figure below.

An additional piece of information in the cross-reference listing in alphabetical order is the 01 level data-name which appears in parentheses after each non-01 level data-name. This shows the record-name to which each data-name belongs. For 01 record-names belonging to an FD or an SD, the corresponding file-name appears between parentheses after each record-name; similarly, report-items contain their father names between parentheses. For paragraph-names within section, the section-name appears between parentheses after the paragraph-name.

To enable the listing in this figure to fit in this manual, some non-significant characters were suppressed.

Sample Data Map and Procedure Definition Listing

```
FIND-DAY     DATA MAP AND PROCEDURE DEFINITION LISTING
LM NAME                        PN   ADDRESS   DESCRIPTION          DEF

   FIND-DAY                                   PROGRAM-NAME         1     SEPARATELY-COMPILED

01 X                         1:000055   DISPLAY 9(10)        16

01 Y                         1:00005F   DISPLAY 9(5)         17

01 PREC-D-TAB                1:000064   GROUP   X(36)        21

01 PREC-D-TAB-RED            1:000064   GROUP   X(36)        34
02 PRECEDING-DAYS (+ 0)      1:000064   DISPLAY 9(3)         35

01 OTHER-UNUSED              1:000088   GROUP   X(4)         37.0.1

01 DITWEEK-TAB               1:00008C   GROUP   X(70)        37.0.4

01 DITWEEK-TAB-RED           1:00008C   GROUP   X(70)        46
02 DAY-IN-THE-WEEK (+ 0)     1:00008C   DISPLAY X(10)        47

01 SPLIT-DATE                1:0000D4   GROUP   X(8)         50
02 CENTURY (+ 0)             1:0000D4   DISPLAY 99           51
02 SHORT-DATE (+ 2)          1:0000D6   GROUP   X(6)         52
03 MONTH (+ 4)               1:0000D8   DISPLAY 99           54
03 DAY-OF-MONTH (+ 6)        1:0000DA   DISPLAY 99           55
03 DAY-OF-MONTH-X (+ 6)      1:0000DA   DISPLAY XX           56

01 YEAR                      1:0000D4   DISPLAY 9(4)         57

01 DAYS-IN-THE-ERA           1:0000DC   DISPLAY 9(10)        61

01 FULL-DATE                 1    000000   DISPLAY X(8)      67

01 DAY-OF-THE-WEEK           2    000000   DISPLAY 9         70

01 DAY-ITSELF                3    000000   DISPLAY X(10)     72

   BEGIN                                    PARAGRAPH-NAME    76
   THE-END                                  PARAGRAPH-NAME    78
   COMPUTE-DAY-OF-THE-WEEK                  PARAGRAPH-NAME    84
```

To enable the listing in the figure below to fit in this manual, some non-significant characters were suppressed.

Sample CROSS-REFERENCE Listing (Declaration Order)

```
FIND-DAY      CROSS-REFERENCE LISTING (DECLARATION ORDER)
LM NAME                  PN ADDRESS DESCRIPTION    DEF.  REF. (+ WHEN MULTIPLE, * WHEN CHANGED)

   FIND-DAY                        PROGRAM-NAME   1     NOREF SEPARATELY-COMPILED

01 X                       1:000055 DISPLAY 9(10)  16    99* 100 101* 102 103* 104 109*

01 Y                       1:00005F DISPLAY 9(5)   17    109* 110 111* 113* 114 115

01 PREC-D-TAB              1:000064 GROUP   X(36)  21    34

01 PREC-D-TAB-RED          1:000064 GROUP   X(36)  34    NOREF
02 PRECEDING-DAYS (+ 0)    1:000064 DISPLAY 9(3)   35    93

01 OTHER-UNUSED            1:000088 GROUP   X(4)   37.0.1 NOREF

01 DITWEEK-TAB             1:00008C GROUP   X(70)  37.0.4 46

01 DITWEEK-TAB-RED         1:00008C GROUP   X(70)  46    NOREF
02 DAY-IN-THE-WEEK (+ 0)   1:00008C DISPLAY X(10)  47    115

01 SPLIT-DATE              1:0000D4 GROUP   X(8)   50    57 85* 87
02 CENTURY (+ 0)           1:0000D4 DISPLAY 99     51    88*
02 SHORT-DATE (+ 2)        1:0000D6 GROUP   X(6)   52    87*
03 MONTH (+ 4)             1:0000D8 DISPLAY 99     54    93 98
03 DAY-OF-MONTH (+ 6)      1:0000DA DISPLAY 99     55    56 92
03 DAY-OF-MONTH-X (+ 6)    1:0000DA DISPLAY XX     56    86

01 YEAR                    1:0000D4 DISPLAY 9(4)   57    94 98+* 99 101 103

01 DAYS-IN-THE-ERA         1:0000DC DISPLAY 9(10)  61    91* 100* 102* 104* 109

01 FULL-DATE              1  000000 DISPLAY X(8)   67    74 85

01 DAY-OF-THE-WEEK        2  000000 DISPLAY 9      70    74 114*

01 DAY-ITSELF             3  000000 DISPLAY X(10)  72    74 115*
```

To enable the listing in the figure below to fit in this manual, some non-significant characters were suppressed and some lines were split.

Sample CROSS-REFERENCE Listing (Alphabetic Order)

```
FIND-DAY       CROSS-REFERENCE LISTING (ALPHABETIC ORDER)
LM NAME                EDITION    DEF.  REF.      (+ WHEN MULTIPLE,WHEN CHANGED)

   BEGIN                                          PARAGRAPH-NAME 76    NOREF
   THE-END                                        PARAGRAPH-NAME 78    NOREF
   COMPUTE-DAY-OF-THE-WEEK                        PARAGRAPH-NAME 84    77
   BEGIN                                          PARAGRAPH-NAME 76    NOREF
02 CENTURY (SPLIT-DATE + 0)               1:0000D4 DISPLAY 99     51    88*
   COMPUTE-DAY-OF-THE-WEEK                        PARAGRAPH-NAME 84    77
02 DAY-IN-THE-WEEK (DITWEEK-TAB-RED + 0)  1:00008C DISPLAY X(10)  47    115
01 DAY-ITSELF                           3   000000 DISPLAY X(10)  72    74 115*
03 DAY-OF-MONTH (SPLIT-DATE + 6)          1:0000DA DISPLAY 99     55    56 92
03 DAY-OF-MONTH-X (SPLIT-DATE + 6)        1:0000DA DISPLAY XX     56    86
01 DAY-OF-THE-WEEK                      2   000000 DISPLAY 9      70    74 114*
01 DAYS-IN-THE-ERA                        1:0000DC DISPLAY 9(10)  61    91* 100* 102* 104* 109
01 DITWEEK-TAB                            1:00008C GROUP   X(70)  37.0.4 46
01 DITWEEK-TAB-RED                        1:00008C GROUP   X(70)  46    NOREF

   FIND-DAY                                   PROGRAM-NAME         1    NOREF
                                                                        SEPARATELY-COMPILED

01 FULL-DATE                              1:000000 DISPLAY X(8)   67    74 85
03 MONTH (SPLIT-DATE + 4)                 1:0000D8 DISPLAY 99     54    93 98
01 OTHER-UNUSED                           1:000088 GROUP   X(4)   37.0.1 NOREF
01 PREC-D-TAB                             1:000064 GROUP   X(36)  21    34
01 PREC-D-TAB-RED                         1:000064 GROUP   X(36)  34    NOREF
02 PRECEDING-DAYS (PREC-D-TAB-RED + 0)    1:000064 DISPLAY 9(3)   35    93
02 SHORT-DATE (SPLIT-DATE + 2)            1:0000D6 GROUP   X(6)   52    87*
01 SPLIT-DATE                             1:0000D4 GROUP   X(8)   50    57 85* 87
   THE-END                                        PARAGRAPH-NAME 78    NOREF
01 X                                      1:000055 DISPLAY 9(10)  16    99* 100 101* 102 103*
                                                                        104 109*
01 Y                                      1:00005F DISPLAY 9(5)   17    109* 110 111* 113* 114
                                                                        115
01 YEAR                                   1:0000D4 DISPLAY 9(4)   57    94 98+* 99 101 103
```

Procedure Map Listing

The procedure map listing consists of a table of Procedure Division line numbers and the corresponding starting memory addresses for the generated object code. Line numbers are internal line numbers if the parameter ILN is specified in the CBL JCL statement; they are external line numbers if XLN is specified. See the figure below for an example.

The memory address is of the form isn:sra, where isn is the internal segment number (decimal) and sra is the address (hexadecimal) relative to the start of the internal segment. The listing is printed in memory address order so that the user can quickly obtain a line number from a corresponding memory address. This is necessary when a user program terminates abnormally and a memory address is printed in the JOR.

A memory address is printed for each statement in the Procedure Division. Therefore, there will be several memory addresses for the same line number if there is more than one statement on a source line or if the statement implies several simpler statements.

Line numbers will normally be in source order in the procedure map listing. However, if the user has segmented the program using COBOL segment numbers, the object code may be rearranged by the compiler. If this occurs, line numbers in the procedure map listing will not be in source order. In this case, the complete procedure map listing is repeated in source order. That is, two listings are produced, one in memory address order and one in source order.

PERFORM/ALTER Bucket Listing

The information in the perform/alter bucket listing may be used in conjunction with a load module dump to trace the flow of control through the load module which occurred prior to an abnormal termination. (Note that this listing has no connection with the alter listing or alter facility). The listing contains the following information:

- The start address of the 4-byte bucket associated with each paragraph or section that is the last in a sequence of paragraphs or sections referenced in a PERFORM statement. At execution time, if a paragraph or section is being performed, this bucket will point to the instruction following the PERFORM statement which last performed the paragraph or section. If the paragraph or section is not being performed, the bucket will contain the address of the next paragraph.
- The start address of the 4-byte bucket associated with each paragraph referred to in an ALTER statement. At execution time this bucket will point to an address corresponding to the current value of the GO TO in the ALTER paragraph.

For each such address, the line number of the relevant paragraph or section is given in source order before the address. The line number is the internal line number if the ILN parameter is specified in the CBL JCL statement; it is the external line number if XLN is specified.

See below for an example.

To enable the listing in this figure to fit in this manual, some non-significant characters were suppressed and some lines were split.

Sample Procedure Map & Perform/Alter Bucket Listing

```
FIND-DAY          PROCEDURE MAP LISTING
ADDRESS XLN  ADDRESS XLN  ADDRESS XLN  ADDRESS XLN  ADDRESS XLN  ADDRESS XLN  ADDRESS XLN

2:0003A 77   2:00042 79   2:0005E 85   2:00066 86   2:00072 87   2:0007A 88   2:00082 91
2:000E8 98   2:000F0 99   2:00100 100  2:00108 101  2:00112 102  2:0011A 103  2:00124 104
2:0018C 110  2:00198 111  2:001A4 113  2:001AC 114  2:00184 115



FIND-DAY          PERFORM/ALTER BUCKET LISTING
XLN ADDRESS  XLN ADDRESS  XLN ADDRESS  XLN ADDRESS  XLN ADDRESS  XLN ADDRESS  XLN ADDRESS

 84 1:000E8
```

Summary Page

A sample summary page is shown below. The information contained on this page is discussed in the following paragraphs.

Sample Summary Page

```
FIND-DAY       COMPILATION SUMMARY

121 SOURCE LINES


 SUMMARY OF ERRORS

      *             3     ON LINES  A.4   37.0.3   87
    * *             3     ON LINES   87  114
  * * *             0
* * * *             0

N S T D             1


CU PRODUCED ON LIBRARY ;106030.TEMP.CULIB

SEGMENT NAME       TYPE      BYTES   (CODE)    ADDRESS

FIND-DAY.0          ..L        112             ?BR7.50->
FIND-DAY.1          .D.        236             ?BR7.28->
FIND-DAY.2          C..        508   (508)     ?BR7.5C->

STACK                           91

RUN TIME PACKAGE PROCEDURES INVOKED

NONE
```

Source Lines

This shows the total number of lines compiled for this program (including COPY and SCHEMA).

Summary of Errors

This shows the number of observation (*), warning (**), serious error (***), fatal error (****) and, when the LEVEL=NSTD parameter is used in the CBL JCL statement, non standard (NSTD) conditions detected by the compiler. The line number of the lines for which the compiler has output a message is shown for each type of message. The line number is the internal line number if the ILN parameter is specified in the CBL JCL statement; it is the external line number if XLN is specified. When necessary the line number is prefixed by "A." to differentiate between the alter listing and source listing.

CU Produced

If a compile unit has been produced by the compiler the message CU PRODUCED ON LIBRARY: ... is printed. If no compile unit has been produced, the compiler prints NO CU PRODUCED.

Segment List

The segment list contains a list of the internal segments produced by the compiler. For each internal segment the type and size is given. The type can be code (C..), data (.D.), linkage (..L) or code and linkage (C.L). In addition, when COBOL segment numbers are used in section headers of the Procedure Division, the COBOL segment number from the COBOL section header is printed.

The name of an internal segment is the program-name followed by a period and the internal segment number. When the program contains a Sub-Schema, separate segments are generated which contain the IDS control structures and the records selected in the schema. The name of these segments includes "-IDS" at the end of the program-name. Extra segments are generated when the DEBUG parameter is included in the CBL JCL statement (for Program Checkout Facility). The names of these segments include "_PCF" at the end of the program-name. Tables generated as a result of a USE FOR DEBUGGING statement also form an extra segment whose name is the program-name suffixed by "_DBG". For an explanation of segment numbers see "Linking", Chapter 3.

At the end of the segment list the initial size of the ring 3 stack is given. This size does not include the standard part of a stack frame, nor the parameter area. The size of each segment is also given to help in segmenting the program (see Chapter 7) and calculating working set requirements.

Run-Time Package Procedures

A list of the run-time package procedures referenced by the compiled object code is printed.

JOR (Job Occurrence Report)

A summary of the compilation (message CBL02) is printed in the JOR. This summary contains the program name, the number of error messages of each type and an indication whether the compile unit was produced. An example of this summary is as follows:

```
CBL02. SUMMARY FOR FIND-DAY: **=4 *=2 CU PRODUCED.
```

Note that, if there was no PROGRAM-ID paragraph in the program or if the source program could not be found by the compiler, the program-name in the CBL02 summary would be generated by the compiler according to the current system date and time. This type of program-name is described in "Banner Page", above.

Abnormal Compiler Termination

Abnormal termination of the compiler occurs when the compiler detects an abnormal situation. The most frequent errors are associated with an abnormal return code generated while performing a system function.

For such errors, a fatal diagnostic is printed in the compilation listing, and at least one of the following error messages is written in the JOR.

```
CBL06. error-message-text-as-in-source-listing
```

Message CBL06 will contain the same error message text as the diagnostic appearing in the source listing which is associated with the abnormal compiler termination. For example:

```
CBL06. CULIB IS FULL.

CBL06. I/O ERROR ON CULIB.

CBL06. WORKn IS FULL.

CBL06. UNRECOVERABLE DIFFICULTY DUE TO SYSTEM ERROR.
```

Message CBL06 may appear together with a message CBL01:

```
CBL01. ERROR WHILE COMPILING [LINE xxx OF] program-id

       RC=xxxxxxxx -> siuic, ic primitive ON file (ifn)
```

Other CBL01 messages can be written in the JOR without a corresponding diagnostic in the compilation listing:

```
CBL01. ERROR WHILE COMPILING program-id.

       LISTING FILE EXHAUSTED.
```

This means that the file on which the compilation listing is written (either a standard SYSOUT subfile, a PRTFILE sequential file, a PRTLIB library or the TEMP source library) is full.

Another CBL01 message without a corresponding diagnostic in the compilation listing is:

```
CBL01. ERROR WHILE COMPILING program-id

                             PRTLIB
       INVALID TYPE FOR      PRTFILE  .
                             SYSOUT
```

This means that the file to which the compilation listing is to be written is not the correct type of file. For example, a library file has been specified in the PRTFILE parameter of the CBL JCL statement, a sequential file has been specified in the PRTLIB parameter, or the library specified in the PRTLIB parameter is not an SL library.

Another CBL01 message without a corresponding diagnostic in the compilation listing is:

```
CBL01. ERROR WHILE COMPILING program-id.

       IMPOSSIBLE TO OPEN H_PR.
       H_PR UNKNOWN.
```

This means that the file to which the compilation listing is to be written cannot be opened. Either the file does not exist or an abnormal return code has occurred.

Other JOR Error Messages

```
CBL08. PROGRAM NAME TOO LONG, LISTING IS CREATED WITH THE NAME
       XXXX.

CBL09. INTERACTIVE COMPILATION IS NOT AVAILABLE ON YOUR SITE.
       PLEASE CONTACT SUPPLIER.

CBL10. THIS COBOL COMPILER IS NOT AVAILABLE ON YOUR SITE.
       PLEASE CONTACT SUPPLIER.

CBL11. THE REPORT WRITER IS NOT AVAILABLE ON YOUR SITE.
       PLEASE CONTACT SUPPLIER.

CBL12. THE DATA MANIPULATION LANGUAGE (DML) IS NOT AVAILABLE ON
       YOUR SITE. PLEASE CONTACT SUPPLIER.

CBL13. THE DATA DICTIONARY DIRECT INPUT IS NOT AVAILABLE ON YOUR
       SITE. PLEASE CONTACT SUPPLIER.

CBL14. THE TEST COVERAGE FACILITY IS NOT AVAILABLE ON YOUR SITE.
       PLEASE CONTACT SUPPLIER.

CBL15. THE SUBSCHEMA PROCESSOR IS NOT AVAILABLE ON YOUR SITE.
       PLEASE CONTACT SUPPLIER.
```

It is possible that, in unusual situations, the compiler will detect an internal problem and will issue a fatal diagnostic identifying this problem. For example:

```
9-nn COMPILER ERROR. text.
```

or:

```
x-nn IMPLEMENTATION RESTRICTION. text.
```

or possibly:

```
9-nn UNRECOVERABLE DIFFICULTY
```

When such an error is issued, the compilation output includes in addition to the listings a few printed pages for use by Field Engineering; they consist of various compiler traces and dumps.

3. Linking

LINKER is a utility which builds an executable load module from a set of compile units. These compile units may result from the compilation of programs written in different source languages. LINKER resolves all references between compile units and sets up links to COBOL run-time package procedures and system procedures which are resolved at run-time.

LINKER may be called under JCL or GCL. We will use here the JCL form. For GCL, refer to the *IOF Terminal User's Reference Manual*.

The information concerning LINKER given below is confined to the following topics:

- LINKER JCL,
- serial linkage,
- interactive use of LINKER,
- LINKER commands of interest to the COBOL programmer,
- listings of interest to the COBOL programmer.

For a more detailed description of the LINKER listings and commands see the *LINKER User's Guide*.

3.1 Segment Numbers

The system recognizes three forms of segment number during compilation, linking, program loading and execution. To avoid confusion in this and later chapters these forms of segment number are explained below:

- COBOL Segment Number. This is the segment number specified by the programmer in the section header of a COBOL program. The COBOL segment number is included in the segment list produced by the COBOL compiler.
- Internal Segment Number. This is the segment number generated by the COBOL compiler to identify the segments within a compile unit. It is this number which appears to the left of the colon in the data map, cross-reference and procedure map listings produced by the compiler. Internal segment numbers are also included in the segment lists produced by the compiler and by LINKER.
- LINKER Segment Number. This is the segment number generated by LINKER to uniquely identify each segment in the load module. It is formed from a concatenation of segment table number and segment table entry (stn.ste). LINKER segment numbers are included in the segment list produced by LINKER and in the memory dump listing.

3.2 Job Control Language

The LINKER utility is called by the extended JCL statement LINKER. The figure below shows the format of the LINKER statement.

```
LINKER  { load-module-name }
        { *                }

        [ INLIB = (input-library-description) ]

        [           { (output-library-description) } ]
        [ OUTLIB = {                               } ]
        [           { TEMP                         } ]

        [ { COMFILE = (sequential-input-file-description) } ]
        [ { COMMAND = 'command [command]...'               } ]
        [ { entry  = entry-name [COMFAC]                   } ]

        [ { PRTFILE = (print-file-description)     } ]
        [ { PRTLIB  = (print-library-description) } ]

        [ STEPOPT = (step-parameters) ] ;
```

Linker JCL Statement Format

As the LINKER statement is extended JCL, it must not appear inside a step enclosure. The following example illustrates the use of this statement:

```
$JOB...
   LIB CU       INLIB1=CU.LIB;
   LINKER       INV
                ENTRY=INV-A
                OUTLIB=LM.LIB;
$ENDJOB;
```

The LIB CU JCL statement is used to set up a "search path" for LINKER to enable it to find the referenced compile units. LINKER will look in CU.LIB for a compile unit with a member-name INV-A (specified in ENTRY=INV-A). This is used as the starting point for building the load module. The resulting load module will be stored in library LM.LIB with the name INV.

Note that, if either the LIB CU statement or the INLIB parameter is used, TEMP will not be included in the search path unless it is specified in one of these statements.

The LINKER utility produces a load module and a listing. The load module may, optionally, be stored in a temporary or a permanent library (OUTLIB parameter). The listing may, optionally, be stored in the standard SYSOUT file or in a permanent library or file (PRTLIB and PRTFILE parameters).

The following paragraphs describe the parameters which may be used in the LINKER statement. Note that the following symbolic names used in the figure "Linker JCL Statement Format" refer to standard parameter groups which are described in the *JCL Reference Manual*:

- input-library-description
- output-library-description
- sequential-input-file-description
- print-file-description
- print-library-description

These parameter groups are not described below. See the *JCL Reference Manual*.

3.3 Load-module-name Parameter

This parameter is used to specify the name of the load module to be produced by LINKER. The load-module-name must be alphanumeric. It can be up to 31 characters long.

If there is no ENTRY parameter or command in the LINKER statement, the main compile unit (at which linking starts) is assumed to have the same name as the load module. During the development of a program it is advisable to use the same name for the source program, the compile unit and the load module. It should therefore be normal practice to omit the ENTRY parameter and command from the LINKER statement.

It is not possible to use the same name in this way when several source programs and compile units are to be linked into a single load module. However, it is advisable to adopt a systematic convention for program naming. For example:

- Load module INV comprises compile units INV-A, INV-B and INV-C which were compiled from source programs INV-A, INV-B and INV-C respectively.
- Load module UPDATE comprises compile units MAIN-UPDATE and ADMIN-UPDATE which were compiled from source programs MAIN-UPDATE and ADMIN-UPDATE respectively.

An asterisk (*) may be specified instead of output-module-name. This indicates that a series of load modules are to be linked during a single execution of LINKER. See "Serial Linkage", below.

3.3.1 INLIB Parameter

This parameter is used to modify the search path used by LINKER. The input library specified in this parameter will be used as the first library in the search path. Note that, if either INLIB or the LIB CU JCL statement is used, TEMP is not included in the search path unless it is specified in the LIB CU statement.

If no LIB CU JCL statement precedes the LINKER statement and no INLIB parameter is used the search path will be:

1. TEMP compile unit library
2. SYS.HCULIB system compile unit library

If the INLIB parameter is used but no LIB CU statement is used, the search path will be:

1. Library specified in INLIB parameter
2. SYS.HCULIB

If a LIB CU statement is used but no INLIB parameter is used, the search path will be:

1. Libraries specified in LIB CU statement
2. SYS.HCULIB

If a LIB CU statement and the INLIB parameter are both used, the search path will be:

1. Library specified in INLIB parameter
2. Library specified in LIB CU statement
3. SYS.HCULIB

If both LIB CU and INLIB are used, only three libraries can be specified in the LIB CU statement. This is because the search path can only contain four user specified libraries in addition to the SYS.HCULIB which is included at the end of every search path automatically. If a fourth library (INLIB4) is specified in the LIB CU statement, it will be ignored if the INLIB parameter is also used.

3.3.2 OUTLIB Parameter

The OUTLIB parameter specifies the library in which the load module is to be stored. An output-library-description or the keyword TEMP may be used in the OUTLIB parameter.

If a library is specified, it must have been allocated previously by the LIBALLOC LM utility (see the *Library Maintenance Reference Manual*) unless the SIZE parameter is used in the output-library-description of OUTLIB. If TEMP is specified, the load module will be written as a member of a temporary system library.

If the OUTLIB parameter is omitted, this is equivalent to OUTLIB=TEMP.

The load module is stored in a library according to the following rules:

- If a load module of the same name is not already present in the library, and there is no fatal LINKER error, the load module is stored in the library with the load-module-name given in the LINKER statement.
- If a load module with the same name (normally a former version of the load module) is in the library and there is no fatal linking error, the old load module is deleted and the new one replaces it. If there is a fatal error during the linkage no load module is stored; the old load module is still usable.

When an old version exists in the load module library, it is good practice to use a new load-module-name for storing the new load module to assure retaining the old and new versions together until the new one is proven executable. Once the new load module is debugged, the old version can be deleted and the new one renamed with the old name. Deletion and renaming are done using the LIBMAINT LM utility.

Alternatively, the user can maintain a "stable" and a "development" library. The stable library should contain a working version of each program. The development library should contain the latest version of each program currently being developed and tested. Once successfully tested, programs can be moved from the development library to the stable library.

3.3.3 COMMAND and COMFILE Parameters

The COMMAND and COMFILE parameters allow the user to specify a set of commands to be obeyed by LINKER during the linkage process. The commands can be stored in a command file (COMFILE parameter) or can be specified directly (COMMAND parameter). The maximum length of a command string specified in the COMMAND parameter is 2500 characters.

The available commands are CODE, ENTRY, EXITR, FETCH, FILE, GATE, INCLUDE, LIST, LINKTYPE, MSEGAT, PLACE, REPLACE, and TASK. The commands CODE, ENTRY, INCLUDE, and LINKTYPE are described briefly below as they are of special interest to the COBOL programmer. For a full description of all commands, see the *LINKER User's Guide*. Commands must be separated by one or more spaces or by a comma and zero or more spaces. The final command must be followed by a semi-colon (;).

The COMMAND and COMFILE parameters can also be used to specify a series of load modules to be linked during a single execution of LINKER. See "Serial Linkage", below.

3.3.4 ENTRY Parameter

This parameter specifies the entry-name to be used as the start point for program execution. The compile unit containing this entry-name will be the first one used by LINKER in building the load module. It can be omitted if the entry-name is the same as the load-module-name.

Entry-name must be the value written in the PROGRAM-ID paragraph of the source program and must comply with the COBOL rules for this paragraph. COMFAC indicates that the load module to be generated is derived from MCS COBOL. See the appropriate manual for details.

When the ENTRY parameter is used the COMMAND and COMFILE parameters must be omitted.

3.3.5 PRTFILE Parameter

This parameter requests that the LINKER listing be appended to a permanent SYSOUT file for printing or processing at a later stage by, for example, WRITER or any text handling program or utility. Otherwise, the listing is printed at the end of the job and no permanent copy is kept.

If the PRTFILE parameter is used, LINKER adds the listing to the SYSOUT file in append mode. The PRTLIB parameter, on the other hand, replaces any previous listing of the same name (see below). In either case, the LINKER listing will not be printed automatically. Printing can be requested later by using a WRITER JCL statement. Only the JOR will be printed at the end of job execution.

When serial linkage is requested (see "Serial Linkage" below) and the PRTFILE parameter is used, all listings are stored in a single file.

3.3.6 PRTLIB Parameter

This parameter is similar to PRTFILE except that the listing will be stored in a member of the library specified in the PRTLIB parameter. If several programs are linked in series when the PRTLIB parameter is used, the listing for each program will be stored in a separate library member. Each library member will be given a name comprising the load-module-name suffixed by "_K". It replaces any member of the same name. See "PRTFILE Parameter" above.

3.3.7 STEPOPT Parameter

The STEPOPT parameter can be used to specify one or more of the parameters included in the STEP JCL statement (see the *JCL Reference Manual*). However, the following cannot be included in the STEPOPT parameter for LINKER:

- load-module-name
- TEMP, SYS or input-library-description
- the ALL option of the DUMP parameter
- the OPTIONS parameter

3.4 Serial Linkage

LINKER can link a series of load modules during a single execution. In order to do this, an asterisk (*) is specified in the LINKER statement instead of load-module-name. The only other parameter which is permitted in such a LINKER statement is either COMMAND or COMFILE, and it is mandatory for serial linkage. The COMMAND or COMFILE parameter is used to name each load module to be linked and to specify the commands applicable to each load module. For each load module to be linked the load-module-name must be specified, followed by the associated commands (if any). Each group of load-module-name and commands, including the last, must be followed by a semi-colon (;).

EXAMPLE:

```
LINKER *
   COMMAND='LOAD-MODULE-1,  ENTRY=ALPHA;
            LOAD-MODULE-2;
            LOAD-MODULE-3,  ENTRY=BETA;';

LINKER *
   COMFILE=*CMD;
$INPUT CMD;
   LOAD-MODULE-1,  ENTRY=ALPHA;
   LOAD-MODULE-2;
   LOAD-MODULE-3,  ENTRY=BETA;
$ENDINPUT;
```

Note that the asterisk (*), when it is specified in the LINKER statement, does not have the same meaning as the star convention used in the SOURCE parameter of the CBL statement.

❑

3.5 Interactive Linkage

LINKER may be executed from an IOF terminal. The normal LINKER JCL statement is used. LINKER operates as in batch mode, except that the LINKER listing is not directed to the SYSOUT file. The listing, unless otherwise specified, is stored in the TEMP source library, and must be printed via the WRITER JCL statement. The print subfile is stored in TEMP with the name lmn_K, where lmn is the load module name. The following example illustrates the use of LINKER at an IOF terminal.

```
S: LK MYPROG;
>>>14:31 LINKER 90.00
WORKING ON:MYPROG
LKOO.(90.00) SUMMARY FOR MYPROG       NO ERROR DETECTED
                                   OUTPUT MODULE PRODUCED
<<<14:33
S: WRITER (TEMP, SUBFILE=MYPROG_K);
```

3.6 LINKER Commands

3.6.1 CODE Command

The format of the CODE command is:

```
        { OBJA  }
CODE = {        }
        { OBJCD }
```

The LINKER checks that the CUs used to build the load module are of a type compatible with the class indicated in the CODE command. If no CODE command is specified, the default is OBJA when linking under GCOS 7-V3A, OBJCD when linking from GCOS 7-V3B.

3.6.2 ENTRY Command

The format of the ENTRY command is:

```
ENTRY = member-name
```

The ENTRY command specifies the entry-name to be used as the start point for program execution. This command is used in the same way as the ENTRY parameter except that COMFAC cannot be specified in the ENTRY command. When the COMMAND or COMFILE parameter is used in the LINKER statement, the ENTRY parameter cannot be used. The ENTRY command should be used instead.

3.6.3 INCLUDE Command

The format of the INCLUDE command is as follows:

```
           {(compile-unit-name  compile-unit-name  ... ) }
           { INLIB                                       }
           { INLIB1                                       }
INCLUDE = { INLIB2                                        }
           { INLIB3                                       }
           { INLIB4                                       }
```

The INCLUDE command is used to incorporate compile units referred to in the COBOL "CALL identifier" statement. This form of the statement does not specify a program name at compilation time, so LINKER cannot automatically incorporate the required compile unit into the load module. This has to be done by the programmer by using the INCLUDE command, which names all the compile units which may possibly be named in the data item referenced by CALL.

If INLIB is specified in the INCLUDE command, the contents of the library specified in the INLIB parameter of the LINKER JCL statement are included. If INLIBn is specified in the INCLUDE command, the contents of the first, second, third or fourth library specified in the preceding LIB CU JCL statement are included, depending upon the value of n.

There may be several INCLUDE commands for one load module. A list of included compile units will be printed by LINKER.

3.6.4 LINKTYPE Command

The format of the LINKTYPE command is:

```
LINKTYPE = MAM
```

This command indicates the type of linkage to be performed. If this command is omitted, a standard user load module is produced. If LINKTYPE=MAM is specified, a load module is produced which can interface with the Basic Message Access Method. LINKTYPE=MAM should be used for load modules which are derived from MCS COBOL source programs. See the appropriate manual for details.

Only one LINKTYPE command can be used per load module.

3.7 Printer Output

The following paragraphs briefly describe the printer output produced by LINKER. For a more detailed description of the printer output see the *LINKER User's Guide*. A sample LINKER listing is provided in Appendix A, together with a listing of the COBOL program with which it is associated, The LINKER listing is composed of the following sections.

- Banner page and LINKER commands listing. All commands included in the COMMAND parameter or command file of the LINKER JCL statement are listed in the LINKER commands listing.
- Included compile units (if any). Details are printed for each compile unit included in the load module as a result of using the INCLUDE command.
- Task listing. There is a task listing for each task in the load module.
- Group information. This listing contains general information about the entire process group. The listing is in two parts: global segment list and segment list. The segment list is the most useful part of the LINKER listing for the programmer and is described in more detail below.
- Cross-reference listing (if any). The cross-reference listing is only produced if the LIST=XREF LINKER command is used. In this listing, for each external name, the location of each reference to the name is shown.
- Linkage report and end page. The linkage report gives a summary of the error messages generated by the LINKER. This report is described below. The end page simply contains the percentage of the total library space used by all load modules currently in the library.

3.7.1 Segment List

The segment list, contains an entry for each segment in the load module (including global segments). See Chapter 4 for a sample segment list. The segment list is the most useful part of the LINKER listing for the following reasons:

- The LINKER segment number and internal segment number are shown for each segment generated directly from user source code. The relationship between these segment numbers has to be known when tracing the origin of abnormal step terminations and in analyzing memory dump listings (see Chapter 4).
- The size of each segment in bytes is shown. This may be useful when segmenting a COBOL program (see Chapter 7) and when estimating working set requirements for program execution.

The headings and information in the segment list which are of use to the COBOL programmer are as follows.

SEG	LINKER segment number in the form stn.ste.
IN CU	The name of the segment as it appears in the segment list of the COBOL summary page. Segments which are generated directly from user source code have a name of the form cun.isn where cun is the compile unit name and isn is the internal segment number.
TYPE	This indicates that the segment contains code (C..), data (.D.) or linkage information (..L). Combinations of these types are also possible (e.g., C.L when the CODAPND parameter is specified in the CBL JCL statement).
SIZE	This indicates the size of the segment, in bytes.
MAXSIZE	This indicates, in the case of a variable length segment, the maximum size of the segment, in bytes. Note that SIZE and MAXSIZE values are needed for working set calculations.

3.7.2 Linkage Report

The first line of the linkage report contains either "ERRORS DETECTED" or "NO ERRORS DETECTED". If no errors have been detected, the linkage report ends immediately after printing the line "OUTPUT MODULE PRODUCED ON LIBRARY library-name". However, if errors have been detected, a summary of errors is now printed.

The summary of errors comprises one or more of the following lines:

- WARNINGS (SEV.1): n
- ERRORS SEVERITY 2: n
- ERRORS SEVERITY 3: n
- ERRORS SEVERITY 4: n

where "n" is the number of errors in each category. If there are any errors of severity 4 (fatal), an output load module will not be produced and the linkage report will end with the line "NO OUTPUT MODULE PRODUCED". If there are no errors of severity 4, the linkage report will end with the line "OUTPUT MODULE PRODUCED ON LIBRARY library-name".

The end page simply contains the percentage of the total library space used by all load modules currently present in the library.

3.7.3 Error Messages

Each error detected at linkage time saves at least one test execution of the user program. In order to detect as many errors and inconsistencies as possible, LINKER carries out checks on the interface between linked procedures. For example, the arguments of a calling and called procedure must be compatible in number and attributes; external data declared in different procedures must have consistent attributes.

When an error is detected, LINKER outputs a message at the point in the listing at which the error occurred. Error messages have one of the following formats:

```
**** WARNING#nnnn              message-text

**** ERROR #nnnn SEVERITY s    message-text
```

where "nnnn" is the message number, "s" is the severity and "message-text" is an explanation of the situation. Severity "s" may have a value of 2, 3 or 4. (Severity 1 corresponds to a WARNING). Severity 4 is fatal and no load module will be output. The total number of error messages of each severity is given in the linkage report.

4. Executing

This chapter introduces the debugging facilities and describes the use of these facilities for program testing. The analysis of user program memory dumps is also discussed. Various types of abnormal step termination are described and hints are provided to help the programmer diagnose their cause.

NOTE:
For an explanation of COBOL segment number, internal segment number and LINKER segment number, see Chapter 3, "Linking".

4.1 Program Debugging

The programmer has two tools at his disposal for program debugging:

- The insertion of debugging code into the COBOL source program. This is a purely COBOL tool and does not rely upon any facility external to the COBOL program.
- The use of the Program Checkout Facility. This is a facility external to COBOL and does not have to be requested within the COBOL program.

The use of these tools is discussed in the following paragraphs.

4.1.1 Debugging Code

The following types of debugging code can be inserted in the COBOL program:

- One or more debugging sections in the Procedure Division Declaratives. Such a section includes a USE FOR DEBUGGING statement which specifies the data-names, procedure-names etc., that are to be monitored by the remainder of the section. The remainder of the debugging section contains normal Procedure Division statements, typically DISPLAY, which are executed when the data-names, procedure-names etc., are referenced. Debugging section code can access a special register, DEBUG-ITEM, that contains information such as: the internal line number of the line for which the USE FOR DEBUGGING section is invoked, the value and data-name, procedure-name etc. of the data item, altered paragraph etc. Note that, among other things, a USE FOR DEBUGGING section can be invoked each time a data-name is referenced; this facility is not available when using the Program Checkout Facility alone.
- One or more debugging lines anywhere in the program after the OBJECT-COMPUTER paragraph. Such a line is identified by a "D" in the indicator area (column 7). A frequent practice is to insert, in the Procedure Division, DISPLAY statements which will display the contents of significant variables at various stages of program execution. In fact, any COBOL procedures may be coded as debugging lines; the only requirement is that the program be logically consistent both with and without such code.

If the WITH DEBUGGING MODE clause is present in the SOURCE-COMPUTER paragraph of the program, the debugging code will be compiled as normal program code. If the WITH DEBUGGING MODE clause is absent the debugging code will be treated as comment and will not be compiled.

The presence or absence of the WITH DEBUGGING MODE clause can be overridden by the CBL JCL statement parameters DEBUGMD and NDEBUGMD. If DEBUGMD is specified, the program is compiled as if a WITH DEBUGGING MODE clause was included in the program. If NDEBUGMD is specified any WITH DEBUGGING MODE clause is ignored and debugging code is not compiled.

However, if the debugging code is compiled, the USE FOR DEBUGGING sections will only be executed if the DEBUG parameter is included in the STEP JCL statement. If the DEBUG parameter is absent the USE FOR DEBUGGING sections have no effect upon program execution.

The presence or absence of the DEBUG parameter has no effect upon debugging lines (containing a "D" in column 7).

When a load module consists of more than one COBOL program and the DEBUG parameter is used in the STEP JCL statement, all USE FOR DEBUGGING sections of all programs are activated.

However, one can deactivate the sections of one or more of these programs by using the Program Checkout Facility (PCF) as follows:

```
C ?stn.ste.30#X2 = "000a"X;
```

or

```
A IN program-name;
C :0.30#X2 = "000a"X;
```

A full explanation of the APPLY (A) and CHANGE (C) commands can be found in the *Program Checkout Facility User's Guide*. "stn.ste" gives the segment table number and the segment table entry corresponding to the internal segment number 0 of the compile unit whose USE FOR DEBUGGING sections are to be deactivated. stn and ste comprise the LINKER segment number which is defined in Chapter 3 together with the internal segment number. The relationship between the LINKER segment number and the internal segment number is described under "Dump Analysis", below.

"a" must be zero for complete deactivation of the USE FOR DEBUGGING sections.

The USE FOR DEBUGGING sections may be partially activated. In this case the above PCF command must be used with "a" taking a value of 1 or 2. These values have the following significance:

1. The USE FOR DEBUGGING sections are activated only "ON procedure-names".
2. The USE FOR DEBUGGING sections are activated only "ON identifiers, cd-names and file-names".

Programs which are not referenced in the above commands have all their USE FOR DEBUGGING sections activated when the DEBUG parameter is used in the STEP JCL statement.

USE FOR DEBUGGING sections can be activated, partially activated or deactivated dynamically by using the "AT" and/or "IF" options of the above PCF commands. Full activation of all USE FOR DEBUGGING sections in all programs can be achieved by using the above PCF command with "a" having a value of 3. Note that if the DEBUG parameter is used in the STEP JCL statement then, unless commands specify otherwise, all USE FOR DEBUGGING sections are activated in all programs when execution starts.

4.1.2 Program Checkout Facility

The Program Checkout Facility (PCF) is a diagnostic system which (if requested) is executed in parallel with a user program being tested. PCF may be used to monitor the user program in the following ways.

- The flow of program control can be traced through specified points in the program. Each time control passes through such a point, PCF records this fact.
- The values of specified data items can be changed when control reaches specified points within the program.
- The values of specified data items can be dumped when control reaches specified points within the program.
- Procedures and data can be referred to using symbolic or effective addressing.
- Commands can be applied to selected compile units.
- Commands can be made conditional upon the value of specified data items.

The type of monitoring to be done by the PCF is specified by the programmer in a file of PCF commands. The commands used to request the above monitoring for example are TRACE, CHANGE and DUMP.

The use of the PCF will not be described further in the current manual. See the *Program Checkout Facility User's Guide* for further details. However, the following paragraphs discuss the JCL required in order to run the PCF.

The PCF is requested by including the DEBUG parameter in the STEP JCL statement of the user program. In addition a sequential file of PCF commands must be created and must be assigned to the job step with an internal-file-name H_DB.

In addition to the above JCL it is advisable to include the DEBUG parameter (not to be confused with the DEBUGMD parameter) in the CBL JCL statement. This parameter causes the compiler to build a table of all the source names in the program, with a record of the name type (data-name, paragraph-name etc.) and the generated segment address. This table is then stored in the compile unit and is incorporated in the load module by LINKER. It is possible to use the PCF in the absence of this table. However, if this is done, the user must specify the actual memory addresses when referring to the code and data in the load module (effective addressing). The presence of the table enables the programmer to refer to data and code by the names used in the COBOL source program, or code by the line number (symbolic addressing). However, the size of these tables should be borne in mind (about 50 bytes per source line). A segment containing these tables is generated for each 240 lines of source code (approximately).

The DEBUG parameter in the STEP JCL statement, in addition to requesting the PCF, has a special effect on two exceptions (see "Exception Messages" below).

These are:

```
EX01.  EXCEPTION 09-01:  ILLEGAL DECIMAL DATA...
EX01.  EXCEPTION 17-02:  OUT OF ARRAY RANGE...
```

When the DEBUG parameter is used together with the PCF commands RECOVER ILLDEC and RECOVER SUBSCRIPT these exceptions disappear and the step is not abnormally terminated. Instead, the error is reported in the PCF report and action is taken to compensate for the error. These two exceptions usually occur more often than any others during program debugging and their suppression can avoid numerous unproductive test executions.

When a step is launched from, or is running on an interactive terminal using IOF, the H_DB internal-file-name assignation may not be used. In that case, PCF commands and reports will be taken from, or sent to, the terminal. This enables the programmer to interactively change PCF commands, depending on PCF outputs.

4.2 Dump Analysis

A memory dump of the user program can be obtained if the DUMP parameter is included in the STEP JCL statement. The dump is only printed if the program terminates abnormally.

It is recommended that the DATA option be used with the DUMP parameter. For example:

```
STEP PROG1, TEMP
     DUMP = DATA;
```

The DATA option will produce a dump of data segments only: code and linkage segments will not be dumped. They are not required for user programs.

In discussing dump analysis the following terms will be used:

- task;
- process;
- process group;
- protection ring.

These terms are used because they are needed to describe the information that appears in a program dump. However, these terms are not defined in the current manual; they are defined in the *LINKER Manual*.

4.2.1 Structure of the Dump Listing

The various elements of a memory dump are listed in the following order:

- the PCS (segment 8.0);
- the PCB;
- the stack;
- the segment the exception occurred (if an exception occurred);
- type 0 segments;
- the PGCS;
- type 2 segments;
- type 3 segments (except segment 8.0);
- type 1 segments.

See Figure 4-1 for an example of the first page of a dump.

The first segment is the Process Control Structure for the process. It is preceded by the following heading:

```
***********
****PCS****
***********
```

See Figure 4-1 for an example of this heading. The Process Control Structure contains the Process Control Block which contains a dump of the stacks used by the process. One of these stacks is of great interest to the programmer and can be used to isolate the part of the user program that was active when the abnormal termination occurred.

The dump of the Process Control Block starts with the following heading:

```
******PCB******
```

Type 2 segments are those which are shared by all processes of the process group. The first segment in this part of the dump listing is the Process Group Control Structure for the step.

Each segment in the dump has a two line header similar to the following:

```
/J=02/P=00/ /STN=09/ STE=01/ SEGMENT DESCRPT: 9800F81B 42000000
SEGM.HEADR: 000F8170 000F81C0 00000200 0001DDC8 01032E41 01098901...
```

The only items of interest to the user programmer are the values shown for STN (segment table number) and STE (segment table entry). Type 2 segments in the dump listing include the following:

- File buffers.
- Physical channel program segments.
- Data Management control structures.
- Job control structures.

The only segments of interest to the user programmer in this part of the dump listing are those containing file buffers.

Type 3 segments comprise the data segments (and code and linkage segments if DUMP = ALL was specified) which make up the COBOL program proper.

```
********************************************************************************************************************************
********************************************************************************************************************************
**** GCOS7                                                                                                                  ****
****                                                    D U M P                                                             ****
****                                                                                  VERSION: 70.00  DATED: JUN 29, 1985 ****
********************************************************************************************************************************
********************************************************************************************************************************

COPYRIGHT (C) BULL S.A. 1986          S

DUMP  J=12    P=00     TERM.MSG 208404CS

***********
****PCS****
***********

/J=12/P=00/  /STN=08/ STE=00/  SEGMENT  DESCRPT:  9C00E293 C2000013

000000 0E2930 00090100 181A0328 00000000 00000C22  00000900 129220C0 00000000 00000000  @@@@@@@@@@@@@@@@@@@@k@@@@@@@@@@
000020 0E2950 00000000 007800A0 005C0000 0068006E  00001400 81506005 00004000 00008000  @@@@@@@@@@*@@@@@>@@@@a&-@@@ @@@@@
000040 0E2970 09040010 081C0000 0902007C C0390001  1A000000 00000000 7FFFFED8 FFFFFFFF  @@@@@@@@@@@@@@@@@@@@@@@@@"@@Q@@@@
000060 0E2990 FFFFFFFF FFFFFFFF FF811D00 01E00000  00000000 00000000 0000A8D2 0C9D9000  @@@@@@@@@a@@@@@@@@@@@@@@@@yK@@@@
000080 0E29B0 00000000 833A5000 00000003 6CA5A000  00000000 4189B000 00000000 06F94000  @@@@c@&@@@@@%v@@@@@@@i@@@@@@@=@@
0000A0 0E29D0 80C06000 90000000 090C02E0 07000000  0700E2A7 0800E2A9 0C6E0010 081A0DA0  @@-@@@@@@@@@@@@@@@Sx@@Sz@>@@@@@@
0000C0 0E29F0 1CE90E2C 00000000 08130000 08140000  081B0000 181A0D88 081A0DA0 19140290  @Z@@@@@@@@@@@@@@@@@@@@@@h@@@@@@@@
0000E0 0E2A10 08000080 181A0CA9 1CE904D8 181A0D88  0CE90DD6 081A0DA0 00000000 00000020  @@@@@@@z@Z@Q@@@h@Z@0@@@@@@@@@@@@
000100 0E2A30 191401D8 00000000 0000A000 1C620520  00000035 0000001F 00000422 00000000  @@@Q@@@@@@@@@@@@@@@@@@@@@@@@@@@@
000120 0E2A50 191401D8 191401D8 00000000 00000006  29C30000 00000000 00000000 00000000  @@@Q@@@Q@@@@@@@@@@C@@@@@@@@@@@@@

******PCB******

               PMW0: 80C06000    PMW1: D0000000     PMW2: 090C02E0    PMW3: 07000000

               ASW0: 0700E2A7    ASW1: 0800E2A9      EXW: 0C6E0010     SKW: 081A13B9

                ICW: 0C170CF6    SBW0: 08130000     SBW1: 08140000    SBW2: 081B0000

                BR0: 08130138     BR1: 08130158      BR2: 08010000     BR3: 08000000

                BR4: 0D670000     BR5: 40000000      BR6: 0DF30028     BR7: 0C170050

                GR0: 00000000     GR1: 12001200      GR2: 00000002     GR3: 0AA70030

                GR4: 00000000     GR5: 00000001      GR6: 00000001     GR7: 00001200

                XR0: 000E2930     XR1: 000000A0      XR2: 00000000     XR3: 0AA70030

                XR4: 00000002     XR5: 00000008      XR6: FFFFFFF8     XR7: 00000000
```

Figure 4-1. First Page of DUMP

```
RING 0 STACK STN=08   STE=13   SEGDESCR 9C010F1C  02000115
 *STACK FRAME 00001
  *WRKAREA
     0090    FF1200FF FFFFFF00 0000020C 00000012  00640010 0000FFFF FFFF0000 00000000   @@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@
     0080    00000000 00000000 00000000 00000000  00000000 0000A8CE A3B86000 0000       @@@@@@@@@@@@@@@@@@@@@@@@y@t@-@@@
  *SAVAREA
     0000        SAM= 0FFF7FF0
     00D4      BR0-7: 08130070 08130090 400E28D0 400E2930 00670000 40000000 08130070 0AA70048
     00F4      GR0-7: 00000000 00001200 00000002 00000000 ******** 00000000 00000001 00001200
     0110      XR0-7: 000E2930 000000A0 00000000 38120082 000004F1 00000008 FFFFFFF8 00000000
     0130        STR: 80000000
     0134        PTV: 081300CE
     0138        PSA: 081300D0
     013C        ICC: 0AA7067A

  *COMAREA
     0140        NBP= 00000014
     0144    08130091 0AA7008C 0AA70054 08130097  0AA700E5
 *STACK FRAME 00002
  *WRKAREA
  *PAGE=081A0000
     0000    208404C5 381C0168                                                   @d@E@@@@
  *SAVAREA
     0008        SAM= 0FFF7FF0
     000C      BR0-7: 381C0168 08130000 08000000 08120082 081C00FC 081C0000 381C0178 0C730020
     002C      GR0-7: 08130000 00000901 00000001 00000901 ******** 00000000 FFFFFFFA 00001200
     0048      XR0-7: 38120082 12000901 00000000 38120082 000004F1 00000008 FFFFFFF8 FFFFFFFA
     0068        STR: 80000000
     006C        PTV: 08130008
     0070        PSA: 08130008
     0074        ICC: 0C73048A

  *COMAREA
     0078        NBP= 00000014
     007C    0C730057 08130000 0C730055 08130004  0C730058
```

Figure 4-2. Ring 0 Stack

```
RING 3 STACK STN=08   STE=1C   SEGDESCR 9C0F0C89  FE00007F
 *STACK FRAME 00001
  *WRKAREA
     00A0    0FFFFFF0 381C0000 081C0000 0810BC08  081C0018 FFFFFFFF 08000000 081C008C   @@@0@@@@@@@@@@@@@@@@@@@@@@@@@@@@
     00C0    0C220008 081C0000 00000000 00000000  09070000 00000000 00000000 00000000   @@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@
     00E0    00000000 00000000 00000000 00000000  00000000 00000000 000000             @@@@@@@@@@@@@@@@@@@@@@@@@@@@
  *SAVAREA
     00FC        SAM= 0FFFFFF0
     0100       BR0-7: 381C008C 081C00A0 08110000 381100E4 0812000A 08000000 381C008C 08100010
     0120       GR0-7: 081C00A0 081201D6 00000000 09070000 00000000 00000000 00000000 000007FE
     0140       XR0-7: 00000000 00000000 00000000 00000000 00000000 00000000 00000000 00000000
     0160         STR: 7C000400
     0164         PTV: 081C00FB
     0168         PSA: 081C00FC
     016C         ICC: 38120082
  *COMAREA
     0170        NBP= 00000010
     0174    000004F1 12000901 00000000 08F10000
 *STACK FRAME 00002
  *WRKAREA
  *PAGE=381C0000
     0000    00000000 08100008 38120082 12000901  00000000 FFFFFFFF 00000000 FFFFFFFF   @@@@@@@@@@@b@@@@@@@@@@@@@@@@@@@@
     0020    0C220008                                                                  @@@@
  *SAVAREA
     0024        SAM= 0FFF7FF0
     0028       BR0-7: 381C0000 081C0000 08100008 081C0018 FFFFFFFF 08000000 00000000 0C220008
     0048       GR0-7: 081C0000 00000000 00000000 09070000 ******** 00000000 00000000 00000000
     0064       XR0-7: 00000000 00000000 00000000 00000000 00000000 00000000 00000000 00000000
     0084         STR: 3E000000
     0088         PTV: 081C0024
     008C         PSA: 081C0024
     0090         ICC: 3C2200CA
  *COMAREA
     0094        NBP= 80000008
     0098    081C0018 FFFFFFFF


/J=12/P=00/  /STN=08/ STE=12/  SEGMENT  DESCRPT:  980F19FD FD000020

SEGM.HEADR:   00F19F60 00F1A1E0 00011200 000E2B40  01030B41 04009504 00000021 00000000

000000 F19FD0 C2000008 0080BC40 01F2752F 00289701  000ECC01 000E00F1 CD01000E 000C9601  B@@@@@@ @2@@@@p@@@@@@@@1@@@@@@o@@
000020 F19FF0 000E8270 07FE4372 0000B880 00086272  00005217 003C7212 00E47582 00E4BD30  @@b@@@@@@@@@@@@@@@@@@@@@@@U@@@U@@
000040 F1A010 0020754F 00404E6C 00004E70 00048200  0FFF3C60 3C703367 B8800006 1800D208  @@@!@ +%@@+@@@b@@@@-@@@@@@@@@@K@
000060 F1A030 000C0802 00D0D322 00D62107 002DB860  0014D202 00D00602 00D2FA27 00442202  @@@@@@L@@@0@@@@@-@@K@@@@@@K@@@@@@
000080 F1A050 00D0FA02 00D62401 0019C422 00D42080  5E870018 F0028064 34010019 FA020000  @@@@@0@@@@D@@M@@;g@@0@@@@@@@@@@@@
0000A0 F1A070 45010021 F1070035 15010021 FA010021  5801002E F4070046 3801002E FA01002E  @@@@1@@@@@@@@@@@@@@@4@@@@@@@@@@@@
0000C0 F1A090 88010026 FA010019 49010036 F0010026  89010036 FA210036 9A0200D8 F3070046  h@@@@@@@@@@@0@@@i@@@@@@@@@@Q3@@@
```

Figure 4-3. Ring 3 User Stack

4.2.2 The Stack

For each protection ring in each process there is a "stack". For a normal program there are three stacks. The stack is used each time the COBOL program executes a CALL statement. At that time the addresses of arguments, the contents of registers and the contents of the instruction counter are loaded onto the stack. When the called program is entered, data items declared in INITIAL programs may be allocated in the stack. The stack is a last-in-first-out data structure. This means that data pertaining to the last CALL statement executed is at the logical top of the stack. Data pertaining to the last-but-one CALL statement executed is next in the stack, and so on. The stack is also used in the same manner when an exception occurs or when the code generated by the compiler or contained within COBOL run-time package procedures executes an instruction equivalent to a COBOL CALL statement.

Therefore, after an abnormal termination, the relevant stack will point either to the instruction following the last CALL (or equivalent) executed or the instruction at which an exception occurred.

The stack for each ring starts with a heading such as:

```
RING 3 STACK STN=08 STE=1C SEGDESCR 9C0F0C89 FE00007F
```

The ring 3 stack is the one relevant to the user program. See Figures 4-2 and 4-3 for examples of a ring 0 and a ring 3 stacks.

The stack is separated into stack frames by headings such as:

```
*STACK FRAME 001
```

Each frame is associated with an individual CALL statement (or equivalent) or with an exception. The data for the latest CALL or an exception is in stack frame 001 of the ring 3 stack. In this stack frame the value of the instruction counter is printed in the *SAVAREA next to the characters "ICC:".

See below for a sample stack frame 001.

```
*STACK FRAME 001
 *WRKAREA
    00A0    0FFFFFF0 3300008C 030000A0 08110000 0300001C FFFFFFFF
 *SAVAREA
    00C0       SAM=   0FFFFFF0
    00C4    BR0-7:    3300008C 030000A0 08110000 0811006E 19190134
    00E4    GR0-7:    00000000 00000014 00000001 00000003 B5C20000
    0104    XR0-7:    00000009 00000010 00000004 00000000 000000F1
    0124       STR:   BE000000
    0128       PTV:   030000BE
    012C       PSA:   030000C0
    0130       ICC:   38120448
 *COMAREA
   0134        NBP=   00000010
   0138   000000F1    04000901 00000000 08F10000
```

Sample Stack Frame 001 Dump

In this example the line containing the instruction counter is 4 lines from the end of the stack frame and reads as follows:

```
   0130  ICC:    38120448
```

The instruction counter is of the form rneeaaaa, where:

- `r` is not relevant;
- `n` is the segment table number (stn)
- `ee` is the segment table entry (ste)
- `aaaa` is the address relative to the start of the segment (sra)

The stn, ste and sra point to the machine instruction which was executing when the program terminated. Normally, the stn, ste and sra will also be printed in an exception message in the JOR (see "Exception Messages", below). This is useful when no dump has been produced. Even if there is a dump, the stn, ste and sra can be found more conveniently from the exception message.

The machine instruction indicated by the stn, ste and sra normally corresponds to a COBOL statement in the source program. The following paragraphs explain how to find the line in the source listing which was being executed when the program terminated. Note that the stn, ste and sra might not point to an instruction in the source listing. This is the case when the program terminates while executing an instruction in the prologue or the epilogue of the COBOL program or in one of the procedures of the COBOL run-time package.

The segment table number and segment table entry when written in the form stn.ste make the LINKER segment number shown in the segment list produced by LINKER. For example, the instruction counter in the above example gives a LINKER segment number 8.12. From the LINKER segment list below it can be seen that the segment with this LINKER segment number (SEG.) is a code segment (TYPE is C.. in listing) from internal segment number 4 of the compile unit CPD208.

```
                                SEGMENT LIST

SEG.  IN CU.ISN           TYPE  SH RF RD WR EX WP EP  G  S  SIZE

08.00 PCS                 .D.   3  3  3  0  0  W              336
08.01 NPCS                .D.   3  3  3  1  0  W               32
08.03 ASL.3               .D.    3  3  1  0  0  W                 16
08.10 CPD208.0            ..L   3  3  3  3  3                 528
08.11 CPD208.1            .D.   3  3  3  3  3  W             1920
08.12 CPD208.4            C..   3  3  3  3  3     E          2992
08.13 H_CBL_DRTP          .D.   3  3  3  3  3  W              608
08.14 __BLANK             .D.   3  3  3  3  3  W               16
```

Sample Linker Segment List

The procedure map listing produced by the COBOL compiler for the indicated compile unit (in this example CPD208) should be consulted in order to find the line number of the COBOL source line following the relevant CALL or that at which the exception occurred (see "Exception Messages", below). The addresses in the procedure map listing are formed by concatenating the internal segment number (suffixed to the compile unit name of the LINKER segment list) with the sra obtained from the instruction counter. For example, the LINKER segment number 8.12 in the above example indicates internal segment number 4 of compile unit CPD208. The internal segment number (4) should be concatenated with the sra (00448) from the instruction counter. Thus the address to look for in the procedure map listing is 4:00448. This address will normally lie between two of the addresses shown on the procedure map listing. The earlier address should be used as this is the start address of the compiled object code.

Information concerning parameters specified in a CALL (or equivalent) is printed in the *COMAREA section of the ring 3 stack frames. The first word of the *COMAREA contains the number of bytes (hexadecimal) in the list of parameter addresses which follows. The addresses are of the same form as the instruction counter (stn.ste.sra.). These addresses point to the locations in the dump at which each parameter starts (the number of bytes in each parameter is not given). The method of finding these locations in the dump is described below (Data Division Variables).

In order to find the data-name of a parameter for which an address is given, the following should be done. The internal segment number should be obtained from the stn and ste, as described above, and should be concatenated with the sra. The resulting address should be searched for in the data map or cross-reference listing to find the data-name.

4.2.3 Data Division Variables

The value of any Data Division data item at the time of an abnormal termination can be found from the dump listing.

In order to do this a COBOL compilation listing is needed which includes at least one of the following (see Chapter 2 for the associated CBL JCL parameters):

- cross-reference listing (alphabetic order);
- cross-reference listing (declaration order);
- data map and procedure definition listing.

The address of the data item may be found in one of the above listings by referring to the associated data-name. Consider the following line from a data map listing:

```
LN NAME                      PN ADDRESS DESCRIPTION  DEF.

02 A-KRBAB (KR-KRBAHEADER)    1:000EC DISPLAY 9    84
```

The address of data item A-KRBAB in this listing is 1:000EC, where 1 is the internal segment number of the segment containing the data item and 000EC is its address relative to the start of the segment (sra).

To find the address of the data item in the dump the internal segment number must be converted into a LINKER segment number.

The LINKER segment list of the abnormally terminated load module must be consulted. See the sample segment list above. From this sample it can be seen that the LINKER segment number (SEG.#) corresponding to internal segment number 1 is 8.11. One can verify that this segment is a data segment from the TYPE which is ".D.".

This LINKER segment number comprises a segment table number (stn) and segment table entry (ste) in the form stn.ste. That is, if the LINKER segment number is 8.11, the stn is 8 and the ste is 11. The segment containing the required data item can be found by looking for the segment header containing the correct stn and ste.

A sample segment with a header containing stn = 08 and ste = 11 is shown in Figure 4-4. Each line of a segment dump shows the values held in 8 consecutive words of memory. (A word comprises 4 eight bit bytes.) The first part of the line shows the hexadecimal representation of each word. The second part of the line shows the EBCDIC representation. At the extreme left of each line are two columns of memory addresses. The first column is the address relative to the start of the segment (sra). The second column is the address relative to the start of memory.

The addresses in the first column should be searched, for the address of the data item as specified in the COBOL compilation data map or cross-reference listing. In the above example this address is EC. The addresses in the segment dump are those of the leftmost word on each line so the last digit of this address is always zero. So, if the address EC is being looked for, the line beginning 00E0 should be selected and the byte with address EC will be the first byte in the fourth word from the left (i.e., the 13th byte from the left).

When a data item is allocated in the stack, the corresponding line of the data map listing gives its address in one of the following two formats:

- "?BR1.offset" for an item allocated in the first stack area
- "?BR1.base->offset" for an item allocated in an area other than the first one.

These data items are allocated in the stack frame portion called "*WRKAREA". When the address of a data item is of the form "?BR1.offset", "offset" is its offset in *WRKAREA. When the address is of the form "?BR1.base->offset", "base" is the offset in *WRKAREA of a word that contains an address; "offset" must be added to this address to get the address of the data item.

The method described above allows to find Working-Storage items in a dump listing. The address of based data items (a data item described in a Linkage Section and which is not part of any parameter) is shown in the data map listing in the form "pointer-address->offset", in which "pointer-address" is the address of a pointer data item which contains the address to which "offset" must be added to get the address of the based data item. The pointer data item can be found using the method described for Working-Storage data items.

4.2.4 General Information

The following information may also be of interest:

1. The segment whose name is program-name.1 (usually pointed to by BR2) contains:
   - at offset 08 (hexadecimal) the program-name.
   - at offset 26 (hexadecimal) the version of the compiler used to compile the program.
   - at offset 42 (hexadecimal) the date of compilation.
   - at offset 47 (hexadecimal) the time of compilation.

   These items can be checked to ensure that the COBOL and LINKER listings used to analyze the dump correspond with its listing.
2. When PERFORM and ALTER statements are used in the program, it is advisable to determine which of them is active. This can be done by requesting that a "perform/alter bucket listing" be printed by the compiler (MAP parameter in COBOL JCL statement). This listing is explained in Chapter 2.

```
000760 07FE80 0FC00010 0FC00008 0FB50008 0FC10008  0FB60008 0FC40008 0FAE0010 5CC86DE2  @@@@@@@@@@@@@A@@@@@@@D@@@@@@*H_S
000780 07FEA0 D4F1F07A E2E8E24B E2E8E2E3 C5D40008                                     M10:SYS.SYSTEM@@

/J=08/P=00/  /STN=08/ STE=10/  SEGMENT  DESCRPT:  980F1E50 FC000006

5EGM.HEADR:   00F1E170 00F1E570 00010800 00104EA0  01031B41 040FD302 00000007 00000000

000000 F1E500 F0000010 0000C303 48120000 08100010  FF00003C 08100064 08100018 0810001C  0@@@@@C@@@@@@@@@@@@@@@@@@@@@@@@@
000020 F1E520 08110008 08100024 000C0003 0007000A  00021000 00C6C9D5 08110000 F0405C4D  @@@@@@@@@@@@@@@@@@@@@@@FIN@@@@0 *
000040 F1E540 FF007FFF 00F10C1C 1DD14040 081201D6  0C170050 F1F9F3F6 F5F4F7F0 001889FF  @@"@@1@@@J  @@@0@@@&19365470@@i@
000060 F1E560 08100000 00000001 D0010000 08120000                                     @@@@@@@@@@@@@@@@

/J=08/P=00/  /STN=08/ STE=11/  SEGMENT  DESCRPT:  9C0F1660 FE00000E

SEGM.HEADR:   00F165A0 00F166F0 00010800 00104EA8  01030B41 040FD303 0000000F 00000000

000000 F16600 07FE0010 0000C303 C6C9D5C4 60C4C1E8  40404040 40404040 40404040 40404040  @@@@@@C@FIND-DAY
000020 F16620 40404040 4040C3D6 C2D6D340 E560F0F0  4BF1F0F8 60F14040 40404040 40404000        COBOL V-00.108-1         @
000040 F16640 03F8F8FB F0F8F4F1 F87AF0F6 7AF0F0F6  F4404040 40F9F3F6 F5F4F7F0 001889FF  @88808418:06:0064    9365470@@i@
000060 F16660 08100000 F0F0F0F0 F3F1F0F5 F9F0F9F0  F1F2F0F1 F5F1F1F8 F1F2F1F2 F2F4F3F2  @@@@000031059090120151181212243 2
000080 F16680 F7F3F3F0 F4F3F3F4 D4D6D5C4 C1E84040  4040E3E4 C5E2C4C1 E8404040 E6C5C4D5  73304334MONDAY    TUESDAY   WEDN
0000A0 F166A0 C5E2C4C1 E840E3C8 E4D9E2C4 C1E84040  C6D9C9C4 C1E84040 4040E2C1 E3E4D9C4  ESDAY THURSDAY  FRIDAY    SATURD
0000C0 F166C0 C1E84040 E2E4D5C4 C1E84040 40404040  00000000 FFFFFFFF 40404040 40404040  AY  SUNDAY      @@@@@@@@
0000E0 F166E0 40404040 08120042 40404000 00000000                                        @@@@   @@@@@

/J=08/P=00/  /STN=08/ STE=12/  SEGMENT  DESCRPT:  980F1671 FD000020

SEGM HEADR:   00F165E0 00F16920 00010800 00104EB0  01030B41 040FD304 00000021 00000000

000000 F16710 C2000008 0080BC40 01F2752F 00289701  000ECC01 000E00F1 CD01000E 000C9601  B@@@@@@ @2@@@@p@@@@@@@@1@@@@@@o@
000020 F16730 000E8270 07FE4372 0000B880 00086272  00005217 003C7212 00E475B2 00E4BD30  @@b@@@@@@@@@@@@@@@@@@@@@@U@@@U@@
000040 F16750 0020754F 00404E6C 00004E70 00048200  0FFF3C60 3C703367 B8800006 1800D208  @@@!@ +%@@+@@@b@@@@-@@@@@@@@@@k@
000060 F16770 000C0802 00D0D322 00D62107 002DB860  0014D202 00D00602 00D2FA27 00442202  @@@@@@L@@0@@@@@-@@K@@@@@@K@@@@@@
000080 F16790 00D0FA02 00D62401 0019C422 00D42080  5E870018 F0028064 34010019 FA0200D0  @@@@@0@@@@D@@M@@;g@@0@@@@@@@@@@@
0000A0 F167B0 4S010021 F1070035 15010021 FA010021  5801002E F4070046 3801002E FA01002E  @@@@1@@@@@@@@@@@@@@@4@@@@@@@@@@@
0000C0 F167D0 88010026 FA010019 49010036 F0010026  89010036 FA210036 9A0200D8 F3070046  h@@@@@@@@@@@0@@@i@@@@@@@@@Q3@@@
0000E0 F167F0 120200D4 B8A0000C F1270035 140200D0  FA0200D0 44010019 F7070049 14010019  @@@M@@@@1@@@@@@@@@@@@@@@7@@@@@@@
000100 F16810 FA210019 4A020055 F0220055 AA0200D0  1502F822 00D04A02 0055F122 0055AA02  @@@@[@@@0@@@@@@Q@@8@@@[@@@1@@@@@
000120 F16830 00D81503 F82200D0 4A020055 F0220055  AA0200D8 FA0200D8 AA010023 F707004A  @Q@@8@@@[@@@0@@@@@@Q@@@@Q@@@@7@@5
000140 F16850 1A010023 FA010023 AA010019 FA210019  AA01002D FA01002D AA010041 F407004A  @@@@@@@@@@@@@@@@@@@@@@@@@@@@4@@[
000160 F16870 1A010041 FA010041 AA010037 FA0200DB  AB010050 F1010037 AB010050 FA010050  @@@@@@@@@@@@@@@@Q@@@&1@@@@@@&@@@@&
000180 F16890 8501004B FA210019 AA020055 FA21004B  5502005F F3070049 1502005F B8C00010  @@@.@@@@@@@@@@@.@@@^3@@@@@@^@@@@
0001A0 F168B0 F1270049 1502005F B8F0000C F0270046  1502005F FA22005F 51080010 C422005F  1@@@@@@^@0@@0@@@@@@^@@@^@@@@D@@^
000AC0 F168D0 50905E97 001CD202 90880A08 00147532  00E4B53A 00E4754F 00404E6C 00004E70  &@;p@@K@@h@@@@@@@U@@@U@!@ +%@@+@
0001E0 F168F0 00048200 0FFF3C60 3C703367 B8800006  18005247 004B1B81 8210005B 18110510  @@b@@@@-@@@@@@@@@@@@@.@ab@@$@@@@
000200 F16910 B0F40000 00000000 00000000 00000000                                     @4@@@@@@@@@@@@@@@

/J=08/P=00/  /STN=08/ STE=14/  SEGMENT  DESCRPT:  9C0EEE60 5600007F

SEGM.HEADR:   00EEE4E0 00EEEE00 00010800 00104EC0  00030F51 00000000 000000FF 08140800
```

Figure 4-4. Segment Dump

4.3 Job Execution Messages

Messages may be output to the JOR from the following sources:

- the system;
- COBOL run-time package procedures.

The types of messages output from these sources are described in the following paragraphs.

4.3.1 Messages Output by the System

The general format of messages output by the system in the JOR is as follows:

```
ccnn.text
```

where: cc is a two letter classification code;
nn is the number of the message within its class.

The messages are classified according to the nature of the system function which generated the message. Some of the more common classification codes and corresponding system functions are:

- CK Checkpoint/Restart
- DV Device Management
- EX Exception Handling
- FP File Open/Close

Depending on the error class, the text following the code may be a brief explanation of the cause of the error or else a further numerical classification followed by a return code specification. A complete list of classification codes, messages and return codes is given in the *Messages and Return Codes Directory*.

The message may be prefixed by WARNING, FATAL or SYSTEM. The significance of these prefixes is as follows:

- WARNING: Processing conditions are inconsistent with the expected conditions but the inconsistency is not severe enough to prevent the program execution from continuing.
- FATAL: This is caused by a serious user, operator or system error. Usually, program execution cannot continue and the step is abnormally terminated.
- SYSTEM: This is probably caused by some malfunction of the system. Normally the message comprises simply a message class and number with no text. Unlike WARNING and FATAL messages, the meaning of SYSTEM messages will not be self-evident and should be referred to Field Engineering.

4.3.2 Messages Output by COBOL

The following messages may be output by the COBOL run-time package:

```
CBL11.  DISPLAY   program_name console_displayed_string

CBL12.  IFN:ifn   RELATIVE KEY CANNOT BE USED

CBL13.  ACCEPT    program-name console_accepted_string

CBL14.  IFN:ifn   ORGANIZATION OVERRIDDEN

                                          [          { IN INPUT }]
CBL15.  IFN:ifn   RECORD LENGTH CONFLICT [ ACCEPTED {           }]
                                          [          { FOR NSTD }]
                  (length ON FILE)

        {CALL  }
CBL16.  {      }  program-name Rc = xxxxxxxx siuic, retcode
        {CANCEL}
                  AT ADDRESS stn.ste.sra

CBL17.  STOP      literal

CBL18.  IFN:ifn   RC = xxxxxxxx siuic, retcode
                  AT ADDRESS stn.ste.sra
                  program-name [ILN=internal-name
                  [XLN=external-line]]

CBL19.  IFN:ifn   CONTROL RECORD 101 TRUNCATED

CBL20.  USETST    RC = xxxxxxxx siuic, retcode
                  AT ADDRESS stn.ste.sra

CBL22.  IFN:ifn   SYNCOMP RC GAVE 9G FILE STATUS

CBL23. IFN:ifn    DATAFORM OVERRIDING NOT TAKEN INTO ACCOUNT
```

CBL11, CBL13 and CBL17 are simply DISPLAY, ACCEPT and STOP literal messages which are echoed in the JOR when they are directed to or from a CONSOLE or ALTERNATE-CONSOLE (see Chapter 11). The remaining messages indicate that an inconsistency has been detected. Messages CBL12, CBL19, CBL20, CBL22 and CBL23 are warnings that do not cause an abnormal termination. The step will be abnormally terminated if the message is CBL16. If the message is CBL18, the step will be abnormally terminated only if the return code indicates a serious error (see the *Messages and Return Codes Directory* for a full list of return codes).

Abnormal termination will not occur for message CBL15 if the file is an input file, however if a record greater than the record area exists in the file, reading this record will set file status "04", the record will be truncated on the right to comply with the maximum size, and the CPU time for the READ will be substantially increased. If the clause COBOL 1974 FOR FILE is used in the DEFAULT SECTION, the file status set will be "9U", and the program will abort unless a USE procedure exists. Abnormal termination will not occur for message CBL14 if overriding is permitted.

4.3.3 Exception Messages

Most abnormal step terminations result from the detection of an "exception" by the system. An exception is an error condition detected during the execution of an instruction (e.g., illegal operation code, illegal decimal data). The system outputs an exception message in the JOR whenever an exception is detected. Exception messages have a classification code EX. There are four possible exception messages: EX01, EX03 which are normally fatal; and EX02, EX04 which are non fatal.

Format of Exception Messages

The formats of the exception messages are as follows:

```
EX01.  EXCEPTION cc tt : message text (message parameter)
       IN TASK name.nnn    { AT ADDRESS  } stn.sre.sra
                           { RETURNED BY }

EX02.  EXCEPTION cc-tt : message-text  (message-parameter)
       IN TASK name.nnn    AT ADDRESS   stn.ste.sra

EX03.  {UNEXPECTED RETURN CODE (mnemonic) GOT }
       { ABNORMAL  RETURN CODE (mnemonic) SET}
       IN TASK name.nnn    AT ADDRESS  stn.ste.sra

EX04.  MAXIMUM EXPECTED WARNING COUNT EXHAUSTED
```

where:

Cc	is the class of exception (decimal);
Tt	is the type of exception (decimal);
Message-text	is a plain English explanation of the error;
Message-parameter	is an optional value to help diagnosis;
Name	is the task name from the LINKER listing (normally MAIN);
Nnn	is the task occurrence number (decimal);
Stn	is the segment table number;
Ste	is the segment table entry;
sra	is the segment relative address;
mnemonic	is a character string equivalent to the return code. A list of return codes and mnemonics is given in the *Messages and Return Codes Directory*.

NOTES:

1. stn.ste.sra are discussed in "Dump Analysis", above.
2. EX01 and EX03 are normally fatal. EX02 and EX04 are non-fatal.
3. An EX03 message specifying "UNEXPECTED RETURN CODE" indicates that the COBOL run-time package has received an unexpected return code from Data Management. An EX03 message specifying "ABNORMAL RETURN CODE" indicates that the COBOL run-time package has requested abnormal termination of the step.

4. EX03 need not be fatal if the COBOL program contains a USE AFTER ERROR PROCEDURE section in the Declaratives for the relevant file (See Chapter 9).
5. EX04 is printed when more than 99 EX02 messages have been printed for the current step. After EX04 no further EX02 messages will be printed.

Four of the more common exception messages are:

```
EX01. EXCEPTION 09-00: ILLEGAL FLOATING POINT DATA ...

EX01. EXCEPTION 09-01: ILLEGAL DECIMAL DATA...

EX01. EXCEPTION 17-02: OUT OF ARRAY RANGE...

EX01. EXCEPTION 06-00: OUT OF SEGMENT BOUNDS...

EX03. UNEXPECTED RETURN CODE...
```

These exceptions are discussed in the following paragraphs. A full list of exception messages is given in the *Messages and Return Codes Directory*.

```
EXCEPTION 09-00 ILLEGAL FLOATING POINT DATA
```

This exception occurs when a floating point number referenced in a machine code divide or compare instruction is not normalized. That is, the most significant hexadecimal digits of the mantissa are zero.

```
EXCEPTION 09-01 ILLEGAL DECIMAL DATA
```

This exception occurs when a non-decimal value is moved to a data item which is described as numeric or is involved in computation or is used as a subscript. The following example shows how this can happen:

```
.
WORKING-STORAGE SECTION.
01  GR1 VALUE LOW-VALUES.
    02 ZONE PIC 9(4).
.
.
PROCEDURE DIVISION.
P1.
    ADD 1 TO ZONE.
    STOP RUN.
.
```

ZONE is a numeric data item. In the native collating sequence (EBCDIC), the figurative constant LOW-VALUE corresponds to hexadecimal "00", with all bits set to 0. This configuration is not decimal, hence the exception.

This exception disappears if the DEBUG parameter is included in the STEP JCL statement and the RECOVER ILLDEC command is given to the PCF. Instead, the error is reported in the PCF report. See the *Program Checkout Facility User's Guide* for details. See Chapter 5 of the current manual for the format of decimal data.

```
EXCEPTION 17-02 OUT OF ARRAY RANGE
```

This is caused when attempting to access a data item beyond the upper or lower limits of the table which contains the data item. This exception disappears if the DEBUG parameter is included in the STEP JCL statement and the RECOVER SUBSCRIPT command is given to the PCF. If this is done, the error is reported in the PCF report.

```
EXCEPTION 06-00 OUT OF SEGMENT BOUNDS
```

This is caused when attempting to access a data item outside the memory areas allocated to the segments of the executing program. This exception can occur if the SUBOPT parameter is used in the CBL JCL statement. Under certain circumstances this parameter can result in no array bound checks being performed. Thus the program may attempt to access data outside the array, and possibly outside the program segments. It is advisable only to use the SUBOPT parameter after the program has been debugged. At this stage the program should be unlikely to access data beyond array bounds.

Unexpected Return Code

This is caused either by a user error or by a system difficulty. Some return codes are specific to COBOL programs. They are:

USER 0, RPWUNBUN	Attempt to use the COBOL Report Writer when it is not included in the set of features delivered with the system.
USER 0, ALREADY	Attempt to INITIATE an already initiated report (Report Writer).
USER 0, NOINIT	Attempt to execute a Report Writer statement when the involved report is not in the initiated state (no INITIATE has been executed for the report that has not been followed by a TERMINATE).
USER 0, LNERR	the data item referenced in the DEPENDING ON option of an OCCURS or a PICTURE clause lies outside the limits specified in that clause, or a reference modification specifies characters outside the data-name.

USER 0, JUMPERR	attempt to execute a GO TO statement without procedure-name, before it is altered.
USER 0, SEQERR	the flow of control attempts to go beyond the end of the program.
USER 0, SNDARERR	attempt to write a record whose length is less than the minimum length or greater than the maximum length for the related file.
COBOL 1, RECERR	the maximum record size of the file is not the same as that specified in the program. (See Record Size, Chapter 9).
COBOL 1, KEYERR	the number of record keys of the indexed file, their position relative to the beginning of the record, their length and/or the permissible duplicate keys are not the same as those specified in the program.
COBOL 1, WRONGORG	the file assigned has an organization that cannot override that specified in the program.
COBOL 6, IDERR	identifier in a "CALL identifier" statement does not contain a program-name.
COBOL 7, ARGERR	error in exponentiation arguments.
any other	abnormal code returned by the system (e.g., by Data Management) if no USE procedure is used, by the Message Control System, by the sort routines, ..., . Refer to the *Messages and Return Codes Directory* for a description of these return codes.

5. Representation of Data

This chapter describes the way in which data descriptions are interpreted by the COBOL compiler and the way in which data is held in memory in a COBOL program.

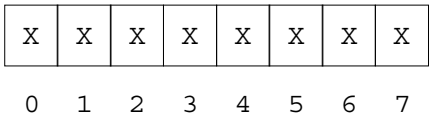
The value of a numeric item may be represented in either binary or decimal form. In addition, there are several ways of expressing decimal. The selection of radix is dependent upon factors included in clauses such as USAGE.

The types of data supported by COBOL are listed below, according to factors included in the USAGE and PICTURE clauses. The usage of an item specifies the format of the data item in computer storage. Note that only the usages DISPLAY, COMPUTATIONAL and INDEX are part of the ANS standard. The following paragraphs describe how each data type is represented in internal memory.

5.1 Format of Data in Memory

The basic element of information in memory which is handled by instructions is the byte (eight bits). A group of two consecutive bytes forms a halfword. Four consecutive bytes form a word. An address defines the location of a byte in main storage. The location of a group of bytes (e.g., halfword, word) is defined by the address of the left-most byte. Consecutive bytes from left to right are defined by consecutive increasing addresses. A group of bytes is called halfword-, word-, or doubleword-aligned, if its address is a multiple of two, four, or eight, respectively.

The bits forming a byte are defined from left to right and are numbered zero through seven. Byte format is represented as follows:



Data Representation

Usage	Machine Description	Picture
DISPLAY	EBCDIC Byte or unpacked decimal	R
*COMPUTATIONAL or COMP	Packed decimal (possibly without sign position depending on picture)	R
PACKED DECIMAL	Packed decimal (always with sign position)	R
BINARY	16-bit or 32-bit fixed binary	R
COMPUTATIONAL-1 or COMP-1	**16-bit fixed point binary	NR
COMPUTATIONAL-2 or COMP-2	32-bit fixed point binary	NR
*COMPUTATIONAL-3 or COMP-3	Packed decimal (possibly without sign position depending on picture)	R
COMPUTATIONAL-5 or COMP-5	Packed decimal (always with sign position; sign has ASCII representation)	R
COMPUTATIONAL-8 or COMP-8	Packed decimal (always with sign position)	R
COMPUTATIONAL-9 or COMP-9	Floating-point binary single precision	NA
COMPUTATIONAL-10 or COMP-10	Floating-point binary double precision	NA
COMPUTATIONAL-15 or COMP-15	Floating-point binary quadruple precision	NA
BIT	1 bit per Boolean position	R
POINTER	32-bit direct ITS	NA
INDEX	6 bytes	NA

Notes for this table:

R=	PICTURE clause required in data description entry.
NR=	PICTURE clause not required in data description entry.
NA=	PICTURE clause not allowed in data description entry.

- * These items have the same meaning, unless specified otherwise in the Default Section of the Control Division;
- ** If the PICTURE clause specified more than 4 digits, a 32 bit fixed-point binary data item is used.

5.2 Display Data Items

Character-strings are defined, explicitly or implicitly, by a USAGE IS DISPLAY clause. Character-strings, represented in EBCDIC code, are stored in memory in contiguous bytes with one character per byte. These character-strings may be non-numeric data as well as unpacked decimal numbers or Boolean character string.

An unpacked fixed point decimal number (PIC 9 or PIC S9) has the following format. Note that for PIC S9 the sign position is merely a zone.

zone	digit	zone	digit	zone	digit	zone	digit
byte		byte		byte		byte	

Each digit occupies the rightmost four bits of each byte:

- values from 0 (0000) to 9 (1001) are legal;
- values from A (1010) to F (1111) are illegal and produce an exception.

Zone values are not checked by decimal instructions.

The sign occupies the four left most bits of the last byte:

- values from A to F (1010 to 1111) are legal;
- values from 0 (0000) to 9 (1001) are illegal and produce an exception.

Signs are interpreted by instructions in the following manner:

Sign Encoding	Sign
1010	+
1011	-
1100	+
1101	-
1110	+
1111	+

Instructions which use the encoded sign put the sign into the result field in the following manner:

Sign	Sign Encoding
+	1100 (C)
-	1101 (D)
Absolute Value Result	1111 (F)

Decimal instructions do not examine the zones of source operands. The code 1111 is put in all zones in the result fields. The length of an unpacked decimal number may be from one to thirty one digits.

The contents of a Boolean character-string are restricted to the EBCDIC representation of characters "0" (zero) and "1".

5.3 Packed Decimal Numbers

A packed decimal number (USAGE IS COMP, PACKED-DECIMAL, COMP-3, COMP-5 or COMP-8) is represented as a series of contiguous bytes, each containing two 4-bit encoding portions, except for the rightmost byte. The leftmost four bits of this byte represent a digit, while the rightmost four bits define a sign. However, if the USAGE is COMP, COMP-3 or COMP-5 and the picture character string does not have a sign, the rightmost four bits represent the rightmost digit. Unsigned COMP, COMP-3 and COMP-5 items should be avoided, if possible, for efficiency reasons (see Chapter 8, Efficiency Techniques).

A signed packed decimal number (COMP, PACKED-DECIMAL, COMP-3, COMP-5 or COMP-8) has the following format:

digit	digit	digit	digit	digit	digit	digit	sign
byte		byte		byte		byte	

Each digit occupies four bits:

- values from 0 (0000) to 9 (1001) are legal;
- values from A (1010) to F (1111) are illegal, and produce an exception.

The sign, if present, occupies the last digit position:

- values from A to F are legal (when usage is COMP-5, A,B,C,E,F, = + , D = - ; for other usages, A,C,E,F = + , B,D = -);
- values from 0 (0000) to 9 (1001) are illegal and produce an exception.

Signs are interpreted by instructions in the following manner:

Sign Encoding	Sign	sign COMP-5
1010	+	+
1011	-	+
1100	+	+
1101	-	-
1110	+	+
1111	+	+

Signs put in result fields by instructions are encoded in the following manner:

Sign	Sign Encoding	Sign COMP-5
+	1100 (C)	1011 (B)
-	1101 (D)	1101 (D)
Absolute Value Result	1111 (F)	1111 (F)

A packed decimal number may occupy from one to sixteen bytes. The length, L, of a packed decimal number, is specified in digits. The number of bytes occupied is determined in the following manner.

Type	Number of bytes
L even	L / 2 + 1
L odd	(L + 1) / 2

When L is even, the leftmost digit position must be zero.

5.4 Fixed-Point Binary Numbers

Fixed-point binary data can be specified as either 16-bit binary (USAGE IS COMP-1 and no PICTURE clause, or USAGE IS COMP-1 or BINARY and a PICTURE clause showing less than 5 digits) or 32-bit binary (USAGE IS COMP-2 or USAGE IS COMP-1 or BINARY with a PICTURE clause showing more than 4 digits). The short binary data item consists of two contiguous bytes; the long binary data item, of four contiguous bytes. In both types of data, a decimal point is assumed to be to the right of the least significant bit. Negative values are stored in two's complement form.

5.5 Floating-Point Binary Numbers

Floating-point binary data can be specified either as 32-bit binary (USAGE IS COMP-9) or 64-bit (USAGE IS COMP-10) or 128-bit binary (USAGE IS COMP-15). The short floating-point binary data item gives single precision (a precision of approximately 7 decimal digits). The long floating-point binary data item gives double precision (a precision of approximately 16 decimal digits). The extended floating-point binary data item gives quadruple precision (a precision of approximately 27 decimal digits).

The value of a floating-point binary number, V, is defined by the following equation:

$$V = (-1)^S \times 16^E \times .M$$

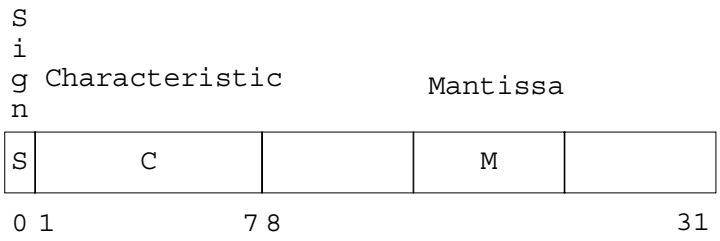
where $E = C - 64$.

S is the sign, E is the exponent, C is the characteristic, and M is the mantissa of the floating-point binary number.

The value zero is represented by a floating-point binary number with mantissa equal to zero. A value of true zero is represented by a floating-point binary number with all bits equal to zero.

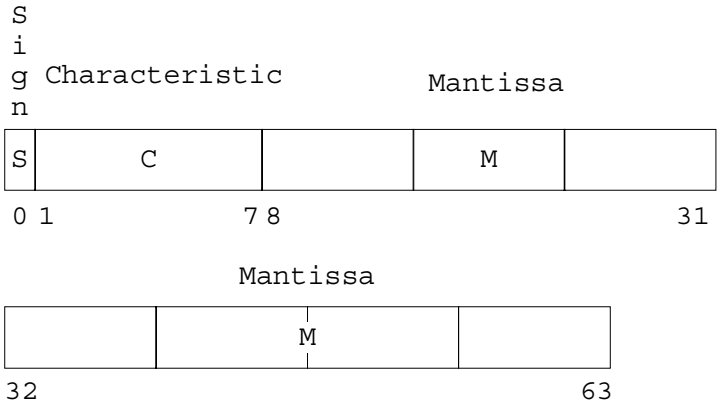
Short Floating Point Binary Number

A short floating-point binary number occupies four bytes. The format is as follows:



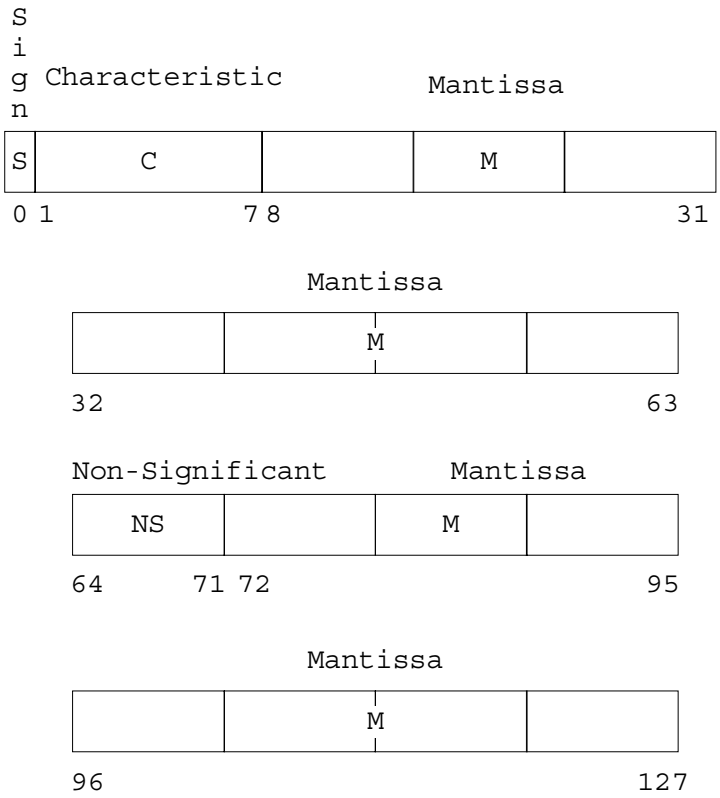
Long Floating Point Binary Number

A long floating-point binary number occupies eight bytes. The format is as follows:



Extended Floating Point Binary Number

An extended floating-point binary number occupies sixteen bytes. The format is as follows:



The sign S of a floating-point binary number is contained in bit 0.

- S = 0 positive sign
- S = 1 negative sign

The characteristic C of a floating-point binary number is contained in bits 1 through 7. Its range is 0 through 127.

The exponent, E, is the power to which 16 is raised in calculating the value of the floating-point binary number. The exponent E is equal to C - 64.

The mantissa M is the hexadecimal number contained in bits 8 through 31 for a short floating-point binary number, or in bits 8 through 63 for a long floating-point binary number, or in bits 8 through 63 then 72 through 127 for an extended floating-point binary number. The radix point is at the left of the high-order digit position.

5.6 Bit Strings

Bit strings are defined, explicitly or implicitly by a USAGE IS BIT clause. Bit strings are represented in memory in contiguous bits with one bit per Boolean position. The bit values "0" (zero) and "1" (one) correspond respectively to the Boolean values "0" (zero) and "1" (one). Bit strings are not necessarily left and/or right byte aligned. Byte or bit alignment depends on size, SYNCHRONIZED attributes and the presence of immediately preceding other bit strings.

5.7 INDEX Data Item

An INDEX data item (USAGE IS INDEX) consists of 48 bits (six bytes) of binary data and contains the relative displacement from the beginning of the table and the occurrence number of the table element.

5.8 DB-KEY Data Item

A data-item USAGE DB-KEY consists of 32 bits of binary data representing the logical address which uniquely identifies a data-base record within the whole data-base.

6. Calling and Called Programs

An application can be divided into several separately compiled programs. These programs can later be linked together by the LINKER utility to form a single executable load module. Control is transferred between programs by the CALL and EXIT PROGRAM statements.

This chapter does not describe the use of contained and nested contained programs. The description of this new COBOL 85 feature is given later in this manual (Structured Programming, Chapter 13).

The use of calling (and called) separately compiled programs has the following advantages:

- A program which has been written and compiled and stored in a compile unit library can be called by other programs and can thus be included in several different load modules without having to be compiled each time.
- Programs written in languages other than COBOL (e.g., FORTRAN) can call or be called by COBOL programs.
- Programs can be written by several programmers and can later be combined into a single load module.

However, the use of calling and called separately compiled programs may increase execution time slightly.

The following ANS standard COBOL language facilities are used in calling and called programs:

- the CALL and EXIT PROGRAM statements and optionally the CANCEL statement;
- the Linkage Section;
- the USING phrase of the PROCEDURE DIVISION header.

The EXTERNAL phrase is available as an alternative or complement to the Linkage Section and USING phrase. The use of all the above facilities is discussed in the following paragraphs.

NOTE:

For an explanation of COBOL segment number, internal segment number and LINKER segment number see Chapter 3, Linking.

6.1 Transfer of Control

The transfer of control between COBOL programs is achieved by using the CALL and EXIT PROGRAM statements.

The CALL statement passes control to the program with the specified program-name (from the PROGRAM-ID paragraph). For example:

```
CALL "PROG2".
```

In this example, control will transfer to the program whose name is PROG2. (All separately compiled programs which are to be linked into a single load module must have names which are unique within that load module). Control is handed to the first non-declarative statement in the Procedure Division of the called program.

It is important to note the distinction between calling programs and called programs. A calling program contains a CALL statement which may refer to a separately compiled program. A separately compiled called program is the subject of a CALL statement in another separately compiled program. Called programs may also be calling programs. That is, they may call other programs. However, a load module contains only one program which is not a called program. This is the "main program" which is specified in the ENTRY parameter of LINKER (or by default). Step execution commences from the first non declarative statement in the Procedure Division of the main program.

The EXIT PROGRAM statement returns control from a called program to the calling program at the point immediately following the CALL statement. An EXIT PROGRAM statement which is not in a called program (i.e., it is in the main program) is ignored when the program is executed. The main program should be terminated by a STOP RUN statement. This statement may also appear in any called program. A STOP RUN statement in any program of a load module will cause execution of the load module to be terminated immediately.

Any separately compiled program may be used in more than one load module. Such a program may sometimes act as a main program and sometimes as a subordinate program. In this case the program should be terminated by an EXIT PROGRAM statement followed immediately (in the next paragraph) by a STOP RUN statement. When the program is a main program the EXIT PROGRAM statement will be ignored and load module execution will be terminated by the STOP RUN statement. When the program is not a main program the EXIT PROGRAM statement will hand control back to the calling program.

The transfer of control between COBOL programs and non COBOL programs is covered later in this chapter.

6.2 Linkage Section and USING Phrase

A calling program may provide data for a called program to process. Similarly, the called program may return processed data to the calling program. This exchange of data is achieved using the Linkage Section and the USING phrase of the PROCEDURE DIVISION header. An alternative or additional method of exchanging data, using the EXTERNAL phrase, is discussed under a later heading.

Data to be passed to a called program is specified in the USING phrase of the CALL as shown in the following example:

```
CALL "PROG3" USING QUANTITY, PRICE, VALU.
```

In this example let us assume that QUANTITY and PRICE are elementary items on an input file and that VALU is an elementary item in the Working-Storage Section. The called program must contain a Linkage Section which contains three data items each of which has the same picture and usage as one of the data items specified in the USING phrase of the CALL statement. In addition these three items must be mentioned in the USING phrase of the PROCEDURE DIVISION header in the same order in which they appear in the CALL statement. For example:

```
PROGRAM-ID. PROG3.
   .
   .
LINKAGE SECTION.
01     VALUE-L     PIC...
77     PRICE-L     PIC...
77     QUANT-L     PIC...
   .
   .
PROCEDURE DIVISION USING QUANT-L, PRICE-L, VALUE-L.
   .
   .
   IF QUANT-L > 100 MULTIPLY .90 BY PRICE-L.
   MULTIPLY QUANT-L BY PRICE-L GIVING VALUE-L.
   .
   .
EXIT-PARA.
   EXIT PROGRAM.
```

The data items described in the Linkage Section are not allocated any storage space in the data segment(s) of the called program; instead data names in the Linkage Section are associated with locations in the data segment or segments of the calling program.

The purpose of the Linkage Section is to enable the programmer to specify the pictures of the data items that are to be processed in the called program or are to receive results from the called program. Another purpose of the Linkage Section is to enable the programmer to give local names to the data items to be processed. These data names need not be the same as those used in the calling program, nor need the data items be described in the same order as they appear in the calling program. However, the order of data names in the PROCEDURE DIVISION header and in the CALL statement must be the same.

The code generated by the compiler when a reference is made to a data item in the Linkage Section is based on the data descriptions contained in the Linkage Section. However, the generated code actually refers to the storage areas allocated in the calling program. If the data description in the Linkage Section is not identical to that in the calling program, the results will be unpredictable.

In the above example, the content of PRICE and VALU are changed when control returns from PROG3 to the calling program. A called program may reference Linkage Section data items as receiving operands. However, the user may prevent a called program from changing the contents of data items referenced in the USING phrase of the corresponding CALL statement. This is done by specifying the BY CONTENT phrase as shown in the following example:

```
CALL "PROG3" USING QUANTITY, BY CONTENT PRICE,
                             BY REFERENCE VALU.
```

With the same PROG3 as in the previous example, VALU will be changed but PRICE will not. The first MULTIPLY statement will actually modify the contents of PRICE-L, but this parameter is associated with a dummy data item allocated in the called program to which PRICE is moved just before PROG3 is called. There is no such dummy data item for VALU because of the BY REFERENCE phrase.

Note that no dummy data name is created for QUANTITY since the BY REFERENCE phrase is inserted before the first parameter.

6.3 The EXTERNAL Phrase

The EXTERNAL phrase may be included in any FD entry in the File Section or in the 01 or 77 level of any data description in the Working-Storage Section or the Constant Section (the Constant Section is not part of the ANS standard). The effect of this phrase is to make the file and/or the constituent data items available to every program in the load module which describes that file or record. The EXTERNAL phrase is used as an alternative or complement to the Linkage Section and USING phrase. The use of this phrase in a calling program is shown in the following example:

```
   .
WORKING-STORAGE SECTION.
   .
01 SHARED-DATA EXTERNAL.
   02 QUANTITY    PIC...
   02 PRICE        PIC...
   02 VALU        PIC...
   .
PROCEDURE DIVISION.
   .
   CALL "PROG3".
   .
```

If the called program has to refer to the data items in the record SHARED-DATA, it must contain an identical record description (including identical level 01 data-name, other data-names used need not be identical). For example:

```
PROGRAM-ID. PROG3.
   .
WORKING-STORAGE SECTION.
   .
01 SHARED-DATA EXTERNAL.
   02 QUANTITY    PIC...
   02 PRICE        PIC...
   02 VALU        PIC...
   .
PROCEDURE DIVISION.
   .
   .
   MULTIPLY QUANTITY BY PRICE GIVING VALU.
   .
   .
EXIT-PARA.
   EXIT PROGRAM.
```

Identical record descriptions which contain an EXTERNAL phrase and which occur in more than one program in a load module are all allocated the same storage space in memory. The code generated by the compiler when any program in a load module makes a reference to a particular external data item always refers to the same data segment address. If a calling program and a called program both use an external record which has the same record name but different descriptions for the elementary items, the results are unpredictable.

6.4 The CALL Identifier

The "CALL identifier" statement enables the program to call different programs with the same CALL statement. The name of the called program is stored in "identifier" and can be changed during step execution. If the CALL identifier statement is used to call separately compiled programs, the called compile units must be specified to LINKER in the INCLUDE command. See Chapter 3, Linking.

6.5 The CANCEL Statement

The CANCEL statement may be used in a calling or called program. The main function of this statement is to initialize the state of the specified program or programs. For example:

```
CANCEL "PROG4".
```

In this example the variables in the program PROG4 are set to the values which existed when the load module began execution (including PERFORM return and ALTER buckets - see " Perform/Alter Bucket Listing" Chapter 2).

Note that the ANS standard specifies that cancelled programs are removed from memory and are initialized when returned to memory. Removal from memory is unnecessary under Virtual Memory Management, so the program is simply initialized.

6.6 Interface with COBOL74 Programs

It is possible to call a COBOL 74 program from a COBOL 85 program and vice-versa (DPS 7 extension to the ANSI Standard).

6.7 Interface with FORTRAN77 Programs

Programs written in FORTRAN77 can be called by COBOL programs. These programs are called as if they were COBOL programs containing a Linkage Section. That is, they are called with the USING phrase of the CALL statement.

Called FORTRAN77 programs must be in the form of a normal FORTRAN77 subroutine. For example:

```
SUBROUTINE FORSUB (A,B,C...)
     .
     .
RETURN
```

The arguments A,B,C etc., must have the same data format and must be in the same sequence as the data items named in the USING phrase of the COBOL CALL statement. The COBOL data formats which are recognized by FORTRAN77 are shown below. No other data formats are permitted.

Data Formats in FORTRAN77 Called Programs

Data Format	COBOL Data Description	FORTRAN 77 Declaration
Alphabetic, alphanumeric or edited	PIC A PIC X PIC $9.9 etc.	CHARACTER*n
One word binary	BINARY PIC S99999 COMP-1 COMP-2	INTEGER
One half-word binary	BINARY PIC S9 thru S9999 COMP-1 PIC S9 thru S9999 COMP-1	INTEGER*2
Single word floating point.	COMP-9	REAL
Double word floating point.	COMP-10	DOUBLE PRECISION
Quadruple word floating point.	COMP-15	QUADRUPLE PRECISION

NOTE:

If an argument of alphabetic, alphanumeric or edited data format is to be passed between COBOL and FORTRAN77 programs, the FORTRAN77 program must either be compiled with the LEVEL=GCOS1E parameter in the FOR77 JCL statement, or use the REFCHAR intrinsic function.

The following example shows a COBOL program calling a FORTRAN77 program:

```
    .
    .
WORKING-STORAGE SECTION.
    .
    .
77 ANGLE            COMP-10.
    .
    .
PROCEDURE DIVISION.
    .
    .
    CALL "COSINESQ" USING ANGLE.
    .
    .
```

The called FORTRAN77 program may be as follows:

```
    .
    .
SUBROUTINE COSINESQ (ARG)
    .
    .
DOUBLE PRECISION ARG,WORK
    .
    .
WORK = COS (ARG)
ARG = WORK**2
    .
    .
RETURN
```

In the case of CHARACTER DATA, the FORTRAN77 program need a descriptor containing pointer and length. In the calling COBOL program, this descriptor is built by using the ADDRESS OF facility in the CALL statement.

Programs written in COBOL can also be called by FORTRAN77 programs. The PROCEDURE DIVISION header in the COBOL program must contain a USING phrase. The arguments specified in this phrase must be in the same format and must have the same sequence as in the FORTRAN77 CALL statement. The data formats that can be used for arguments have been shown above.

In the case of character data, FORTRAN77 passes a descriptor containing pointer and length; the called COBOL program must use a pointer data in its PROCEDURE DIVISION clause.

COBOL and FORTRAN77 programs may share external data. The data items must be defined with the EXTERNAL attribute in COBOL, and in a named COMMON in FORTRAN77. In addition, the external data must be assigned at Linkage time in the same segment as the FORTRAN77 common.

For example, the following COBOL program:

```
   .
   .
WORKING-STORAGE SECTION.
   .
   .
01 C EXTERNAL.
   02 C1 PIC X(100).
   02 C2 PIC X(100).
   .
   .
```

shares the same data structure C as the following FORTRAN77 program

```
   .
   .
COMMON/C/C1,C2
CHARACTER*100 C1,C2
   .
   .
```

provided the following commands

```
PLACE = (C, C), MSEGAT = (GLOBSEG, C, ...)
```

are specified in the LINKER JCL statement.

FORTRAN77 and COBOL programs which call each other can share common files. When a file is to be shared, the following rules must be observed:

- The file must be described with the EXTERNAL attribute in the SELECT clause of the COBOL program. The REPORT clause and the LINAGE clause must not be specified in the file description entry of the Data Division of the COBOL program.
- The internal-file-name specified in the ASSIGN TO phrase of the SELECT EXTERNAL clause must be HCOBFOR1, HCOBFOR2, HCOBFOR3, HCOBFOR4 or HCOBFOR5, or the association between COBOL and FORTRAN77 internal-file-names must be done at Linkage time by using the LINKER command:

  ```
  REPLACE = (HCOBFORi, cobol-ifn)
  ```

- The file must be connected in the FORTRAN77 program. That is, an OPEN (n, FILE='IFN=ifn') statement must be executed with ifn being HCOBFOR1, HCOBFOR2, HCOBFOR3, HCOBFOR4 or HCOBFOR5.
- When any COBOL program calls a FORTRAN77 program and the FORTRAN77 program uses a common file, this file must be closed at the time of calling. The file must also be closed when the FORTRAN77 program returns to the COBOL program.
- When any FORTRAN77 program calls a COBOL program and the COBOL program uses a common file, and if the file is open at the time of calling, it must not be closed by the COBOL program; in addition:
  - a COBOL sequential READ statement may only be executed if the last FORTRAN77 I/O statement executed for the file was a FORTRAN77 sequential READ statement,
  - a COBOL sequential WRITE statement may only be executed if the last FORTRAN77 I/O statement executed for the file was a FORTRAN77 sequential WRITE statement,
  - a COBOL direct READ or WRITE statement may only be executed if the last FORTRAN77 I/O statement executed for the file was a FORTRAN77 direct READ or WRITE statement.

If the file is closed at the time of calling, it must be closed when the COBOL program returns to the FORTRAN77 program.

Remember that, in a FORTRAN77 program, a file is considered to be closed when:

- it has not yet been used in a FORTRAN77 program,
- the step begins execution,
- a CLOSE statement has just been executed.

FORTRAN77 files referenced in READ (*, ...) and WRITE (*, ...) statements are the same as the files referenced in COBOL ACCEPT and DISPLAY statements (explicitly or implicitly referencing SYSIN and SYSOUT), respectively, i.e. H_RD-SYSIN and H_PR-SYSOUT. In batch mode, the FORTRAN77 system input file must be assigned to H_RD, and the FORTRAN77 system output file may be assigned to H_PR. The above rules concerning common files do not apply; however, a FORTRAN77 READ (*, ...) statement must have been executed before any COBOL ACCEPT statement (explicitly or implicitly referencing SYSIN) is executed, and a FORTRAN77 WRITE (*, ...) statement must have been executed before any COBOL DISPLAY statement (explicitly or implicitly referencing SYSOUT) is executed.

6.8 Interface with GPL Programs

Programs written in GPL can be called by COBOL programs. These programs are called as if they were COBOL programs containing a Linkage Section. That is, they are called with the USING phrase of the CALL statement.

Called GPL programs must be in the form of a normal GPL procedure. For example:

```
GPLPROC: PROC (A,B,C...);
    .
    .
END;
```

The arguments A,B,C etc. must have the same data format and must be in the same sequence as the data items named in the USING phrase of the COBOL CALL statement. The COBOL data formats which are recognized by GPL are shown below. No other data formats are permitted.

Data Formats in GPL Called Programs

Data Format	COBOL Data Description	GPL Declaration
Alphabetic, Alphanumeric or edited	PIC A PIC X PIC $9.9 etc	CHAR (n)
One word binary	BINARY PIC S99999.. COMP-1 PIC s99999.. COMP-2	FIXED BIN (31)
One half word binary	BINARY PIC S9 thru S9999 COMP-1 PIC S9thru S9999 COMP-1	FIXED BIN (15)
One word pointer	A COBOL program can pass a pointer value as an argument to a GPL parameter declared as PTR in a GPL program, either by passing a USAGE POINTER data item or by using the 'ADDRESS OF identifier' phrase in the USING phrase of the COBOL CALL statement.	PTR

The following example shows a COBOL program calling a GPL program:

```
   .
   .
WORKING-STORAGE SECTION.
   .
   .
77 A               PIC XXX.
77 A1              COMP-2.
77 A2              COMP-1 PIC 9(6).
77 A3              POINTER.
   .
   .
PROCEDURE DIVISION.
   .
   .
   CALL "GPLPROC" USING A1, A3.
   .
   .
   CALL "GPLPROC" USING A1, ADDRESS OF A.
   .
   .
   CALL "GPLENTRY" USING A2.
   .
   .
```

The called GPL program may be as follows:

```
GPLPROC:  PROC (ARG1, ARG2);
   .
   .
DCL       ARG1 FIXED BIN (31);
DCL       ARG2 PTR;
   .
   .
GPLENTRY: ENTRY (ARG1);
   .
   .
END GPLPROC;
```

Programs written in COBOL can also be called by GPL programs. The PROCEDURE DIVISION header in the COBOL program must contain a USING phrase. The arguments specified in this phrase must be in the same format and must have the same sequence as in the GPL CALL statement. The data formats that can be used for arguments have been shown above.

COBOL and GPL programs may share external data. The data items must be defined with the EXTERNAL attribute in programs sharing the data. For example, the following COBOL program

```
   .
WORKING-STORAGE SECTION.
   .
   .
01 C EXTERNAL.
   02 C1     PIC X(100).
   02 FILLER COMP-1.
   .
```

and the following GPL program

```
   .
DCL      01 C EXTERNAL,
            02 * CHAR (100),
            02 C-2 FIXED BIN (15);
   .
```

share the same area C.

GPL and COBOL programs that call each other can share common files. When a file is to be shared, the following rules must be observed:

- In the COBOL program, the file must be described with either the EXTERNAL attribute in the SELECT clause (not ANS standard) or the EXTERNAL clause in the FD entry. The REPORT clause and the LINAGE clause must not be specified in the file description entry of the Data Division of the COBOL program.
- The file must have consistent definitions in the GPL and in the COBOL programs. More particularly, if the FD entry in the COBOL program contains the EXTERNAL clause, data-names such as that of the record area or the relative key must be declared as external variables in the GPL program. The external name given by the COBOL compiler to the record area of an external file connector is the concatenation of "H_", the ifn and "_RECORD".
- The file must be given the same internal-file-name in the $H_FD primitive of the GPL program as in the ASSIGN TO phrase of the SELECT clause of the COBOL program.
- A shared file opened by a COBOL program must be closed by a COBOL program before being opened by a GPL program. A shared file opened by a GPL program must be closed by a GPL program before being opened by a COBOL program. Apart from these rules, any legal I/O operation may be executed in either a COBOL program or a GPL program, provided they are consistent with the opening mode, the access mode and the implicit or explicit description of the file.

6.9 Interface with C Language Programs

Programs written in C can be called by COBOL programs. These programs are called as if they were COBOL programs containing a Linkage Section. That is, they are called with the USING and GIVING phrases of the CALL statement.

Called C programs must be in the form of a normal C procedure. For example:

```
float lac (ss, tt, ff, bb, kk, dd)
struct s2 {
int   c1, c2, c3;
} ss;
short tt;
float ff;
double bb;
char kk;
int dd[2];
{dd[0] = ss.c3;
ff = tt;
bb = 13;
kk = 'K';
return (ff);
}
```

Beware of upper- and lower-case letters in names and note that minus sign (-) is not allowed in C names.

The arguments ss, tt, etc. must have the same data format and must be in the same sequence as the data items named in the USING phrase of the COBOL CALL statement. The COBOL data formats which are recognized by C are shown below. No other data formats are permitted.

Data Formats in C Called Procedure

COBOL Data Format	COBOL Data Description	C language Declaration	C language Data format
Alphabetic Alphanumeric	PIC A PIC X	Char	The right hand byte of a word
One word binary	BINARY PIC S9(n) COMP-2	Int	One word
Half-word binary	BINARY pic S9 thru S9999 COMP-1	short int	The right part of a word
Single word floating point	COMP-9	Float	
Double word floating point	COMP-10	Double	
Pointer	USAGE POINTER ADDRESS OF	Pointer	

Aggregates are in correspondence, with the following rules:

1. Structures elements are allocated in the same order.
2. Arrays are allocated in the same manner in C and COBOL, but note that the first element is t[0] in C, T(1) in COBOL.

Very Important:

In Standard C mode, be aware of the type promotion when passing parameters. Especially:

- float is promoted to double;
- int, short, long and char are promoted to long;
- arrays are changed to pointers.

The standard way for C to pass parameters is "by value". It means that the object passed to the callee is the value, not the address.

For example, in the program fragment below:

```
int i;
i = 3;
p (i);
```

what is passed to p is the value of i (here 3), not i. It means that any modification done on the parameter in p has no influence on the argument i. In other words, when returning from p into the caller, the value of i is still 3, whatever the code of p can be. Therefore, in COBOL, the parameters would be passed BY CONTENT.

If the parameter must be passed BY REFERENCE, the usual way is to pass a pointer to the object rather than the object itself:

```
 p (&i)
```

where p is declared

```
void p (*int);
```

To allow easy inter language communication, a specific syntactical feature has been introduced in C to declare a function that accepts parameters passed by reference.

```
extern q (&);
```

The usage of this feature is restricted to LEVEL=GCOS7.

This specifies that all the parameters are passed by reference.

The usage of the byref (&) feature is strongly recommended when the called procedure is written in COBOL.

If a value is to be returned to the calling COBOL program, a GIVING phrase must be used in the CALL statement with the following rules:

A return value of int type must be stored in a data item of USAGE COMP-1 or COMP-2 or in an alphabetic, alphanumeric or Boolean item of length = 1.

A return value of double type must be stored in a data-item of USAGE COMP-9, COMP-10 or COMP-15.

A returned value of pointer type must be stored in a data item of USAGE POINTER.

Programs written in COBOL can also be called by C language procedures. The PROCEDURE DIVISION header in the COBOL program must contain a USING phrase; the EXIT PROGRAM statement must contain a GIVING phrase if a returned value is requested. The arguments specified in this phrase must be in the same format and must have the same sequence as in the C language call verb. The data formats which can be used for arguments have been shown above.

The following example shows a C language procedure calling a COBOL program (LAC4) which calls another C language function (LAC5).

Main C Procedure

```
main ()
{
extern void lac1 (&);
struct s1 {
int  c1, c2, c3;
} ss;
int dd[9];
short tt;
float ff;
double bb;
char kk;
extern int LAC4 (&);
ss.c1 = 2;
ss.c2 = 3;
ss.c3 = 7;
dd[0] = 8;
dd[1] = 9;
tt = 11;
ff = 12;
bb = 13;
kk = 'K';
dd[1] = dd[1] + ss.c1 + LAC4 (ss, tt, ff, bb, kk, dd);
fprintf ("LAC: Res= ", dd[1]);
}
```

LAC5 C Function

```
int lac5 ( & ss, tt, ff, bb, kk, dd)
struct s3 {
int   c1, c2, c3;
} ss;
short tt;
float ff;
double bb;
char kk;
int dd[9];
{
dd[0] = ss.c3;
dd[7] = 77;
ff = tt;
bb = 13;
kk = 'K';
return (dd[7]);
}
```

LAC4 COBOL Program

```
IDENTIFICATION DIVISION.
PROGRAM-ID.    LAC4.
DATA DIVISION.
LINKAGE SECTION.
01  SS.
 02  C1    COMP-2.
 02  C2    COMP-2.
 02  C3    COMP-2.
01  TDD.
 02  DD   OCCURS 9  COMP-2.
01  TT     COMP-1.
01  FF     COMP-9.
01  BB     COMP-10.
01  KK     PIC X.
PROCEDURE DIVISION  USING SS, TT, FF, BB, KK, TDD.
PARAG.
    MOVE 21 TO TT   MOVE 22 TO FF   MOVE 23 TO BB
    MOVE "H" TO KK.
    ADD C1 C2  GIVING C3.
    CALL "lac5"  USING SS, TT, FF, BB, KK, DD
                 GIVING C1.
    MOVE FF TO DD(2)  MOVE DD(8) TO DD(5).
    EXIT PROGRAM  GIVING DD(5).
```

6.10 Constraints

The following paragraphs describe some constraints which must be observed when writing calling and called programs.

6.10.1 Using Files

A file which is used in more than one program of a load module must be described in each program in which it is used. The descriptions must specify identical file parameters, though the actual data names used need not be the same. Either the File Description entry in each program must contain the EXTERNAL clause or the SELECT clause in each program must contain an EXTERNAL phrase (not ANS standard). The internal-file-name used in the SELECT clause of each program must be the same.

The COBOL file-names must be identical in each program if the EXTERNAL clause is present in the File Description entry. When the EXTERNAL phrase of the SELECT clause is used, file-names need not be identical.

Conversely, internal-file-names must be different for files which are not EXTERNAL (i.e., not to be used in more than one program) though the COBOL file-names may be the same (in different programs).

Note that record areas of files whose File Description entries do not contain the EXTERNAL clause are local to the program in which they are described (even when the SELECT clause contains the EXTERNAL phrase). That is, if program A reads a record from such a file, this record is not automatically available to program B when it is called, even though program B contains an identical description for this file. On the contrary, all programs that describe the same external file-connector (same file-name and EXTERNAL clause in the FD entry) share the record area of this file. So, if one of these programs reads a record from such a file, the record is available for others.

6.10.2 Report Writer

A report description (RD) is local to a compile-unit. Therefore, Report Writer statements used in a particular program can refer only to a report description in the same compile unit.

However, more than one program in a load module can produce reports using the Report Writer. If these reports are produced concurrently on the same (EXTERNAL) file and the CODE clause is used, the report code for each report must be unique within the load module.

For more information on the use of the Report Writer see Chapter 12.

6.11 Guidelines

The benefit of calling and called separately compiled programs have been listed at the beginning of this chapter. If none of these benefits apply to a particular called program, then this program should be written as a contained program.

Programs should normally be "structured". That is, they should be divided into logical units or modules of source code. However, this can be done without the use of calling and called separately compiled programs. The transfer of control between source modules can be made using contained programs or the PERFORM statement rather than calling separately compiled programs. The CALL and EXIT PROGRAM statements when contained programs are concerned (and the PERFORM statement) are more efficient than the same statements concerning separately compiled programs. See Chapter 7 for more details of the use of structured programming with the PERFORM statement.

7. Segmentation

Segmentation is the process of physically dividing a program into segments which can be located in main memory or on disk independently of each other during load module execution. The process of moving program segments between main memory and disk (swapping) is not the responsibility of the user program. It is handled by the system component Virtual Memory Management (VMM). For a description of VMM, see the *GCOS 7-V6 System Administrator's Guide*.

However, the programmer can influence the way in which the program is divided into segments. Good program segmentation will achieve the following:

- The number of times segments must be swapped between main memory and disk will be reduced. This has three benefits: first, the reduced rate of I/O minimizes queuing for the disk drives; second, the elapsed time of the program is reduced because waiting for segments to be loaded from disk is avoided as far as possible; third, the CPU time consumed by VMM is minimized.
- Execution of a load module can begin when only part of the load module is in main memory. This has two benefits: first, large load modules do not have to wait for a large amount of memory to become available at one time; second, large load modules that cannot fit into the available memory in one piece can be executed in segments.

The remainder of this chapter provides guide-lines for efficient program segmentation.

NOTE:

For an explanation of COBOL segment number, internal segment number and LINKER segment number, see Chapter 3, Linking.

7.1 Methods of Segmentation

Programs may be segmented in one or both of the following ways:

- COBOL segment numbers can be used on the section headers of the Procedure Division. This technique applies only to the Procedure Division. The Data Division can be segmented by using the DSEGMAX parameter of the CBL JCL statement. This parameter controls the automatic segmentation done by the compiler.
- Segmentation will be done automatically by the compiler whenever the size of a data or procedure segment exceeds the preferred segment size. The preferred segment size has a default value of 4K bytes (K = 1024). This value can be modified by parameters in the CBL JCL statement or by clauses in the OBJECT-COMPUTER paragraph (described below).

7.2 Control of Segmentation by the Programmer

Segmentation should be viewed as a tool by which a user programmer can take advantage of VMM. The user program should be segmented according to the logical structure of the program. This will enable VMM to hold in memory a minimum number of segments, thus minimizing the memory requirement and reducing the amount of swapping.

If the programmer does not explicitly segment his program in this way, the compiler will automatically divide the program into segments (see Automatic Segmentation, below). However, the compiler does not have enough information about the program logic to optimize automatic segmentation. Therefore, the programmer should segment his program explicitly (unless the program is very small, in which case no automatic segmentation will be done by the compiler). The ways in which the programmer can influence segmentation to ensure efficient use of Virtual Memory are described below.

7.2.1 Procedure Division Segmentation

By specifying segment numbers in the section headers of the Procedure Division the programmer can control the segmentation of the Procedure Division. (These segment numbers are called "COBOL segment numbers" in this manual to differentiate them from "internal segment numbers" and "LINKER segment numbers" - see Chapter 3, Linking).

COBOL segment numbers can have a value from 0 to 99 inclusive. For each set of sections with the same COBOL segment number, provided this number is greater than the SEGMENT-LIMIT value, the compiler generates object code in a single segment. One segment is generated for each segment number greater than the SEGMENT-LIMIT. Sections can be grouped into separate segments in this way irrespective of whether they are physically contiguous in the Procedure Division. Sections whose segment numbers are less than the SEGMENT-LIMIT are all grouped into a single segment. All of the above segments may be further subdivided by the compiler if the segment size exceeds the preferred procedure segment size (see below).

According to the ANS standard, a segmented COBOL program can contain three types of segment:

- Fixed permanent segments: these are identified by a COBOL segment number from zero up to but not including a value specified as the SEGMENT-LIMIT (given in the OBJECT-COMPUTER paragraph);
- Fixed overlayable segments: these are identified by a COBOL segment number from the SEGMENT-LIMIT to 49 inclusive; if overlaid, such segments are returned to memory in the state they had when last used (in particular, the results of ALTER and PERFORM statements are retained);
- Independent segments: these are variable overlayable segments, identified by a segment number from 50 to 99; they are in their original state (as compiled), each time they enter memory.

VMM eliminates the distinction between "permanent" and "overlayable" segments: both are of the same type, and both are swappable. Only the distinction between fixed and independent segments is significant, determining the handling of ALTER, PERFORM, MERGE and SORT statements.

If the programmer does not use COBOL segment numbers, it is unlikely that the compiler will segment the Procedure Division in an optimal manner. In the absence of COBOL segment numbers, the compiler may segment in the middle of a frequently used iterative sequence of code (i.e. a loop). This can impair program performance by increasing unnecessarily the swapping or memory requirement during execution. It is important, therefore, that the programmer carefully control the segmentation of the Procedure Division.

In order to use COBOL segmentation effectively, the Procedure Division should be divided into a set of code "modules". These modules should represent logically discrete steps in the overall processing and should be reasonably self-contained. The modules should be organized into a tree structure or hierarchy. In other words, the program should be "structured". The subject of structured programming will be discussed in the Chapter 13 of this manual (concerning new COBOL 85 features), but for a complete description of the method, the reader is recommended to consult one of the many publications on this subject.

Each Procedure Division module should be written as a separate section or contained program and should be activated by a PERFORM or CALL statement. Sections which are closely related and which are normally executed at the same stage of program execution should be grouped in the same segment by being given the same COBOL segment number.

For example consider a program in which there are five sections and COBOL segment numbers 40,41,42 and 43 are used together with a SEGMENT-LIMIT less than 40.

- Segment 40 contains a section to open files and initialize data items.
- Segment 41 contains two sections: a file processing section and a record processing section.
- Segment 42 contains a section to close files and terminate the program.
- Segment 43 contains a section to handle error situations detected in any other segment.

Segments 40 and 42 are each used once only at different times during program execution. Therefore, they are separate segments. Segment 41 contains two sections which comprise the most frequently used instruction sequences in the program. This segment is the only one that needs to be in memory during most of the program execution. Segment 43 is activated whenever an error situation is detected in one of the other segments. This segment might never be executed, but, if it is executed, it will execute at the same time as one of the other segments. This segmentation is depicted in figure "Procedure Division Segmentation" below.

This figure also shows an example of bad segmentation. The main processing sections (2 and 3) are here split over three segments (40, 41 and 42). This means either that sections 1 and 4 will be in main memory even though they are not being used or that segments 40 and 42 will be swapped each time a file is referenced or a record is processed.

Note that there are ANS standard restrictions on the use of the following statements in programs that are segmented using the COBOL segment number in section headers:

- ALTER
- PERFORM
- SORT
- MERGE

These restrictions are described in the *COBOL 85 Reference Manual*.

<table>
<tr><th>Good Segmentation</th><th></th><th>Original Program</th><th></th><th>Bad Segmentation</th></tr>
<tr><td>Segment 40</td><td></td><td>Section 1 - Open files & initialize data items</td><td></td><td rowspan="2">Segment 40</td></tr>
<tr><td rowspan="4">Segment 41</td><td rowspan="4"></td><td></td><td rowspan="4"></td></tr>
<tr><td>Section 2 - File Processing</td><td rowspan="3">Segment 41</td></tr>
<tr><td>Section 3 - Record processing</td></tr>
<tr><td></td><td rowspan="2">Segment 42</td></tr>
<tr><td>Segment 42</td><td></td><td>Section 4 - Close files & terminate</td><td></td></tr>
<tr><td>Segment 43</td><td></td><td>Section 5 - Error handling</td><td></td><td>Segment 43</td></tr>
</table>

Procedure Division Segmentation

7.2.2 Data Division Segmentation

Data internal to INITIAL programs or programs contained in an INITIAL program are allocated in the stack segment. Other data are allocated in data segments; only these data are concerned by the following.

The programmer does not have much control over the segmentation of the Data Division. The main method of controlling this is via the preferred data segment size (see below). However, the following points may be kept in mind, but should not necessarily hide other considerations that might be of more importance:

- Group into a single segment all data likely to be used at a given time by the same statement, and if possible by the same sequence of statements, bearing in mind that file record areas are always in the first segment, and/or:
- Describe all frequently used data at the start of the Data Division so that this data will all be included in a single segment (provided that the preferred data segment size is high enough).

7.2.3 Preferred Segment Sizes

As mentioned above, segments derived from the Procedure Division and Data Division can be further subdivided by the compiler if the actual segment size exceeds the preferred segment size. Preferred segment sizes can be specified in the following ways:

- in the OBJECT-COMPUTER paragraph using the MAXIMUM PROCEDURE SEGMENT SIZE and MAXIMUM DATA SEGMENT SIZE clauses;
- in the CBL JCL statement using the PSEGMAX and DSEGMAX parameters.

The use of the CBL JCL parameters is recommended, as the MAXIMUM PROCEDURE SEGMENT SIZE and MAXIMUM DATA SEGMENT SIZE clauses are not part of the ANS standard. Remember that the maximum size of a procedure segment is 64K bytes, and that the maximum size of a data segment is 4096K bytes.

If preferred segment sizes are not specified, the compiler will assume a default value of 4K bytes (K = 1024). In most cases the default segment size will be acceptable and a size need not be specified by the programmer. However, the following points should be remembered when deciding upon the optimum preferred segment sizes.

- Performance will be improved if all segments in the load module are of approximately the same size.
- Large segments tend to be used (i.e., referenced) more often than small ones. For this reason VMM usually allows them to remain in memory for a longer time than small segments. On the other hand, once a large segment has been swapped out of memory considerable rearrangement of memory contents might be necessary in order to provide a large enough area of memory for it to be swapped back into.
- Conversely, smaller segments tend to remain in memory for a shorter time. The CODAPND parameter of the CBL JCL statement can be used to merge linkage and code segments when the code segment is small (see Compilation, Chapter 2).

7.3 Automatic Segmentation

The compile unit generated by the COBOL compiler normally comprises a minimum of three segments as shown in the following segment list printed by the compiler:

```
IC206.0        ..L             101
IC206.1        .D.            1342
IC206.2        C..            1946
```

The meaning of L,D and C is as follows:

- L indicates a linkage segment. This segment, during execution, will contain all the pointers required for the calls and branches in the program. It also contains certain constants. There is only one linkage segment in each compile unit and it always has an internal segment number zero.
- D indicates a data segment. There may be one or more data segments in a compile unit. These segments contain the record areas defined in the program FDs together with the contents of the Working-Storage, Constant and Communication Sections. Certain compiler generated data is also stored in the data segments.
- C indicates a code segment. There may be one or more code segments in a compile unit. These segments contain the object code generated from the statements in the Procedure Division.

If the CODAPND parameter is used in the CBL JCL statement, and the total size of the last code segment of the fixed permanent segments and the linkage segment together is not larger than 64K bytes (K=1024), the linkage segment and this code segment will be merged. The above example segment list would thus appear as follows:

```
IC206.0        C.L            2047
IC206.1        .D.            1342
```

If the preferred data segment size or procedure segment size is exceeded, the compiler will divide the data or code into one or more segments. Circumstances under which this "automatic" segmentation takes place are described below.

7.3.1 Data Segments

Data segments are generated according to the following rules.

- Record areas for the program's files are located at the start of the user data in the first data segment.
- 01 level data items are added one by one to the data segment until the preferred data segment size is reached. At this point a new segment is started. 01 level data items are not split between two segments.
- If an 01 level data item is, individually, greater than the preferred data segment size it is not subdivided but forms a data segment on its own.
- If the compiler has generated any incomplete data segment (less than the preferred data segment size) it will try to insert later 01 level data items into the last segment until the preferred data segment size is reached. If it does not fit, then the previous segment is tried, and so on.

The application of these rules can be observed in the following example. In this example segment 1 contains two record areas (80x2), the group item MAN (80) and three elementary items (1000x3) or about 3240 bytes. Segments 2 and 4 each have a length of exactly 4000 bytes. However, segment 2 is composed of three contiguous data items (DD,EE,FF) and one data item which is not contiguous with the others (HH). This is because a data item of 5000 bytes (GG) occurs between FF and HH. This item is assigned the whole of segment 3. Segment 5 has a length of only 3000, since the next level 01 item is too big, and forms a segment on its own, as does the last level 01 item.

The source program is as follows:

```
DATA DIVISION.
FILE SECTION.
FD  LIST.
01  ARTOUT     PIC X(80).
FD  CARD.
01  ARTIN   PIC X(80).          Internal seg. no 1
WORKING_STORAGE SECTION.
01  MAN.
  02  CODNB PIC 9.
  02  L1    PIC X(10).
  02  L2    PIC X(69).
01  AA      PIC X(1000).
01  BB      PIC X(1000).
01  CC      PIC X(1000).

01  DD      PIC X(1000).
01  EE      PIC X(1000).       Internal seg. no 2
01  FF      PIC X(1000).

01  GG      PIC X(5000).       Internal seg. no 3

01  HH      PIC X(1000).       Internal seg. no 2 (cont)

01  II      PIC X(1000).
01  JJ      PIC X(1000).       Internal seg. no 4
01  KK      PIC X(1000).
01  LL      PIC X(1000).

01  MM      PIC X(1000).
01  NN      PIC X(1000).       Internal seg. no 5
01  OO      PIC X(1000).

01  PP      PIC X(3000).       Internal seg. no 6


01  QQ      PIC X(3000).       Internal seg. no 7


PROCEDURE DIVISION.
```

The corresponding segment list printed by the compiler is as follows:

```
SEGLIM01.0        ..L            129
SEGLIM01.1        .D.           3592
SEGLIM01.2        .D.           4000
SEGLIM01.3        .D.           5000
SEGLIM01.4        .D.           4000
SEGLIM01.5        .D.           3000
SEGLIM01.6        .D.           3000
SEGLIM01.7        .D.           3002
SEGLIM01.8        C..            144
SEGLIMO1.9        C..            464
SEGLIMO1.10        C..             352
SEGLIM01.11        C..             592
```

7.3.2 Procedure Segments

The Procedure Division is divided automatically into segments according to the preferred procedure segment size. As mentioned above this can cause the object code for a frequently used iterative sequence of code (a loop) to be divided between two segments. In the worst case, when there is memory overload, this can result in both segments being swapped each time the loop is executed.

The procedure map listing printed by the compiler may help in determining the preferred procedure segment size. This listing can be used to find the source line number at which a new segment begins.

7.4 Internal Segment Numbers

A COBOL compile unit can have no more than 256 internal segment numbers (ISN). The ISN is the number assigned to each internal segment in a compile unit. These numbers are assigned in the following way.

- 4 ISNs are allocated to special segments not generated directly from the Procedure Division or Data Division.
- Each data segment generated from the Data Division is given an ISN.
- Each procedure segment generated from the Procedure Division is given an ISN.
- The linkage segment is given an ISN unless this segment is merged with a code segment (CODAPND parameter in the CBL JCL statement).
- Each file in the program uses two ISNs.
- If the Program Checkout Facility is to be used (DEBUG parameter in CBL JCL), one segment is generated in the compile unit for each 200 lines (approximately) in the source program. One ISN is given to each segment.
- If the program contains a USE FOR DEBUGGING Section and the DEBUGMD parameter of the CBL JCL statement is active, one or more additional segments are generated. One ISN is given to each segment.
- If a program has External files, one ISN is used for each External file.
- If one of the following cases occurs, one extra ISN is used:
  - an External data item with a VALUE clause is present.
  - a translation table (used for alphabet-name, TRANSFORM...) is used.

If more than 256 ISNs are needed during a compilation, the compiler will print the fatal error message 8-94 and will then terminate.

7.5 Declared Working Set

To avoid memory overload situations, the amount of memory required for each job step should be specified by use of the SIZE JCL statement. The amount of memory required is known as the declared working set (DWS). If no DWS is specified, a default value applies.

8. Efficiency Techniques

The following techniques are recommended to obtain efficient COBOL object programs. Consideration is given to data manipulation and data description techniques. See Chapter 7, Segmentation, for guide-lines on efficient segmentation. See the *UFAS-EXTENDED User's Guide* for guide-lines on the efficient use of files.

Some of the suggestions are designed to reduce memory needs, some are meant to save time, and some will do both. Each recommendation is followed by the designators (T) for time saving, (S) for space saving, or (T and S) for time saving and space saving, to indicate the anticipated type of efficiency.

Note that some of the suggestions recommend the use of language features that are not part of the ANS standard (e.g. COMP-9, COMP-10...).

8.1 Data Manipulation Techniques

- Avoid using the CORRESPONDING option when a simple MOVE statement would suffice. MOVE CORRESPONDING results in a series of moves of individual items; a simple MOVE is instead optimized for the group or record as a whole. Never use MOVE CORRESPONDING for such purposes as transmitting a master file record from the input buffer to the output buffer. Use MOVE CORRESPONDING when it will in fact cause selected items to be moved, or when editing or format conversion is needed on the respective items. (T and S)
- Manipulate a group item or record as a whole whenever possible, rather than manipulating its elementary items separately. This rule is especially important for tables of data items; MOVE or clear a table as a whole whenever possible.

  For example, technique a (below) is quite efficient, while b is less so:

```
a.            MOVE SPACES TO TABLE.

b.            MOVE 1 TO I.
     LOOP.    MOVE SPACES TO TABLE-ITEM (I).
              ADD 1 TO I.
              IF I NOT > TABLE-SIZE GO TO LOOP.
```

  (T and S)

- If a data item is to be used in several subscripts without a change in value, either make it a usage BINARY item or else move it to a temporary area in Working-Storage (described as BINARY) and use the Working-Storage data item in the subscripts.
  (T and S)
- If the length of the repeating data item in a table is a power of 2, use the SUBOPT parameter of the CBL JCL statement. This will enable the compiler to use shift rather than multiply and bound check when calculating the displacement. However, see "Exception 06-00 Out of Segment Bounds", Chapter 4, for a restriction on the use of SUBOPT. (T and S)
- If a subscripted item is to be referred to more than once with the same subscript value(s), consider moving it to a temporary area once for all processing. Or: If a subscripted item is to be referred to more than once, SET an INDEX for this element and use this INDEX as a subscript. If this is done the displacement of the element need not be calculated for each reference. (T and S)
- For MOVEs, conditions, additions, and subtractions, give the items similar PICTURE clauses and USAGE clauses whenever possible. (T)
- In the UNTIL option of the PERFORM statement, use the simplest possible condition to terminate the loop. If necessary, achieve such simplicity by preceding the PERFORM with explicit MOVEs and COMPUTEs. If numeric items are involved in the condition, give them similar PICTURE clauses and the same USAGE clause. (T)
- Tend to use procedural literals rather than constant values. The compiler can optimize the format of procedural literals, but must resort to dynamic format conversions in the object program if data items are not ideally formatted. However, duplicate literals do result in extra memory space requirements. (T)
- Use `GO TO... DEPENDING` or equivalent EVALUATE for decisions whenever possible. In any application for which such construction can be used, more efficient object coding can be generated than by using a succession of IF statements. (T and S)
- `ADD 1 TO A` is equivalent to `COMPUTE A = A + 1`, but `MULTIPLY A BY B` may be better than `COMPUTE B = A * B`. (T)
- When the result of a computation is stored in one of the operands of the computation, ADD, SUBTRACT etc. may be more efficient than COMPUTE. For example, `ADD A TO B` may be more efficient than `COMPUTE B = A + B`. (T)

- If possible, use the DIVIDE statement rather than / in the COMPUTE statement. The compiler will convert operands and intermediate results into floating point decimal whenever division or exponentiation occurs in the COMPUTE statement (unless the division is the last operation in the statement). This is time consuming and can be avoided for division by using the DIVIDE statement. However, COMPUTE must be used for exponentiation (**). The effects of converting into floating point decimal can be minimized by ensuring that division or exponentiation are the last operations to be performed in a COMPUTE statement. For example:

  ```
  COMPUTE T = A**B.
  COMPUTE R = T+C.
  ```

  is more efficient than:

  ```
  COMPUTE R = A**B+C
  ```

  In the first method a temporary data item T is used to hold the result of the exponentiation. C is added to T in a separate COMPUTE which is performed in fixed point decimal. In the second method both the exponentiation and the addition are performed in floating point decimal. Note the following example also:

  ```
  COMPUTE R = (A*B)/C
  ```

  is more efficient than:

  ```
  COMPUTE R = (A/C)*B
  ```

  In the first method the intermediate result may be in fixed point decimal. In the second method it will be in floating point decimal for the whole computation because the division is the first operation to be performed (T).

- Avoid using the / operator in an arithmetic expression of a relation condition (T and S).
- For exponentiation, use COMP-9, COMP-10 or COMP-15 operands or restrict the expression to a single exponentiation (T).

- Use the TEMP clause of the Default Section of the Control Division, or the TEMP= parameter of the CBL JCL statement, whenever the precision of intermediate results is not crucial (T). For example:
  - using TEMP IS 15 or less in the Default Section of the Control Division, or TEMP=15 or less in the CBL JCL statement, may speed up expressions containing division;
  - using TEMP IS 13 in the Default Section of the Control Division, or TEMP=13 in the CBL JCL statement, may speed up expressions, especially those containing exponentiation;
  - using TEMP IS 13 NOT STANDARD in the Default Section of the Control Division, or TEMP=13NS in the CBL JCL statement, may also speed up expressions containing division.
  - Using TEMP IS BINARY in the Default Section of the Control Division, or TEMP= BIN in the CBL JCL statement allows intermediate results of arithmetic expressions be computed in floating binary format; this will speed up execution of complex arithmetic expressions with slightly less accurate results.
- When imperative-statements in several WHEN phrases of an EVALUATE statement logically terminate with identical statement series, removing the common statement series from the EVALUATE statement will save no generated code. It could be better to repeat the common statement series as many times as necessary in the EVALUATE statement rather than using GO TO or PERFORM statements, if this improves the legibility of the programs or if the use of paragraphs that are not logically necessary destroys the program structure (S).

  For example, the following EVALUATE statement:

```
EVALUATE CODE
    WHEN  1      MOVE  A  TO  X
                  MOVE  1  TO  Y
                  MOVE  2  TO  Z
    WHEN  2      MOVE  B  TO  X
                  MOVE  4  TO  Z
    WHEN  3      MOVE  1  TO  Y
                  MOVE  2  TO  Z
    WHEN  4      MOVE  B  TO  X
                  MOVE  5  TO  Y
                  MOVE  4  TO  Z
    WHEN  5      MOVE  6  TO  Y
                  MOVE  2  TO  Z
END-EVALUATE
```

will be generated as:

```
   EVALUATE  CODE
      WHEN  1      MOVE A TO X GO TO INTERNAL-LABEL-1
      WHEN  2      MOVE B TO X GO TO INTERNAL-LABEL-2
      WHEN  3                  GO TO INTERNAL-LABEL-1
      WHEN  4      MOVE B TO X
                   MOVE 5 TO Y GO TO INTERNAL-LABEL-2
      WHEN  5      MOVE 6 TO Y GO TO INTERNAL-LABEL-3
   END-EVALUATE
   GO TO INTERNAL-LABEL-4.
INTERNAL-LABEL-1.
   MOVE 1 TO Y.
INTERNAL-LABEL-3.
   MOVE 2 TO Z.
   GO TO INTERNAL-LABEL-4.
INTERNAL-LABEL-2.
   MOVE 4 TO Z.
INTERNAL-LABEL-4.
```

`INTERNAL-LABEL-1, -2, -4` are names produced by the compiler; they do not interfere with the program structure described by the user.

Note that only identical sequences that are at the end of the imperative - statements are recognized by the compiler. Thus, in the above example, "`MOVE B TO X`" in the second and the fourth WHEN phrases cannot be optimized.

8.2 Data Description Techniques

- Use PACKED-DECIMAL, COMP, COMP-3 or COMP-8 for non-integer data items and or data items which interact with other PACKED-DECIMAL, COMP, COMP-3 or COMP-8 data items. PACKED-DECIMAL, COMP, COMP-3 or COMP-8 must be used if fractional results are required (T).
- Do not use unsigned COMP or COMP-3 data items in computations. The right-hand 4 bits of a packed decimal item represent the sign. When an unsigned COMP or COMP-3 data item is used in a computation a dummy sign position must be added to the data tem before computation can be started. If unsigned data items must be used, they should be described as PACKED-DECIMAL (T and S).
- Use BINARY, COMP-1 or COMP-2 for integer data items which are not involved arithmetically with data items of other usages (T).
- Specify BINARY, COMP-1 or COMP-2 for a data item that will be used as a subscript or that will be a DEPENDING item in a GO TO statement or in an OCCURS clause. This rule is important if the item will be mentioned as a "subscript-name" in PERFORM... VARYING or in any such loop. Again, consider moving the item explicitly to a BINARY, COMP-1 or COMP-2 area in Working-Storage if other considerations dictate USAGE DISPLAY. (However, INDEX is a standard and possibly more efficient method of describing subscripts used in PERFORM... VARYING.) (T).
- USAGE BINARY, COMP-1 or COMP-2 is also recommended for identifier-2 data items in WRITE... ADVANCING statements to avoid unnecessary conversions (T).
- If a record contains BINARY, COMP(-n) and/or SYNCHRONIZED data items, place single-word items and double-word items together whenever possible. Savings in memory space can be obtained; this rule is most applicable for records in a file (S).
- When Boolean data items have more than one Boolean position or when consecutive Boolean data items are contained in the same record, use usage BIT rather than usage DISPLAY. Where a PIC X data item is traditionally used to represent an item that has only two values (e.g. "Y" for "YES" and "N" for "NO"), using usage BIT Boolean data item (PIC 1) will save space. Legibility of the program can be kept and even improved by using a level-88 condition name together with the SET TO TRUE/FALSE statement (S).

- It is often necessary to organize files in a highly efficient space-saving manner, even though it is also desired to save time while processing the data. In this case, describe each record in both the File Section and in the Working-Storage Section. In the File Section, pack the data as closely as possible, without regard to processing efficiency; in the Working-Storage Section, do exactly the opposite. Avoid using READ... INTO and WRITE... FROM. Instead, READ each record and determine whether the record is to be involved in detailed processing. If detailed processing is required, employ the MOVE... CORRESPONDING statement to unpack either the entire record or the significant group(s) within it to the Working-Storage area and refer to the data in that location for all detailed processing. Similarly, use MOVE... CORRESPONDING as appropriate to construct (or reconstruct) the output record. Perform a simple MOVE from input buffer to output buffer if detailed processing is not required (T and S).
- If reports are generated without using the Report Writer facility, use skeleton lines in Working-Storage, with constant information initialized via the VALUE clause rather than by MOVE statements in the Procedure Division (T and S).
- There is some benefit in having BINARY, COMP-1, COMP-2, COMP-9, COMP-10, COMP-15 and INDEX items word-aligned. However, it is not worth using SYNC to achieve this if it cannot be done by arranging the order of other data items (T).
- For programs compiled with CODE=OBJA, running in a class D hardware, or vice versa, which does not contain floating point data, but contains exponentiation, the executing speed may be reduced by a factor 100. See the description of the CODE parameter in Chapter 2.

8.3 Value of DSEGMAX

The best value for DSEGMAX is 32K. The use of a lower or higher value may increase both the length and the time of execution of the program.

8.4 Using the INITIAL Clause in the IDENTIFICATION DIVISION

The use of the INITIAL clause in the IDENTIFICATION DIVISION may result in an inefficient program.

If too many data items are declared, the length of the used stack may be too large.

9. Files

This chapter contains information that is relevant for all types of files. In addition, the use of unit record files is discussed in detail in Chapter 11.

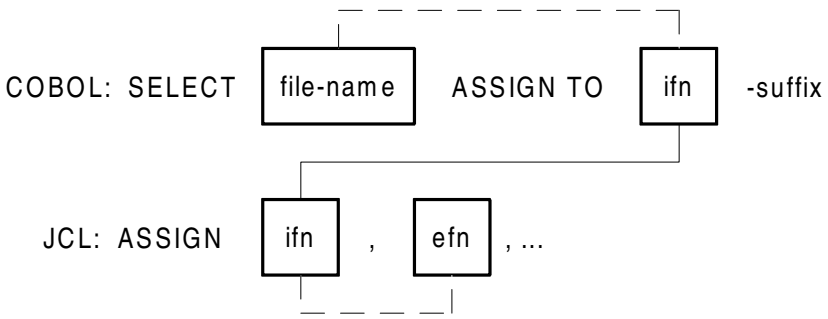
We will use here the JCL form. The equivalent GCL will be found in the *IOF Terminal User's Reference Manual*.

9.1 File Names

Each file is known by three different names:

- COBOL file-name, used to identify the file throughout the COBOL program;
- internal-file-name (ifn), used to connect the COBOL file-name and the external-file-name via the COBOL SELECT clause and the JCL ASSIGN statement;
- external-file-name (efn), the name by which the file is known to the system; it is recorded in the file label and possibly in a catalog.

The relationship between these three file names is shown in the following diagram:



The advantage of assigning the efn to the ifn is that different files can be processed with the same COBOL program, merely by modifying the ASSIGN JCL statement each time the job is to be run. Any file may also be dynamically assigned, see *Dynamic File Assignment* in the current chapter.

Note that subfiles or library members can be assigned to files described in COBOL programs. If the file is described in a SELECT entry with QUEUED in the ORGANIZATION clause, then you must not specify the subfile name in the ASSIGN JCL statement; otherwise, you must specify the subfile name in the ASSIGN JCL statement. See Chapter 10 ("Standard Record Formats") for details, and see "QUEUED FILE PROCESSING" in the current chapter.

The format and content of each type of file name are different. They are as follows:

- ANS standard calls for COBOL file-names of not more than 30 characters chosen from the set A...Z, 0...9, and hyphen (-), and containing at least one alphabetic character.
- Internal-file-names may be up to 8 characters in length, chosen from the set A...Z, 0...9. They must begin with an alphabetic character and may NOT contain a hyphen (except for H-SORT, which may be used only as the internal-file-name of a sort/merge file). The character "_" (underscore) is not an ANS standard COBOL character. It can occur in an internal-file-name only if it begins with the characters H_. It is reserved for system files, for example H_RD and H_PR (used for ACCEPT FROM SYSIN and DISPLAY UPON SYSOUT). Note that this definition of an internal-file-name only applies to COBOL and is more restrictive than the definition in the *JCL Reference Manual*.
- The rules for external-file-names are different for catalogued files, uncataloged files and temporary files. See the *JCL Reference Manual* for details of external-file-names.

The following suffixes may be used on internal-file-names in the COBOL SELECT clause:

- PRINTER
- MSD
- CARD-READER
- CARD-PUNCH
- TAPE
- SYSIN
- SYSOUT

No other suffix may be used. However, only -PRINTER, -SYSIN and -SYSOUT have a significance other than documentation. The significance of -PRINTER and -SYSOUT is explained in "Writing SSF Files in COBOL Programs", Chapter 10. The only significance of -SYSIN is that the associated file can be opened when it is already open. Note that the suffix is NOT part of the ifn, and does not appear in the corresponding ASSIGN JCL statement (if any).

9.2 Data Management Overriding Rules

The source program normally defines the basic file characteristics. However, a number of file parameters (e.g., file organization, block size, device class) can be specified and recorded at different places in the system and at various stages in the creation of the file. The table below shows where these parameters are specified and where they apply.

Specification and Applicability of File Characteristics

Where Specified	Where Applicable
At system generation	Throughout the job
In source program	Compilation time
In JCL (ASSIGN JCL & DEFINE)	Job translation time
File label	File OPEN time

Overriding rules specify the action to be taken by Data Management when there is an absence, or multiplicity, of parameters. They define the final values to be used for file processing and detect violations (FATAL when a choice cannot be made, or WARNING where the decision is made by Data Management).

Data Management overriding rules may be summarized as follows:

- Basic file parameters are specified in the COBOL program, and override (and/or complement) any system values that apply by default.
- File parameters specified in the COBOL program are overridden by any JCL parameters.
- File parameters specified in the JCL are overridden by parameters in the file label if the file exists at OPEN time.

For a detailed description of Data Management overriding rules see the *UFAS-EXTENDED User's Guide*.

Opening a file in a COBOL program provides checks on the record length, which are additional to the general Data Management overriding rules. These checks are discussed in the following paragraphs.

Note that for these checks, DEFINE JCL statements are not taken into account. Checks are done between the program values and the actual values resulting from the overriding rules.

The maximum record length (See "Record Size" below) of the file must be the same as the maximum record length declared in the program that opens the file (apart from the exceptions listed below) otherwise the following message appears in the JOR:

```
CBL15. IFN:ifn RECORD LENGTH CONFLICT (length ON FILE)
```

followed by the EX03. UNEXPECTED RETURN CODE message showing the mnemonic "COBOL 1, RECERR". The file status "39" is returned to the program.

If however the program bypasses this error through a USE procedure, the result is somewhat unpredictable when records are actually larger than the record area specified in the program. In general, records will be truncated, and the following may happen:

- on a READ statement, a file-status "04" is returned
- on a WRITE statement, a file-status "44" is returned if the file is a variable length file, but "00" is returned if the file is a fixed length file.

The exceptions are:

- The file is assigned to SYS.IN or SYS.OUT: no checking is done
- The ifn in the SELECT clause for the file is suffixed by -SYSOUT: no checking is done
- The file is opened in input; the following message appears in the JOR:

  ```
  CBL15. IFN:ifn RECORD LENGTH CONFLICT ACCEPTED IN INPUT
                 (length ON FILE)
  ```

  A normal ("00") file-status is returned to the program. If a record is read whose actual length is greater than the length of the largest record described in the program, the record is truncated, and a file-status "04" is returned to the program.
- The file is opened in output and the RECFORM=UNDEFINED parameter is used in the DEFINE JCL statement. The following message will appear in the JOR:

  ```
  CBL15. IFN:ifn RECORD LENGTH CONFLICT ACCEPTED FOR NSTD
                 (length ON FILE)
  ```

  A normal ("00") file status is returned to the program.

Note that when a member of queued file created by a LIBALLOC JCL statement is assigned to a COBOL file, the implied record size for the physical file is 264. The size of the record as determined from the program (see "Record Size" below) must therefore be 264, for an SSF file the user data must be 256.

The actual file organization that may be associated with a file is shown in the table "Permitted File Organizations" below.

For a file whose ORGANIZATION IS INDEXED in the program, the number of record keys, their position relative to the beginning of the record, and their length must be the same for the file and the program.

Permitted File Organizations

Organization in COBOL program	Type of file				
	UFAS			BFAS	
	Seq	REL	IND	Seq	Que
Sequential no qualifier UFF	1	2	2	2	1
QUEUED					1
Relative no qualifier UFF	3	1		3	3
INDEXED no qualifier UFF			1		

Notes for this table:

The access types referred to in the column headings are:

- Ind - Indexed
- Que - Queued
- Rel - Relative
- Seq - Sequential

The numbers in the body of the table refer to the following notes.

1. Allowed.
2. Allowed if the file is opened in INPUT or I-O.
3. Allowed for a disk file when ACCESS MODE IS SEQUENTIAL for a file opened in INPUT or I-0, or, when ACCESS MODE IS RANDOM or DYNAMIC, for a file opened in INPUT (the RELATIVE KEY clause can only be used if the RECFORM is FB).

   Allowed for a diskette file when ACCESS MODE IS SEQUENTIAL, or when ACCESS MODE IS RANDOM for a file opened in INPUT, or when ACCESS MODE IS RANDOM for a file opened in OUTPUT or I-O provided that foreign access is specified, or when ACCESS MODE IS DYNAMIC for a file opened in INPUT provided that foreign access is not specified or for a file opened in OUTPUT provided that foreign access is specified.

9.3 Optional Files

An optional file is one which may be absent at execution time, even though I/0 operations such as OPEN, CLOSE, READ, WRITE may be attempted for the file.

A SELECT clause containing the OPTIONAL phrase must be used for each optional file. There must also be an ASSIGN JCL statement for each such a file. The ASSIGN statement may contain the OPTIONAL or DUMMY parameter, as explained below.

From the point of view of the ANS standard COBOL, any optional file (i.e., the OPTIONAL parameter is present in the SELECT clause) must be used for files opened in input, I-O or extend mode.

From the system point of view, all files may be optional. A distinction is made between those that are declared absent when the JCL execution is written, and those that are declared absent by the operator.

Files which are declared optional in the COBOL program must have a corresponding ASSIGN JCL statement in the execution JCL, regardless of whether the file is to be used or not. The ASSIGN JCL statement will normally include a DUMMY or OPTIONAL parameter (discussed below). If neither of these parameters is present in the ASSIGN JCL statement, the OPTIONAL parameter in the COBOL SELECT clause is ignored and the file is processed normally. On the other hand, if either DUMMY or OPTIONAL is specified in the ASSIGN JCL statement and the OPTIONAL phrase is not present in the corresponding COBOL SELECT clause, a file status "35" is returned.

- JCL declaration using the ASSIGN statement DUMMY parameter:

  ```
  ASSIGN ifn, DUMMY;
  ```

  This specifies that the file is absent. All references to the file should be ignored, except when an INVALID KEY or anAT END condition arises.

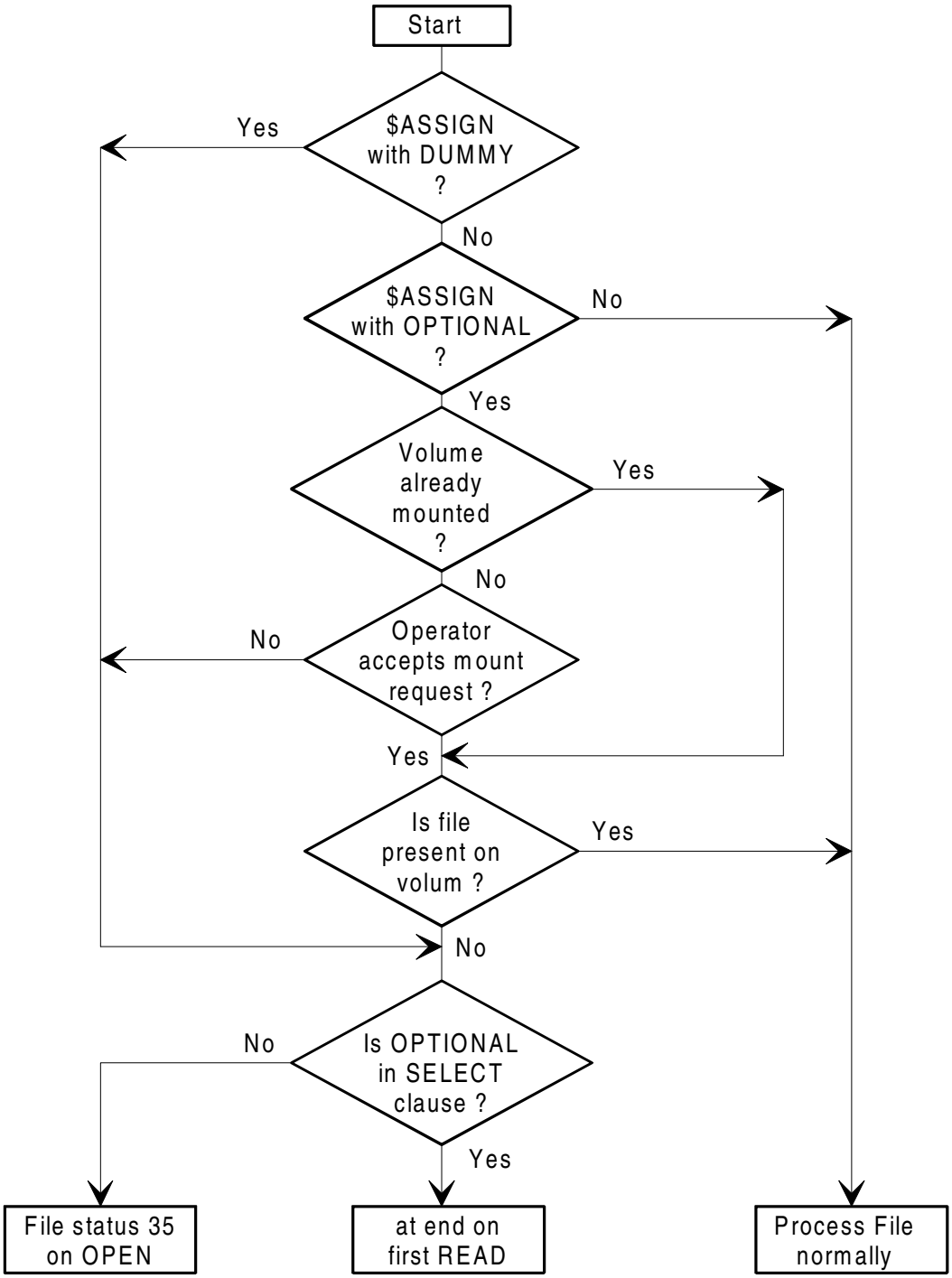
- JCL declaration with operator intervention:

  The JCL provides for the possibility of file absence by the ASSIGN statement OPTIONAL parameter. For example:

  ```
  ASSIGN ifn,efn, DEVCLASS..., MEDIA..., OPTIONAL;
  ```

  In this case, at step execution the system searches the volume named in MEDIA for this file. If the volume is absent, a MOUNT request is sent to the operator. If the operator refuses MOUNT (CR MS...), or if the file is absent from the volume mounted, the file is considered as DUMMY and processed as described above. If the volume is mounted and the file exists, the file is processed as normal.

The use of optional files is summarized in the following figure.



The Use of Optional Files

9.4 CLOSE WITH LOCK

If the WITH LOCK option is used to close a file, Data Management assumes that no further processing of the file is to take place in the current program. Data Management will therefore prohibit re-opening of this file in the same STEP. The resources associated with the file will be returned to the system and may be assigned by another job. Nevertheless, if an OPEN is executed, file status "38" (ifn not assign) is returned to the user, and the step is terminated (unless there is a USE AFTER ERROR PROCEDURE section in the Declaratives).

9.5 The POOL JCL Statement

If a program accesses two or more files that are on different volumes, one device will normally be allocated at step initiation for each volume to be accessed. However, if files on separate volumes are never processed at the same time by the program, only one device is required for these volumes. For example, suppose that an executing COBOL program contains the following statements:

```
SELECT  FILE1  ASSIGN  TO  F1.
SELECT  FILE2  ASSIGN  TO  F2.
.
.
OPEN INPUT FILE1.
.
.
CLOSE FILE1 WITH LOCK.
OPEN INPUT FILE2.
.
.
```

In this example, file FILE1 is completely processed before processing begins on file FILE2. If these files are on different volumes, it will be possible to use the same device for both volumes. The user can inform the system that this is required by using the POOL JCL statement in conjunction with the POOL parameter in the ASSIGN JCL statement:

```
POOL 1*MS/M402;
ASSIGN F1,  MAX.Z,  POOL, FIRST...
ASSIGN F2,  BMY.I,  POOL, NEXT...
```

Thus, a device pool of one MSU0402 will be reserved for the volumes which contain these files. When the program closes FILE1, the system is able to free the device allocated to the associated volume. If desired, more than one device can be requested in the POOL JCL statement. For example, if three or more volumes were to be processed, two devices could be requested so that, whilst one device was being used by the program, the operator could reload the other device with the next volume to be used.

In the above COBOL example, the file FILE1 itself has been de-assigned by the inclusion of WITH LOCK in the CLOSE statement. This action is not necessary for the purposes of the device pool, but it ensures that FILE1 cannot be re-opened in the same job step and therefore the corresponding file can be assigned immediately, if necessary, to another job.

For further information on the use of the POOL JCL statement, see the *JCL User's Guide*.

9.6 Multi-Volume Files

If a file is too big to fit on a single disk or tape volume, it can be stored on more than one volume. Such a file is called a multi-volume file. The following paragraphs are applicable to sequential multi-volume files only. Each part of such a file, disk, or tape, is known as a "physical unit".

In the case of disk volumes that contain other files also, only part of each volume will be occupied by a physical unit of the multi-volume file. The space for disk files is allocated using the PREALLOC utility (see the *Data Management Utilities User's Guide*). Only one ASSIGN JCL statement is needed for a multi-volume disk file or multi-tape file. The volume-names must be listed in the MEDIA parameter of the ASSIGN JCL statement or recorded in the catalog entry for the file.

Boundaries between volumes in a multi-volume file are usually invisible to the COBOL program that processes such a file. The COBOL program does not have to contain any special code to handle multi-volume files: when the end of one volume is reached the system automatically switches to the next volume of the file. However, an end-of-volume condition (i.e. end of physical-unit) can be forced during sequential input or output by using the CLOSE REEL or CLOSE UNIT statement. These statements both have the same effect. They cause processing of the current volume to cease and the next volume to be opened. This works only for multi-volume tape files.

The end of "physical unit" is also visible to the COBOL program when a RERUN ON CHECKPOINT-FILE EVERY END OF REEL/UNIT references the file. Under these circumstances the end of a physical unit causes a checkpoint to be taken. This works only for multi-volume tape files.

For more information about multi-volume files, see the *JCL User's Guide*.

9.7 Multi Logical Unit Files

A COBOL program can process several files during a single execution using a single SELECT clause and FD for all the files. Each file can have a different organization (sequential, relative etc.) and may use a different access system (UFAS, ANSI). Each such individual file is called a "logical unit" of the COBOL file defined in a single SELECT clause and FD.

From the point of view of the COBOL program this set of logical units is seen as a single file. From the system point of view, each logical unit is a complete mono-volume or multi-volume file. If a logical unit occupies several volumes, this may be because the logical unit is a "multi-volume file" (see above) or because "file concatenation" is being used (see below). The use of multi logical unit files is described in the following paragraphs.

There is no need to explicitly open and close each logical unit in a multi logical unit file. Each logical unit is opened and closed automatically in a manner analogous to multi-volume file processing. Alternatively, the COBOL program can swap logical units by using the CLOSE REEL or CLOSE UNIT statement. This causes the current logical unit to be closed before the end has been reached; the next logical unit is then opened immediately. Note that, when using multi logical unit files, the CLOSE REEL and CLOSE UNIT statements cannot be used to close physical units (i.e., volumes). When a RERUN ON CHECKPOINT-FILE EVERY END OF REEL/UNIT statement references the file, the end of a logical unit causes a checkpoint to be taken. For such files, physical units, as described above, are not visible from the COBOL program.

There must be an ASSIGN JCL statement in the JCL for each logical unit to be processed. There may also be a POOL JCL statement to allocate a single device for all the logical units being processed. For example:

```
COBOL:
   SELECT INPUT-FILE ASSIGN TO IFILE...

JCL:
   POOL MT/T9;
   ASSIGN IFILE-1,  E.FILEA,  POOL, FIRST...
   ASSIGN IFILE-2,  E.FILEB,  POOL, NEXT...
   ASSIGN IFILE-3,  E.FILEC,  POOL, NEXT...
```

In this example, the internal-file-name is suffixed with a hyphen followed by the sequence number of the logical unit within the file (without leading zeros). Note that the internal-file-name specified in the COBOL SELECT clause must be short enough to permit the addition of the suffix. The maximum length of an internal-file-name including suffix is 8 characters.

If the COBOL program is to process only one logical unit, a normal ASSIGN JCL statement must be used with no suffix on the internal-file-name.

For example:

```
ASSIGN IFILE, E.FILE,...
```

9.8 Multiple File Tape Volumes

Several self-contained files may be written to or read from a single magnetic tape by a COBOL program (only one file may be opened at one time).

There must be a SELECT clause in the FILE-CONTROL paragraph and an ASSIGN JCL statement in the JCL for each file in a multiple file tape volume. There may also be a POOL JCL statement in the JCL to allocate a single device for the magnetic tape. The FSN (file sequence number) parameter must be used in each ASSIGN JCL statement to specify the position numbers of the files. The DEVCLASS and MEDIA must be the same for all the files. For example:

```
COBOL:
  SELECT FILEA ASSIGN TO IFILEA.
  SELECT FILEB ASSIGN TO IFILEB.
  SELECT FILEC ASSIGN TO IFILEC.
JCL:
  POOL MT/T9;
  ASSIGN IFILEA,E.FILEA,DVC=MT/T9,MD=TAPEA,FSN=1,POOL,FIRST;
  ASSIGN IFILEB,E.FILEB,DVC=MT/T9,MD=TAPEA,FSN=2,POOL,NEXT;
  ASSIGN IFILEC,E.FILEC,DVC=MT/T9,MD=TAPEA,FSN=3,POOL,NEXT;
```

The same internal-file-name can be used for all files in a multiple file tape volume if desired. However, in this case, a MULTIPLE FILE TAPE clause must be present in the I-O-CONTROL paragraph in the COBOL program. For example:

```
COBOL:
 SELECT FILEA ASSIGN TO IFILE...
 SELECT FILEB ASSIGN TO IFILE...
 SELECT FILEC ASSIGN TO IFILE...
 .
 .
 MULTIPLE FILE TAPE FILEA POSITION 1, FILEB POSITION 2,
 FILEC POSITION 12.

JCL:
 POOL   MT/T9;
 ASSIGN IFILE-1,E.FILEA,DVC=MT/T9,MD=TAPEA FSN=1,POOL,FIRST;
 ASSIGN IFILE-2,E.FILEB,DVC=MT/T9,MD=TAPEA,FSN=2,POOL,NEXT;
 ASSIGN IFILE-12,E.FILEL,DVC=MT/T9,MD=TAPEA,FSN=12,POOL,NEXT;
```

In this example the internal-file-name IFILE is used for each of the files. However, the internal-file-names in the ASSIGN JCL statements must be suffixed with a hyphen followed by the position number of the respective file within the multiple tape file volume. The compiler will output a message in the program listing for each internal-file-name, showing the suffix that must be used in the ASSIGN JCL statement. Note that the internal-file-name specified in the COBOL SELECT clause must be short enough to permit the addition of the suffix. The maximum length of an internal-file-name (in a COBOL program) including suffix is 8 characters. The DEVCLASS and MEDIA must be the same for all of the files. Finally, the FSN parameter must also be used to specify the position numbers of the files.

If the same internal-file-name is used for all the files in a multiple file tape volume, but only one of these files is to be read during a particular program execution, then the position number suffix need not be used in the ASSIGN JCL statement, though it can be used if desired. A position number suffix can also be specified even when all the internal-file-names are unique.

The files in a multiple file tape volume can be specified in the MULTIPLE FILE TAPE clause of the I-O-CONTROL paragraph in the COBOL program, even if a different ifn is used for each file in the tape. For example:

```
MULTIPLE FILE TAPE FILEA POSITION 1, FILEB POSITION 2,
                   FILEC  POSITION 12.
```

This clause is not essential (except when the same ifn is used for all files) and any values specified in the clause will be overridden by the JCL values described above. For more details of this clause see the *COBOL 85 Reference Manual*.

9.9 File Concatenation

File concatenation should be distinguished from the concepts of multi-volume files, multi logical unit files and multiple file tape volumes that have been discussed above.

Several UFAS or ANSI sequential files or queued subfiles may be accessed in sequence by means of file concatenation; the program treats the files as if they were one logical sequential file. Boundaries between concatenated files are invisible to the COBOL program that processes such files. When the end of one file is reached, the system automatically switches to the next file. The CLOSE REEL and CLOSE UNIT statements work in a manner similar to multi-volume files. These statements cause the next volume to be opened. This volume may be the next volume in a multi-volume file or the first volume of the next concatenated file. The RERUN ON CHECKPOINT-FILE EVERY END OF REEL/UNIT statement also operates on end of volume.

File concatenation is performed by the specification of the respective ASSIGN JCL statements, in the required sequence, with the omission of the internal-file-name on all but the first ASSIGN statement. It is strongly recommended that the POOL JCL statement be used to allocate one or two devices for all the files to be concatenated. For example:

```
POOL 2*MT/T9;
ASSIGN ifn,MY.FILE1,DEVCLASS=MT/T9,MEDIA=A1,POOL,FIRST;
ASSIGN ,MY.FILE2,DEVCLASS=MT/T9,MEDIA=A2,POOL,NEXT;
ASSIGN ,MY.FILE3,DEVCLASS=MT/T9,MEDIA=A3,POOL,NEXT;
```

In the above example, the three tape files are regarded as a single sequential file, starting at MY.FILE1 and finishing at MY.FILE3. Note that the concatenated files must all have the RECSIZE parameter in accordance with the COBOL description.

One or more of the files to be concatenated may be a multi-volume file if required. Furthermore, file concatenation can be used in a multi logical unit file. One or more of the ASSIGN JCL statements for a multi logical unit file can use the file concatenation facility.

9.10 Organization

The selected file access system may be indicated in the ORGANIZATION IS clause of the FILE-CONTROL paragraph. The following values may be used in this clause:

- UFF (specifies UFAS): this is the default when no ORGANIZATION IS clause is specified, or when the ORGANIZATION IS clause is used without the phrase that indicates the file access system.
- QUEUED.
- ANSI.

UFAS is the primary file access system. It is used by IDS/II for data base management. It offers a wide range of facilities and provides considerable device independence.

However, this phrase in the ORGANISATION IS clause is not part of the ANS standard and is significant only in the following circumstances:

- The file is an input queued file from which one or several members are to be processed by use of dynamic member assignment (QUEUED).
- The file is an output ANSI tape file or an input ANSI tape file with no label (ANSI).

It is recommended that PREALLOC be used for all permanent output disk files. PREALLOC provides a centralized and visible method of specifying file attributes.

9.11 APPLY NO-SORTED-INDEX

The APPLY NO-SORTED-INDEX clause of the I-O-CONTROL paragraph can be used to speed up the creation of a UFAS indexed file with ALTERNATE KEYS by not sorting the alternate (secondary) key-indexes. The alternate indexes can be sorted after program execution by using the utility SORTIDX. See the *Data Management Utilities User's Guide* for details of SORTIDX.

Note that the APPLY NO-SORTED-INDEX clause is not part of the ANS standard.

9.12 NO PADDING

According to the ANSI Standard, the input records are checked to verify if their content is different from the default padding character. In order to speed up the READ statement, the APPLY NO-PADDING clause prevents from doing this check.

9.13 Error Handling

If as a result of an I/O operation, a system procedure returns to the COBOL program an abnormal return code that does not correspond to an AT END or INVALID KEY condition, the step is abnormally terminated and an exception report is printed in the JOR. Abnormal termination can be avoided if the COBOL program contains a USE AFTER ERROR PROCEDURE section in the Declaratives for the relevant file. This section can then diagnose the error using the FILE STATUS phrase associated with the file or the system return code obtained by calling the routine H_CBL_UGETG4. These subjects are discussed in the following paragraphs.

For the AT END and INVALID KEY conditions, the user must provide for their processing by specifying the AT END and INVALID KEY phrases in the I/O statement. If these phrases are not specified, a USE AFTER ERROR procedure must be associated with the file connector. If one of these conditions occurs in the absence of the corresponding phrase, it is the USE AFTER ERROR procedure that will be executed.

9.13.1 The FILE STATUS Clause

The following example shows the way in which a USE AFTER ERROR PROCEDURE section and a FILE STATUS phrase can be used:

```
FILE-CONTROL.
  SELECT FILEA ASSIGN TO FILEA
    FILE STATUS IS FS12...
.
.
WORKING-STORAGE SECTION.
01 FS12 PIC XX.
.
.
PROCEDURE DIVISION.
DECLARATIVES.
FILEA-ERROR SECTION.
  USE AFTER ERROR PROCEDURE ON FILEA.
P1.
  DISPLAY "STATUS = " FS12.
EX-IT.
  EXIT.
END DECLARATIVES.
MAIN SECTION.
DEBUT.
  OPEN INPUT FILEA.
.
.
```

If an I/O operation on FILEA results in an abnormal system return code being generated, control will be handed to paragraph P1. The FILE STATUS FS12 (specified in the SELECT clause for FILEA) is displayed. The EXIT then hands control back to the instruction following the I/O request. See the *COBOL 85 Reference Manual* for a full description of USE AFTER ERROR.

The FILE STATUS data item (two characters) is set by a COBOL run-time package procedure according to the return code obtained from the system Data Management procedures. The meanings of the values to which the FILE STATUS data item can be set are given in the *COBOL 85 Reference Manual*.

When a USE AFTER ERROR PROCEDURE section is invoked for the first time from a given point in the program, the following message is output in the JOR.

```
CBL18. IFN:ifn rc AT ADDRESS address [ILN=iln [XLN=xln]]
```

9.13.2 Return Code

The FILE STATUS facility is part of ANS standard COBOL and is described in the *COBOL 85 Reference Manual*. The information provided in the FILE STATUS data item is normally sufficient to diagnose most I/O errors. However, the full return code generated by Data Management can be obtained and analyzed by the COBOL program. This is done by calling the procedure H_CBL_UGETG4 in the COBOL run-time package. This facility is not part of the ANS standard and for this reason should be avoided whenever the use of the FILE STATUS phrase provides sufficient information. The return code can be translated in symbolic form by calling the procedure "H_STD_UEDTG4". The following example shows the use of H_CBL_UGETG4 and H_STD_UEDTG4:

```
WORKING-STORAGE SECTION.
77  RET-CODE-1    USAGE COMP-1.
77  RET-CODE-2.   USAGE COMP-1.
77  SYMBOLIC-RET  PIC X(30).
77  NOISE COMP-2  VALUE -1.
.
.
PROCEDURE DIVISION.
DECLARATIVES.
FILEA-ERROR SECTION.
    USE AFTER ERROR PROCEDURE ON FILEA.
P1.
    CALL "H_CBL_UGETG4" USING RET-CODE-1 RET-CODE-2
    IF RET-CODE-2 NOT = ZERO
       CALL "H_STD_UEDTG4" USING SYMBOLIC-RET NOISE
       DISPLAY SYMBOLIC-RET.
EX-IT.
    EXIT.
```

```
END DECLARATIVES.
MAIN SECTION.
DEBUT.
    OPEN INPUT FILEA.
.
.
```

The return code is a hexadecimal value. The significance of each return code value is given in the *Messages and Return Codes Directory*.

The symbolic value of a return code is of the form:

```
"RC=4B820909->DSASG 2,IFNNASG"
```

9.14 Restrictions on Certain File Organizations or Formats

Some features, though described in the *COBOL 85 Reference Manual*, are not available for certain file organizations. These are listed below.

Features Not Available with Certain File Organizations

Feature not Available	File Organization/Format
Physical units	UFAS disk sequential
RERUN EVERY END OF UNIT	UFAS sequential
REWRITE	SSF physical format

When such a feature is used for a file organization where it is not available, it generally results in a file status "30" returned to the program and a return code mnemonic FUNCNAV reported in the JOR.

9.15 Record Size

The maximum record size of a COBOL file is determined according to the following criteria:

1. For a report file:
   - If the RECORD CONTAINS clause is present, the specified record length is augmented by 8.
   - If the RECORD CONTAINS clause is not present, the record length is assumed to be 140.
2. For the other files:

   Unless one of the following conditions exists, the record length is taken to be the length specified in the RECORD CONTAINS clause if it is present, or, if not, the length of the largest record defined for the file. If one of the following conditions exists, this length is augmented by 8.

   - The internal-file-name in the SELECT clause is suffixed by -PRINTER or - SYSOUT and WITH ASA or WITH SARF is not specified.
   - WITH SSF is specified in the SELECT clause.
   - The LINAGE clause is specified in the File Description entry.
   - The file is referenced in a WRITE statement with the BEFORE/AFTER ADVANCING phrase.

Queued files may be allocated using the COMPACT option. For such files the length of the records on the media will be different from that of the record in memory, and this will depend on the contents of the record. It is therefore prohibited to use REWRITE statements on such files.

9.16 The ACTUAL KEY Phrase

By using the OUTPUT command of the SORT utility a sequential file of disk addresses can be produced. This contains, in sorted order, the disk addresses of the records input to the SORT utility. The address file can later be read into a COBOL program and can be used to read, in a sorted sequence, the records of the data file input to the SORT utility. That is, the data file is not actually sorted by the SORT utility, but it can be read in the sorted sequence by a COBOL program, using the address file produced by the SORT utility.

In order to read the data file in the sorted sequence the SELECT clause for the data file must contain an ORGANIZATION RELATIVE phrase and an ACTUAL KEY phrase (instead of RELATIVE KEY). The address file, on the other hand, must be read as a sequential file. Each record on the address file will contain a disk address in the first five bytes. As each record of the address file is read, the address must be moved to the 5-byte data item specified in the ACTUAL KEY phrase of the data file. The next READ statement executed on the data file will then input the next data record in the sorted sequence.

The ACTUAL KEY phrase can only be used if the LEVEL=NSTD parameter is specified in the CBL JCL statement when the program is compiled. Note that the ACTUAL KEY phrase is not part of the ANS Standard.

The SORT utility will only write an address file if one of the values ADDROUT, ADDATA or KEYADDR is specified in the OUTPUT command. In addition, if the ADDRFORM parameter is present in the SORT JCL statement, it must contain the value TTRDD (this is the default value) which specifies the format of the address.

9.17 Dynamic File Assignment

In addition to file assignment through the JCL commands, a file may be assigned in a COBOL program using either the SELECT entry, if this entry specifies litera-1, or the ASSIGN statement containing the TO FILE phrase. When a file is to be dynamically assigned with an ASSIGN statement, its SELECT entry must also specify literal-1.

```
SELECT F1 ASSIGN TO IFN1 "MYFILE1:D730:MS/B10".
...
ASSIGN F1 TO FILE "MYFILE2:D730:MS/B10"
...
OPEN INPUT F1
```

When F1 is opened:

1. if an ASSIGN F1 TO FILE statement has been already executed, F1 is dynamically assigned to the file specified in the last executed such ASSIGN statement,
2. if no ASSIGN F1 TO FILE statement has been already executed:
   a. if F1 (IFN1) is referenced in a JCL ASSIGN statement, F1 is (statically) assigned to the file specified in this JCL statement,
   b. if F1 (IFN1) is referenced in no JCL ASSIGN statement, F1 is dynamically assigned to the file specified in the SELECT entry.

Normally, the system allows a step to run only when all resources this step needs are available. Thus a step starts when all files specified in related JCL ASSIGN statements are available. This constraint does not apply for files that are dynamically assigned so when such a file is opened, it is possible that this file is used by a concurrent step. If sharing this file is not possible, the return code "BUSY" is got by the OPEN statement. This OPEN is unsuccessful and the File Status data item (if any) is set to "93" for a file not described with the QUEUED ORGANIZATION clause, or "94" for a file described with the QUEUED ORGANIZATION. When such a situation occurs, the user may wish to iteratively retry opening its file until it is released by the concurrent step. This is possible if a DECLARATIVE section exists for exception recovering. If File Status "93" is got, an OPEN statement for the concerned file must be executed again, if File Status "94" is got, the ASSIGN statement containing the TO MEMBER phrase must be re-executed before re-opening the file. The user must take care that no deadlock is produced.

9.18 Queued File Processing

When the QUEUED clause is specified in the ORGANIZATION clause of the SELECT entry of a COBOL file, it is possible to dynamically select and access any member of a queued file.

The phrase TO MEMBER of the ASSIGN statement allows you to specify which member of the file is to be processed. Such a statement must have been executed before the file is opened.

The queued file itself should have been assigned either statically by JCL or dynamically (see above) without specifying any member name.

The following example shows a program that displays the names of all members in a library:

```
FILE-CONTROL.
     SELECT F1 ASSIGN TO F1 "PSEUDO$RES"
            ORGANIZATION QUEUED SEQUENTIAL.
       .
       .
       .
WORKING-STORAGE SECTION.
01 MEMBER-NAME PIC X(30).
       .
       .
       .
PROCEDURE DIVISION.
DEBUT.
     ASSIGN F1 TO FILE "MYPROJ.TESLIB".
     MOVE SPACE TO MEMBER-NAME
     ASSIGN F1 TO MEMBER > MEMBER-NAME
     PERFORM UNTIL MEMBER-NAME = LOW-VALUE
            OPEN INPUT F1
            DISPLAY MEMBER-NAME
            CLOSE F1
            ASSIGN F1 TO MEMBER > MEMBER-NAME
            END-PERFORM.
```

10. Standard Record Formats

There are three standard record formats recognized by Data Management. These formats may be used in magnetic tape or disk files. They are:

- System Standard Format (SSF)
  In this format each record comprises an eight-byte header followed by normal data. The function of this header is to make the file or subfile device-independent: a file or subfile in system standard format may be routed from the disk or tape to any kind of I/O device. This format provides the Stream Reader, compilers, LIBMAINT utility and Output Writer with a standard method of handling their input and output data.
- Standard Access Record Format (SARF)
  In this format each record is composed exclusively of normal data without any special heading information. This is the format normally used in data files or subfiles that are passed between COBOL programs.
- American Standards Association Format (ASA)
  In this format each record consists of a one-byte header followed by normal data. The header may be thought of as containing a subset of the information held in an SSF header. ASA files however are not device independent: they may only contain data to be printed. ASA files should be used for compatibility with other computer systems. They should not normally be used for print files that are to be processed solely within the system. In order to use this format, the programmer must specify WITH ASA in the SELECT clause for the file. The programmer is responsible for the contents of the first character of the record, which contains the skip information. This format is the only one supported for the 0711 printer.

ASA is rarely used by the COBOL programmer and will not be discussed further in this chapter. The remainder of this chapter discusses SSF and SARF.

NOTE:

Whenever the word "files" is used in the remainder of this chapter it should be taken to mean "files or subfiles".

10.1 System Standard Format (SSF)

SSF records include an eight-byte header in addition to the normal data. The main components of this header are:

- Record type. This indicates whether the record is a control record or a normal data record. Control records are added to the file by the system or by the Report Writer (if used) to control the handling of the file and the production of page headings etc.
- Header type. If the record is a control record this specifies the type of control record.
- Truncation value. This specifies the number of space characters that have been truncated at the rightmost end of the record. Truncation (packing) occurs only in records that were created with a language type of COBOL or COBOLX.
- Line number. This contains the sequence number of the record within the file. It may be derived from the data cards used when the file was read into the system; it may also be generated by the NUMBER option of the LIBMAINT command MOVE or by the LIBMAINT command RENUMBER.
- Form control. This specifies the paper movement required when printing the record.

If the first record in a file is a control record with a header type 101 and WITH SARF is not specified in the SELECT clause for the file, then the file is handled by the system as if all records in the file are in SSF format. If the file does not have such a record at the beginning then it is handled as if all records are in SARF format. If the type 101 control record is present it contains an indication of the language type specified when the file was created (e.g. TYPE = COBOLX in the MOVE command of LIBMAINT). COBOL automatically outputs a type 101 control record if the file is implicitly or explicitly specified as SSF.

The following paragraphs discuss the relationship between SSF and the Stream Reader, the LIBMAINT utility, the COBOL compiler, COBOL programs and the Output Writer.

10.1.1 The Stream Reader, LIBMAINT, and the COBOL Compiler

An SSF file can be created from the contents of an input enclosure. If TYPE = COBOL or TYPE = DATASSF is specified in the $INPUT statement the Stream Reader will create a temporary subfile in the system file SYS.IN and the cards will be read into this subfile as a series of SSF records. This is known as a standard SYSIN subfile and it exists only for the duration of the job.

Any job step or utility may then read the standard SYSIN subfile. For example it can be read by the LIBMAINT utility and moved to a user library:

```
$JOB...
  LIBALLOC SL,(SSF.LIB,SIZE=2),MEMBERS = 13;
  LIBMAINT SL,LIB=SSF.LIB,COMFILE = *SSFENC;
$INPUT     SSFENC,TYPE = DATASSF;
  MOVE     COMFILE: SSFMEMB,TYPE = COBOLX;
  .
  .
$ENDINPUT;
$ENDJOB;
```

In this example, a resident source library SSF.LIB is set up by LIBALLOC with a size of two cylinders. The contents of the input enclosure is then moved from the standard SYSIN subfile to the library SSF.LIB by the MOVE command of LIBMAINT. A new member SSFMEMB is created in library SSF.LIB to contain the data. The TYPE=COBOLX parameter has a special effect on the format of the records in the library member. This is discussed in Chapter 1, *Input and Maintenance of Source Programs*.

A user program may read an SSF library member. Alternatively, a user program may read an input enclosure directly from the standard SYSIN subfile. However, this is normally done only when TYPE=DATA is specified in the $INPUT statement, or if there is no TYPE parameter. In this case, the records will be held in SARF format.

The EDIT and UPDATE commands of LIBMAINT may be used to alter the contents of SSF library members. With these commands the user may specify the lines of the library member to be altered by specifying the line numbers held in the SSF headers. The creation and updating of SSF library members is discussed in detail in Chapter 1 and will not be discussed further in the current chapter.

If the SSF library member contains a COBOL program, the COBOL compiler can process it. The compiler will check the line numbers in the SSF headers and report on any descending sequences in the member. The use of the compiler is covered in Chapter 2. If the SSF library member contains JCL it can be used in an INVOKE, EXECUTE or RUN JCL statement (see *JCL User's Guide*).

10.1.2 Reading SSF Files in COBOL Programs

Any file input to a COBOL program might be SSF or SARF, irrespective of the way in which it is described in the associated SELECT clause in the program that is reading the file. The presence or absence of a type 101 control record at the start of the file indicates the format of the file. Data Management checks for the existence of this record and processes the file accordingly.

A COBOL program can receive SSF records from Data Management either with or without the SSF header. If the WITH SARF phrase is included in the SELECT clause of the file, the complete SSF record including the header will be passed to the COBOL program. If the WITH SSF phrase is included in the SELECT clause, or if there is no WITH phrase, the SSF header will be stripped from the record before it is passed to the COBOL program and control records will not be passed to the COBOL program. When the WITH SSF phrase is used, the leading 8 bytes are stripped off whether the file is SSF or not. Note that the phrases WITH SSF and WITH SARF are not part of the ANS Standard and should be avoided unless they are essential.

The following example illustrates the use of an SSF input file on magnetic tape. In this example the WITH SARF phrase is used which will cause the SSF headers and control records to be passed to the COBOL program.

```
COBOL:
 SELECT INFILE ASSIGN TO F1 WITH SARF.

JCL:
 ASSIGN F1, SSF.FILE, DEVCLASS=MT/T9/D1600, MEDIA=TAPE1;
```

The next example shows the use of an SSF library member on disk. The SSF headers will be stripped from the records before they are passed to the COBOL program. In this example the WITH SSF phrase is redundant and serves as documentation only.

```
COBOL:
 SELECT INMEMBER ASSIGN TO F2 WITH SSF.

JCL:
 ASSIGN. F2,SSF.LIB,SUBFILE=SSFMEMB,
                    DEVCLASS=MS/M450,  MEDIA=DISK1;
```

The final example illustrates the use of an SSF input enclosure (which is a temporary subfile of the system file SYS.IN). No WITH phrase is used. However, the SYS.IN subfile will begin with a type 101 control record, so Data Management will treat it as an SSF file. As there is no WITH SARF phrase the headers will be stripped from the records before they are passed to the COBOL program.

```
COBOL:
   SELECT INCLOSE ASSIGN TO F3.

JCL:
   ASSIGN F3, *SSFENC;
   $INPUT SSFENC, TYPE = DATASSF;
   .
   .
   $ENDINPUT;
```

If the SSF header contains a non-zero truncation value, that is, when blanks have been truncated at the end of a record, the blanks are not restored when the record is read. The record length, in such cases, does not include the truncation value and only refers to the length of the record on the I/O medium. Truncation values are generated when, for example, the file has been created by the LIBMAINT utility with a language type of COBOL or COBOLX.

In fact, unless this input enclosure was needed in SSF form for another step of the same job, it would be better to hold the data in SARF form. In this case the $INPUT statement should not have a TYPE parameter (DATA would be assumed, indicating SARF format) but the SELECT clause would remain unchanged.

10.1.3 Writing SSF Files in COBOL Programs

An output file is in SSF format if the WITH SSF phrase is included in the SELECT clause of the file, or if the ADVANCING phrase is used in an associated WRITE statement, or if the FD contains the REPORTS clause or LINAGE clause or, unless otherwise specified, if the internal-file-name in the SELECT clause has a suffix -PRINTER or -SYSOUT. For example:

```
SELECT  OUTFILE  ASSIGN  TO  F1 WITH SSF.
SELECT  OUTFILE  ASSIGN  TO  F1-PRINTER.
SELECT  OUTFILE  ASSIGN  TO  F1-SYSOUT.
```

However, under certain circumstances, such a file will be output as "edited SYSOUT" instead of SSF. This is explained in Chapter 11, *Using Unit Record Files*.

Output SSF files are usually print or punch files or subfiles. The use of print and punch files is described in Chapter 11 and will not be discussed in the present chapter. Use of SSF files should be restricted, as far as possible, to queued files.

10.2 Standard Access Record Format (SARF)

SARF records have no special header but are composed exclusively of user data. This is the format normally used in data files that are passed between COBOL programs. However, the Stream Reader, the LIBMAINT utility, the compilers and the Output Writer may also handle SARF files. The following paragraphs discuss the use of SARF format.

10.2.1 The Stream Reader, LIBMAINT, and the COBOL Compiler

A SARF library member can be created from cards contained in an input enclosure. Normal practice should be to omit the TYPE parameter from the $INPUT statement (equivalent to TYPE=DATA). If this is done, the Stream Reader will create a temporary subfile in the system file SYS.IN and the cards will be read into this subfile as a series of SARF records. That is, a standard SYSIN subfile will be created for the duration of the job.

The standard SYSIN subfile may then be read by the LIBMAINT utility and may be moved to a user library. The following example illustrates this sequence:

```
$JOB...
  LIBALLOC     SL,(SARF.LIB,SIZE=2),MEMBERS=13;
  LIBMAINT     SL,LIB=SARF.LIB,COMFILE=*SARFENC;
$INPUT         SARFENC;
  MOVE         COMFILE:SARFMEMB,TYPE=DATA;
  .
  .
$ENDINPUT;
$ENDJOB;
```

In this example, a resident source library SARF.LIB is set up by LIBALLOC with a size of two cylinders. The contents of the input enclosure is then moved from the standard SYSIN subfile to the library SARF.LIB by the MOVE command of LIBMAINT. A new member SARFMEMB is created in library SARF.LIB to contain the data.

The EDIT and UPDATE commands of LIBMAINT cannot be used to alter the contents of SARF library members. However, if the SARF library member contains a COBOL program, the COBOL compiler can process it. The use of the compiler is covered in Chapter 2. If the SARF library member contains JCL it can be used in an INVOKE, EXECUTE or RUN JCL statement (see *JCL User's Guide*).

10.2.2 Reading SARF Files in COBOL Programs

As mentioned previously, a COBOL program may read any file in SSF or SARF format without specifying WITH SSF or WITH SARF.

The user must not, implicitly or explicitly, specify WITH SSF in the SELECT clause of SARF input files. Otherwise, the first eight characters of each record will not be passed to the user program and some complete records will not be passed to the user program. On the other hand, if neither WITH SSF nor WITH SARF is specified and if the first record on the file happens to look like a type 101 control record, Data Management will incorrectly assume that the file is in SSF format.

To read a SARF file successfully, either the WITH phrase should be omitted entirely or the WITH SARF phrase should be specified. Note that the phrases WITH SSF and WITH SARF are not part of the ANS standard and should be avoided unless they are essential.

10.2.3 Writing SARF Files in COBOL Programs

An output file may be in SARF format if the WITH SARF phrase is specified in the SELECT clause of the file or if the WITH phrase is omitted entirely and none of the options implying WITH SSF is used for the file (see above). However, under certain circumstances, such a file will be output as "edited SYSOUT" instead of SARF. This is explained in Chapter 11, *Using Unit Record Files*.

10.3 General Points Concerning SSF and SARF

10.3.1 The Output Writer

The Output Writer can print or punch any SSF or SARF file. The JCL statements SYSOUT and WRITER call it. The Output Writer is normally used to output print or punch files produced by user programs. However, it can also print or punch files that have not been specially formatted for output, such as a library member containing a COBOL program or a normal disk or tape file containing data.
The use of the Output Writer for print and punch files is discussed in Chapter 11.

10.3.2 Summary of Rules for the SELECT Clause

The table below summarizes the rules concerning the COBOL SELECT clause when using SARF or SSF files.

Summary of Rules for the SELECT Clause

Type of I/O Required	Type of File Used	SELECT Statement Options Note: WITH SSF/SARF are not ANS Standard
Input with header and control records removed	SSF	WITH SSF (this is the default if the input file is SSF, and should be omitted).
Input with header and control records intact	SSF	WITH SARF (must be specified) Note that the record description must have an eight byte FILLER at the start.
Output	SSF or Edited SYSOUT	WITH SSF, or WRITE with the ADVANCING phrase, or FD containing the REPORT clause or LINAGE clause or –PRINTER suffix or SYSOUT suffix on internal -file-name (one of these must be specified).
INPUT	SARF	WITH SARF (this is the default if the input file is SARF, and should be omitted). WITH SSF must not be used for SARF files.
Output	SARF or Edited SYSOUT	WITH SARF (default for all output files and should be omitted).

IMPORTANT:
When opening in output mode members of libraries, recall that the size of a library record is 264. In order to avoid RECERR, describe a record of 256 or 264 depending on the SSF prefix being present or not.

11. Using Unit Record Files

This chapter describes the way in which the following unit record files are used:

- Print files.
- Punched card files.
- ACCEPT, DISPLAY and STOP literal "files" (strictly speaking, from a COBOL point of view, these are not files because they have no FD).
- Diskette files.

11.1 Printing

Printing can be done in the following ways:

- Data can be stored in a SYSOUT file for printing later by the Output Writer.
- Data can be sent direct to the printer.

Print data should normally be output to a SYSOUT file. The direct use of printers slows down program execution and reduces the throughput of the printer.

See "*The Report Writer*", Chapter 12, for information concerning printed reports produced using the Report Writer facility.

11.1.1 Using SYSOUT Files for Printing

The following types of SYSOUT file can be used to store data to be printed:

- Standard SYSOUT subfile.
- Permanent SYSOUT file.

The standard SYSOUT file (SYS.OUT) is a system file. The SYSOUT file is created at system generation and is located on a resident disk. For each step, one or more subfiles is assigned for each unit record output file defined in the step. During execution of each step, data to be printed or punched is sent to subfiles of the standard SYSOUT file. No ASSIGN JCL statement need be used for a standard SYSOUT subfile. Standard SYSOUT subfiles exist until the data in them has been printed or punched. When output processing is finished, the subfiles are automatically deleted.

A permanent SYSOUT file is a sequential disk file or source library member which is not automatically deleted after Output Writer activity, or a permanent magnetic tape file (useful for large volumes of output). The user must assign a permanent SYSOUT file.

SYSOUT files are normally written in a format known as "edited SYSOUT". This is done automatically if certain conditions, described below, are met. This has the following effect on output data:

- Records are formatted for the output device.
- The page is formatted (page headers, numbers etc.).
- Trailing blanks are suppressed.

An edited SYSOUT file cannot be handled as a normal SSF, SARF or ASA file.

If the record size of the SYSOUT file is less than 600 bytes (specified when the file is allocated using the PREALLOC JCL statement or later in the DEFINE JCL statement) it will be written as an SSF, SARF or ASA file. If the record size is greater than or equal to 600 bytes the file will be in edited SYSOUT format. SSF, SARF or ASA files that are to be printed will be edited subsequently by the Output Writer. Note that the use of a record size of 600 does not imply storage inefficiency, since the RECFORM will be VB (variable). However, editing of SSF, SARF or ASA files during printing, rather than when the file is written, is relatively inefficient and should be avoided. See the *JCL User's Guide* for more information about SYSOUT files and the Output Writer.

There are certain situations in which a SYSOUT file should not be in edited SYSOUT format. If the SYSOUT file is to be processed before printing (e.g., by another COBOL program or by the LIBMAINT utility), it should not be written in edited SYSOUT format. Also, SYSOUT files produced by the COBOL Report Writer using the report selection facility should not be written in edited SYSOUT format. Note that a standard SYSOUT file is always an edited SYSOUT file.

The rules for writing SSF, SARF and ASA files are given in Chapter 10, Standard Record Formats. The method of producing permanent and standard SYSOUT files with or without edited SYSOUT format is summarized in the table below.

Methods of Producing SYSOUT Print Files

JCL		COBOL	TYPE OF SYSOUT FILE CREATED
ASSIGN	SYSOUT	-SYSOUT	
No No	Yes Optional	Optional Yes	A standard SYSOUT file is assigned by the system. The file is written in edited SYSOUT format. The file is printed. WHEN=DEFER in the SYSOUT JCL statement will be ignored. This parameter is used only with permanent SYSOUT files.
No	No	No	The step is abnormally terminated when the file is opened because no implicit or explicit file assignment has be made (RC=IFNNASG).
Yes	Yes	Optional	A permanent SYSOUT file is written. The file will be in edited SYSOUT format if the record size is at least 600 bytes. The file is printed by the output writer unless the WHEN=DEFER parameter is specified in the SYSOUT JCL statement.
Yes	No	Yes	A permanent SYSOUT file is written. The file will be in edited SYSOUT format if the record size is at least 600 bytes. The Output Writer does not print the file. A WRITER JCL statement must be given to print the file.
Yes	No	No	A permanent SYSOUT file is written. The file will not be in edited SYSOUT format, irrespective of the record size. The Output Writer does not print the file. A WRITER JCL statement must be given to print file.

The following notes explain the headings used in this table.

ASSIGN	is an ASSIGN JCL statement for a permanent SYSOUT file to be included in the JCL (YES or NO)?
SYSOUT	is a SYSOUT JCL statement for the SYSOUT file to be included in the JCL (YES, NO or OPTIONAL)?
-SYSOUT	is the -SYSOUT suffix to be used after the internal-file-name in the COBOL SELECT clause (YES, NO or OPTIONAL)?

A standard SYSOUT subfile is automatically assigned by the system when the user does not explicitly assign a SYSOUT file and when one or both of the following conditions apply:

- The SYSOUT JCL statement is used.
- The -SYSOUT suffix in the COBOL SELECT clause is used.

If neither the SYSOUT JCL statement nor -SYSOUT is specified, the user must assign a permanent SYSOUT file using the ASSIGN JCL statement; otherwise, the step is abnormally terminated.

Standard SYSOUT files are always written in edited SYSOUT format. This is also the case if SYSOUT or -SYSOUT is used and the SYSOUT file has been pre-allocated with a record size greater than or equal to 600 bytes. In all other cases the SYSOUT file will not be in edited SYSOUT format.

Standard SYSOUT files are always printed automatically by the Output Writer. They cannot be held for later printing by using the WHEN=DEFER parameter of the SYSOUT JCL statement. Permanent files are printed automatically only if there is a SYSOUT JCL statement in the job step JCL and if this statement does not contain the WHEN = DEFER parameter. In all other cases the permanent SYSOUT file should be printed in a separate job step by using the WRITER JCL statement.

All the SYSOUT files written according to the rules in the table "Methods of Producing SYSOUT Print Files" will have an SSF record format or an edited SYSOUT format. SSF format includes an eight-byte header in each record which enables form control information to be stored for each print line. As a result, WRITE ADVANCING options can be used when writing these files. This is also true for edited SYSOUT files.

SYSOUT files can also be written in SARF format, if the SSF phrase is neither specified nor implied, or simply by including the WITH SARF phrase in the COBOL SELECT clause. However, the use of SARF files is not recommended because WRITE ADVANCING options cannot be used when printing these files.

11.1.2 Printing Directly

When the printer is used directly, an ASSIGN JCL statement must be present at execution time that links the internal-file-name used for the printer to the output device. For example:

```
COBOL:
  SELECT PRINTOUT ASSIGN TO LISTING-PRINTER.

JCL:
  ASSIGN LISTING, DEVCLASS = PR, MEDIA = I20001;
  DEFINE LISTING, MARGIN = 10;
```

The use of the DEFINE statement is optional. See the *JCL Reference Manual* for details of the relevant DEFINE parameters.

11.1.3 Form Control

A "vertical format tape" is a punched tape loop often used in printers to control vertical paper movement. Since Series 60 printers do not use a vertical format tape, a software simulated vertical format unit (VFU) controls vertical paper movement. This VFU works in the same way as a standard 12-channel vertical format tape, with a limitation of 20 stop levels per form, shared among the 12 channels.

A COBOL program can use the VFU to control vertical paper movement by specifying a mnemonic-name in the ADVANCING clause of the WRITE statement. This mnemonic-name must be specified in the CHANNEL-p IS mnemonic-name clause of the SPECIAL-NAMES paragraph. CHANNEL-p indicates the channel of the VFU that is to control vertical paper movement for the current WRITE operation.

VFUs are stored in a system file called SYS.URCINIT. The user can add new VFUs to this file or modify existing ones using the utility URINIT. This process is described in the *Unit Record Devices User's Guide*. Also stored in SYS.URCINIT are the form height, margin, head of form, full form 1 and printing density. All this information is associated with a form number. This form number can be specified in the MEDIA parameter in the ASSIGN, OUTVAL, SYSOUT and WRITER JCL statements in order to ensure that the correct VFU, form height etc. are used when the file is printed. See the *JCL Reference Manual* for details of the MEDIA parameter in ASSIGN, OUTVAL, SYSOUT and WRITER.

The VFU, form height, margin, head of form, full form 1 and printing density stored in SYS.URCINIT can be overridden at execution time by parameters specified in a DEFINE JCL statement. See the *JCL Reference Manual* for details. Note that all form control parameters specified in SYS.URCINIT and in the DEFINE statement for a given file are ignored at execution time if either the LINAGE clause or the Report Writer is used with that file.

11.1.4 The LINAGE Clause

The LINAGE Clause can be used in an FD entry to describe the vertical format of a logical page as follows:

- number of lines of text on the page (LINAGE),
- line number at which the footing zone begins (FOOTING),
- number of lines in the top margin (TOP),
- number of lines in the bottom margin (BOTTOM).

A WRITE statement with an AT END-OF-PAGE phrase can then be used on such a file. When the page being printed reaches the footing zone, the imperative statement following the AT END-OF-PAGE phrase is obeyed. This enables the program to print totals, summaries, banners etc. before the next page is started. At the end of each page the program can change the values of LINAGE, FOOTING, TOP and BOTTOM. Thus, the format of the page can change dynamically during program execution.

LINAGE-COUNTER is a field automatically defined by the compiler whenever the LINAGE clause is used in an FD statement. LINAGE-COUNTER contains the line number at which the printer is positioned within the current page. Therefore, the programmer need not keep a record of the current line number. The value of LINAGE-COUNTER can be referenced in the COBOL program (qualified if necessary by the file-name) but cannot be modified.

In the following paragraphs note that the END-OF-PAGE imperative is executed after the associated WRITE statement and the LINAGE-COUNTER may thus point to the next logical page (instead of to the current footing area) when the imperative is obeyed.

When the compiler encounters an ADVANCING nn LINES, it first calculates the sum of LINAGE-COUNTER and nn. Subsequent actions depend on the value of this sum, as follows:

Situation 1 — If the advance would be within the body of the current logical page, (i.e., the value is not greater than the established LINAGE value):

- The WRITE is done either before or after advancing nn lines, as specified in the program
- LINAGE-COUNTER is increased by nn.
- If FOOTING was specified and the advance would be within the footing area (i.e., greater than or equal to the established footing value), the END-OF-PAGE imperative is obeyed, if one was specified.

Situation 2

If the advance would go beyond the body of the current logical page, (i.e., the value is greater than the established LINAGE clause):

- A new value is set-up for LINESAT TOP, if the COBOL program has changed this value.
- The WRITE is done either before or after (as specified in the program) the device is positioned at the first line of the next logical page.
- LINAGE-COUNTER is set to 1.
- New values are set-up for LINAGE, FOOTING and LINES AT BOTTOM, if the COBOL program has changed these values.
- The END-OF-PAGE imperative is obeyed, if one was specified.

NOTE:

The CHANNEL-p IS mnemonic-name clause of the SPECIAL-NAMES paragraph cannot be associated with a file for which the LINAGE clause has been specified. Also, any form control information specified in the JCL statements for such files is ignored when the files are written. See "*Form Control*", above.

11.2 Reading Cards

Cards can be read in the following ways:

- from a standard SYSIN subfile containing a series of card images which have been spooled by the Input Reader;
- directly from the card reader.

Cards should normally be read from a SYSIN subfile. The use of the card reader directly, slows down program execution and reduces the throughput of the card reader.

11.2.1 Using SYSIN Subfiles for Cards

The standard SYSIN file (SYS.IN) is a system file. It is created at system generation and is located on a resident disk. Whenever an input enclosure is defined in a job, the Stream Reader creates a temporary subfile in the standard SYSIN file. This subfile is known as a standard SYSIN subfile. Card images are then read into this subfile. However, the subfile exists only for the duration of the job.

For each input enclosure to be read by a COBOL program there must be a SELECT clause and an associated file description. There must also be an ASSIGN JCL statement for each input enclosure to be read. The ASSIGN statement specifies the internal-file-name contained in the COBOL SELECT clause and the input-enclosure-name used in the $INPUT statement. The input-enclosure-name must be prefixed by an asterisk in the ASSIGN statement. The following example illustrates the necessary COBOL and JCL:

```
COBOL:
  SELECT CARD ASSIGN TO CARDFILE.

JCL:
  ASSIGN CARDFILE, *INDECK;
  $INPUT INDECK;
  .
  .
  $ENDINPUT;
```

If it is necessary to retain a card file on disk, this can be done using the utilities LIBMAINT or CREATE. The file may then be read in subsequent jobs as a normal sequential file or subfile.

The user can choose to read cards from the standard SYSIN file or from a permanent sequential file or even directly from the card reader, simply by changing the JCL at execution time. The COBOL program remains unchanged. The suffixes -CARD-READER and -SYSIN on the internal-file-names of SELECT clauses are for documentation only. The compiler ignores them.

11.2.2 Reading Cards Directly

To read cards directly from the card reader, there must be an ASSIGN statement in the execution JCL that links the internal-file-name used for the card reader to the input device. For example:

```
COBOL:
  SELECT CARD ASSIGN TO CARDFILE.

JCL:
  ASSIGN CARDFILE, DEVCLASS = CD/R, MEDIA = INDECK;
  DEFINE CARDFILE, OFFSET;

CONSOLE MESSAGE:
  * hh.mm MOUNT INDECK FOR ron
```

where:

hh.mm is the current time in hours and minutes,
ron is the run occurrence number.

The use of the DEFINE statement is optional. See the *JCL Reference Manual* for details of the relevant DEFINE parameters.

The name specified in the MEDIA parameter is displayed on the operator's console at step initiation. This name should also be written on the card deck so that the operator can see clearly which card deck is to be used. The card deck must not be part of a job stream. It must be a separate deck and the last card must be a $EOS statement followed by at least one blank card. The card deck should be mounted in the card reader and the card reader should be switched to "ready".

11.3 Punching Cards

Punched cards can be output in the following ways:

- to a SYSOUT file;
- directly to the card punch.

Cards should normally be output to a SYSOUT file. Direct use of the card punch slows down program execution and reduces the throughput of the card punch. In either case, serious consideration should be given to use of a more compact and less fragile storage medium.

11.3.1 Using SYSOUT Files for Cards

Both standard SYSOUT and permanent SYSOUT files may be used to store data to be punched. They have the same characteristics as the printer SYSOUT files described in the table "Methods of Producing SYSOUT Print Files".

The JCL and COBOL are the same as that shown in the table "Methods of Producing SYSOUT Print Files" except that the SYSOUT JCL statement is mandatory when the ASSIGN JCL statement is not used (otherwise the file will be printed instead of punched).

The SYSOUT JCL statement, when used, must specify a card punch device class. For example:

```
SYSOUT PUNCHER, DEVCLASS = CD/P, MEDIA = PNCOUT;
```

The WRITER JCL statement, when used, must also specify a card-punch device-class. For example:

```
WRITER C.PUNCHER, DEVCLASS = CD/P, MEDIA = PNCOUT;
```

As shown in the table "Methods of Producing SYSOUT Print Files", SYSOUT files may be produced in edited SYSOUT format. That is, the files are edited as if they are going to be printed. When the files are actually punched by the Output Writer they are again edited into a format suitable for the card punch. So editing is performed twice. This will not be a problem if a small number of cards are to be output. However, should large card decks be output, it might be advisable to avoid it.

Note that standard SYSOUT subfiles are always written in edited SYSOUT format. It is therefore better to use permanent SYSOUT files for all card punch output. In fact, if a SYSOUT file is not to be printed it can be output as a normal permanent sequential file and the table "Methods of Producing SYSOUT Print Files" can be simplified as shown in the table "Methods of Producing SYSOUT Punch Files".

Methods of Producing SYSOUT Punch Files

JCL		COBOL	TYPE OF SYSOUT FILE CREATED (RECORD SIZE<60 BYTES)
ASSIGN	SYSOUT	-SYSOUT	
Yes	Yes	No	A permanent SYSOUT file is created. The file will not be in edited SYSOUT format because the record size is less than 600 bytes. The file is punched by the Output Writer unless the WHEN=DEFER parameter is specified in the SYSOUT JCL statement.
Yes	No	No	A permanent SYSOUT file is created. The file will not be in edited SYSOUT format because the record size is less than 600 bytes. The Output Writer does not punch the file. A Writer JCL statement must be used to punch the file.

11.3.2 Punching Cards Directly

To punch cards directly on the card punch, there must be an ASSIGN JCL statement in the execution JCL that links the internal-file-name used for the card punch to the output device. For example:

```
COBOL:
  SELECT CARD ASSIGN TO CARDFILE.

JCL:
  ASSIGN CARDFILE, DEVCLASS = CD/P, MEDIA = OUTDECK;
  DEFINE CARDFILE,OFFSET;

CONSOLE MESSAGE:
  * hh.mm MOUNT OUTDECK FOR ron
```

where:

hh.mm is the current time in hours and minutes.

ron is the run occurrence number.

The use of the DEFINE statement is optional. See the *JCL Reference Manual* for details of the relevant $DEFINE parameters.

The name specified in the MEDIA parameter is displayed on the operator's console at step initiation. A deck of blank cards should be mounted in the card punch and the card punch should be switched to "ready".

11.4 ACCEPT, DISPLAY and STOP Literal

The COBOL ACCEPT and DISPLAY statements are used to input and output small volumes of data. The STOP literal statement is used to suspend execution of the program until the operator enters a value that enables the program to continue. The use of these statements is described in the following paragraphs.

11.4.1 The ACCEPT Statement

The format of the ACCEPT statement to be discussed is as follows:

```
ACCEPT identifier [FROM mnemonic-name]
```

The standard options DATE, DAY, DAY-OF-WEEK, TIME and MESSAGE COUNT of the ACCEPT statement are not used for unit record I/O and will not be discussed here (see the *COBOL 85 Reference Manual*). The options SYSIN, CONSOLE, TERMINAL and ALTERNATE-CONSOLE possibly suffixed by -0, -1, -2 or -X may be used for unit record I/O but they are not part of the ANS standard. It is recommended that the standard option "FROM mnemonic-name" be used instead of SYSIN, CONSOLE, TERMINAL or ALTERNATE-CONSOLE.

Mnemonic-name is defined in the SPECIAL-NAMES paragraph of the Environment Division as follows:

```
{ SYSIN                 }
{ SYSIN-q               }
{ CONSOLE               }
{ CONSOLE-q             } IS mnemonic-name
{ ALTERNATE-CONSOLE     }
{ ALTERNATE-CONSOLE-q   }
{ TERMINAL              }
{ TERMINAL-q            }
```

where -q may be -0, -1, -2 or -X.

The above format of the ACCEPT statement and SPECIAL-NAMES paragraph can be used to input data from the operator's console or from any sequential-file in SSF or SARF format. If the "FROM mnemonic-name" option is not used SYSIN is normally assumed to be the input device. If no suffix is specified for the implicit or explicit device, -1 for SYSIN and -2 for other devices are assumed when the parameter COBOL85 is specified in the CBL JCL statement, -0 is assumed when COBOL74 is specified. These defaults can be overridden by using the ACCEPT IS, SYSIN IS, ACCEPT CONSOLE IS, ACCEPT ALTERNATE-CONSOLE IS and ACCEPT TERMINAL IS in the Default Section (not part of the ANS standard).

If mnemonic-name specifies SYSIN or SYSIN-q, a special ASSIGN JCL statement must be used when the program is executed. This statement assigns the internal-file-name "H_RD" to the sequential input file. The input file may be a standard SYSIN subfile or a user file. If a standard SYSIN subfile is being read the input enclosure name must be specified in the ASSIGN statement. For example:

```
ASSIGN H_RD, *INCARDS;
```

If a user file is being read the file name must be specified in the ASSIGN statement. For example:

```
ASSIGN H_RD, INFILE;
```

In this example INFILE is a cataloged sequential file.

When data is being accepted with a hardware name SYSIN, as many records as necessary are read to get an amount of data of the so-called required size (see *COBOL 85 Reference Manual*); this is the standard method. However, if the first character in the first record of the SYSIN file is an ampersand (&) and the other characters (if any) in that record are spaces, the "console input method" is used (see below). That is, input continues until a record not ending with ampersand is read. This feature is useful when the number of cards to be read by a single ACCEPT statement is variable. If the ampersand is used, it is not necessary to pad the input with blank cards.

If mnemonic-name specifies CONSOLE or CONSOLE-q, no ASSIGN JCL statement is needed. The following message will be displayed on the operator's main console when the program is executed:

```
nn/hh:mm ron progid ACCEPT WAITING
```

where:

nn	is a message number that the operator must enter when replying to this message.
hh:mm	is the time at which the message was displayed.
ron	is the run-occurrence-number.
progid	is the program-id specified in the COBOL program.

The operator must then enter the message number, one space and then up to 64 characters of input data. If more than 64 characters of input data are to be input, each group of 64 characters must be terminated with an ampersand (&). An "ACCEPT WAITING CONTINUED" message will then be displayed and the input can be continued. This feature is useful on an interactive terminal. If it is used, the full 64 characters do not have to be entered on every line of input.

For example, the following pair of entries is equivalent to entering one line comprising XYZ, 61 blanks and a carriage return:

```
& (CR)
XYZ (CR)
```

Each ACCEPT dialog that occurs on the console will be echoed in the JOR prefixed by a report code "CBL13".

If mnemonic-name specifies ALTERNATE-CONSOLE or ALTERNATE-CONSOLE-q, data will be accepted from the alternate operator's console specified in the CONSOLE JCL statement (see the *JCL Reference Manual*). If no CONSOLE JCL statement is used, data will be accepted from the console which submitted the program. If the submitting console is no longer logged, execution stops with the following message in the JOR:

```
EX03. UNEXPECTED RETURN CODE OPRTR 14 CNSLUNKN
```

The format of the console dialog is the same as that on the main console.

If mnemonic-name specifies TERMINAL or TERMINAL-q and the load module is interactively executed from a terminal under the Interactive Operation Facility, the prompt "I:" will be output on the terminal and the terminal operator will then be able to enter data. If the load module is not executed from the terminal, but is executed as a batch job, the ACCEPT will behave as if mnemonic-name specifies ALTERNATE-CONSOLE.

If a break is entered on the console, the data entered is lost, and the ACCEPT WAITING message (or I: prompt for TERMINAL) is output again.

Note that if the suffix of the referenced device is -0, all data entered on a console or terminal will be stored in the user program as if the receiving item had a DISPLAY usage, even if the declared usage of the receiving fields is not DISPLAY. That is, no data conversion is performed. If the suffix is -X, the entered data is considered as the hexadecimal representation of the internal bit-coded value of the receiving item. If the suffix is -1 or -2, the entered data may be converted according to the rules specified in the *COBOL 85 Reference Manual* (see the legible equivalent).

In the examples given in the table "*Examples of Data Retrieved via an ACCEPT Statement*", the following SPECIAL-NAMES paragraph is assumed:

```
SPECIAL-NAMES. TERMINAL-0  IS  TE-0
               TERMINAL-1  IS  TE-1
               TERMINAL-2  IS  TE-2
               TERMINAL-X  IS  TE-X
               SYSIN-0     IS  SY-0
               SYSIN-1     IS  SY-1
               SYSIN-2     IS  SY-2
               SYSIN-X     IS  SY-X.
```

The results are identical if TERMINAL is replaced by CONSOLE or ALTERNATE-CONSOLE:

- when the column entitled 'FROM' contains more than one name, the results are identical for all the listed names; when the column contains 'not-X', the results are identical in all cases except for suffix -X; when the column contains 'any', the results are identical in every cases- in the column entitled 'entered data', each line corresponds to one separate record
- all blank characters are represented by lower case 'b', non printable characters are represented by '.'

Examples of Data Received via an ACCEPT Statement (1/4)

Receiving data item desc.	FROM	Entered Data	Contents of the receiving data item after execution of ACCEPT (in Char)	(in Hexadecimal)	Note
PIC X (8)	not-X	ABCDEFGH	ABCDEFGH	C1C2C3C4C5C6C7C8	
PIC X (8)	not-X	ABCDEFGHbb	ABCDEFGH	C1C2C3C4C5C6C7C8	(1)
PIC X (8)	SY-0 TE-0	ABCDEFGHIJK	ABCDEFGH	C1C2C3C4C5C6C7C8	(2)
PIC X (8)	SY-1 TE-1 SY-2 TE-2	ABCDEFGHIJK			(3)
PIC X (8)	SY-0 SY-1 SY-2	ABC DEF GH	ABCDEFGH	C1C2C3C4C5C6C7C8	(4)
PIC X (8)	SY-X	CIC2C3 C4C5C6 C7C8	ABCDEFGH	C1C2C3C4C5C6C7C8	(4)
PIC X (8)	SY-0 SY-1 SY-2	ABC bbb bb	ABCbbbbb	C1C2C34040404040	(5)
PIC X (8)	SY-0 SY-1 SY-2	ABC End-of-file	ABCbbbbb	C1C2C34040404040	(6) (7)
PIC X (8)	SY-X	C1C2C3 end-of-file	ABC…..	C1C2C30000000000	(6) (8)
PIC X (8)	TE-0 TE-1 TE-2	ABC	ABCbbbbb	C1C2C34040404040	(7)
PIC X (8)	TE-X	C1C2C3	ABC…..	C1C2C30000000000	(8)
PIC X (8)	SY-0 SY-1 SY-2	& ABC	ABCbbbbb	C1C2C34040404040	(9)
PIC X (8)	TE-0 TE-1 TE-2	ABC	ABCbbbbb	C1C2C34040404040	(9)
PIC X (8)	SY-0 SY-1 SY-2	& ABC& DEF	ABCDEFbb	C1C2C3C4C5C64040	(10)
PIC X (8)	TE-0 TE-1 TE-2	ABC& DEF	ABCDEFbb	C1C2C3C4C5C64040	(10)

Examples of Data Received via an ACCEPT Statement (2/4)

Receiving data item desc.	FROM	Entered Data	Contents data item ACCEPT (in Char)	of the receiving after execution of (in Hexadecimal)	Note
PIC S99V99 TRAILING	SY-0 TE-0	123D	12.34 (123D)	F1F2F3C4	(11)
PIC S99V99 LEADING	SY-0 TE-0	}012	-0.12(}012)	D0F0F1F2	(11)
PIC S99V99 TRAILING	SY-1 TE-1	+1234	12.34 (123D)	F1F2F3C4	(12) (13)
PIC S99V99 LEADING	SY-1 TE-1	-0012	-0.12 (}012)	D0F0F1F2	(14)
PIC S99V99 LEADING SEPARATE	SY-2 TE-2	12.34	12.34 +1234)	4EF1F2F3F4	(15)
PIC S99V99 TRAILING	SY-2	Bbbb –12E-2	-0.12 (001K)	F0F0F1D2	(15) (16)
PIC 99PP	SY-0 TE-0 SY-1 TE-1	45	4500 (45)	F4F5	
PIC 99PP	SY-2 TE-2	4500	4500 (45)	F4F5	(15)
PIC 99PP	SY-0 TE-0 SY-1 TE-1	67	0.0067 (67)	F6F7	
PIC 99PP	SY-2 TE-2	+.0067	0.0067 (67)	F6F7	(15)
PIC S99V99 TRAILING SEPARATE	SY-0 TE-0 SY-1 TE-1	1234+	12.34 (1234+)	F1F2F3F44E	(12) (13) (17)
PIC S99V99 TRAILING SEPARATE	SY-0 TE-0 SY-1 TE-1	0012-	-0.12 (0012-)	F0F0F1F260	(14) (17)

Examples of Data Received via an ACCEPT Statement (3/4)

Receiving data item desc.	FROM	Entered Data	Contents data item ACCEPT (in Char)	of the receiving after execution of (in Hexadecimal)	Note
BINARY PIC 999	SY-1 TE-1	012	12	000C	(14)
BINARY PIC 999	SY-2 TE-2	12	12	000C	(15)
BINARY PIC 999	SY-X TE-X	000C	12	000C	
COMP-1	SY-1 TE-1	+00123	123	007B	(12) (14)
COMP-1	SY-2 TE-2	123	123	007B	(15)
COMP-1	SY-X TE-X	007B	123	007B	
PIC 1 (8)	not-X	11010010	11010010	F1F1F0F1F0F0F1F0	
PIC 1 (8)	SY-0 SY-1 SY-2	110 100 10	11010010	F1F1F0F1F0F0F1F0	(4)
PIC 1 (8)	TE-0 TE-1 TE-2	110& 100& 10	11010010	F1F1F0F1F0F0F1F0	(10)
PIC 1 (8)	SY-0 SY-1 SY-2	1101 end-of-file	11010000	F1F1F0F1F0F0F0F0	(6) (18)
PIC 1 (8)	TE-0 TE-1 TE-2	1101	11010000	F1F1F0F1F0F0F0F0	(18)
PIC 1 (4)	SY-X TE-X	F1F1F0F1	1101	F1F1F0F1	

Examples of Data Received via an ACCEPT Statement (4/4)

Receiving data item desc.	FROM	Entered Data	Contents data item ACCEPT (Char)	of the receiving after execution of (Hexadecimal)	Note
BIT PIC 1 (8)	SY-0 TE-0	K	11010010 (K)	D2	
BIT PIC 1 (8)	SY-1 TE-1 SY-2 TE-2	11010010	11010010 (K)	D2	
BIT PIC 1 (8)	SY-1 SY-2	110 100 10	11010010 (K)	D2	(4)
BIT PIC 1 (8)	SY-1 SY-2	& 1101	11010000 ({)	D0	(18)
BIT PIC 1 (8)	TE-1 TE-2	1101	11010000 ({)	D0	(18)
BIT PIC 1 (8)	SY-X TE-X	D2	11010010 (K)	D2	
BIT PIC 1 (16)	SY-X TE-X	00	ZEROS	0000	(8)
POINTER	Any	FFFFFFFF	NULL	FFFFFFFF	
POINTER	Any	381C0A14	….	381C0A14	
POINTER	Any	381C	….	381C0000	(18)

Notes for the table "Examples of Data Retrieved via an ACCEPT Statement":

1. blank characters in excess are allowed
2. characters in excess (IJK) are ignored (lost)
3. entered data is longer than the required size, the program is in error
4. 3 records was necessary to get the 8-character required size for the receiving data item (SYSIN standard method)
5. rightmost blanks must be provided (SYSIN standard method)
6. padding at end of file (SYSIN standard method)
7. blank padding when suffix is -0, -1 or -2
8. zero padding when suffix is -X
9. rightmost blanks need not be provided (TERMINAL or SYSIN console method)
10. use of the continuation character '&' (TERMINAL or SYSIN console method)
11. no conversion with suffix -0
12. + sign must be provided
13. no actual decimal point
14. leftmost zeros must be provided
15. entered data is a COBOL numeric literal when the receiving item is numeric and suffix is -2
16. records are read until a non blank character is got when suffix is -2 and receiving data item is numeric
17. sign character is at end of entered data only when receiving data item has:

    ```
    SIGN TRAILING SEPARATE
    ```

    and suffix is -0 or -1
18. rightmost zeros need not be provided (TERMINAL or SYSIN console method)

11.4.2 The DISPLAY Statement

The format of the DISPLAY statement is as follows:

```
                              { identifier-1 } [, identifier-2 ]
DISPLAY [ WITH CONVERSION ] { literal-1    } [, literal-2    ] ...
        [ UPON mnemonic-name ]
```

The options SYSOUT, CONSOLE, TERMINAL and ALTERNATE-CONSOLE possibly suffixed by -0, -1, -2 or -X may be used for unit record I/O but they are not part of the ANS standard. It is recommended that the standard option "UPON mnemonic-name" be used instead of SYSOUT, CONSOLE, TERMINAL or ALTERNATE-CONSOLE.

Mnemonic-name can be defined in the SPECIAL-NAMES paragraph of the Environment Division as follows:

```
{ SYSIN                }
{ SYSIN-q              }
{ CONSOLE              }
{ CONSOLE-q            } IS mnemonic-name
{ ALTERNATE-CONSOLE    }
{ ALTERNATE-CONSOLE-q  }
{ TERMINAL             }
{ TERMINAL-q           }
```

where -q may be -0, -1, -2 or -X.

The above format of the DISPLAY statement and SPECIAL-NAMES paragraph can be used to output data to the operator's console or to any sequential-output-file in SSF format.

If the "UPON mnemonic-name" option is not used, SYSOUT is normally assumed to be the output device. If no suffix is specified for the implicit or explicit device, -1 for SYSOUT and -2 for other devices is assumed when COBOL85 is specified in the CBL JCL statement, -0 is assumed when COBOL74 is specified. These defaults can be overridden by using the DISPLAY IS, SYSOUT IS, DISPLAY CONSOLE IS, DISPLAY ALTERNATE-CONSOLE IS and DISPLAY TERMINAL IS phrases in the Default Section (not part of the ANS standard).

If mnemonic-name specifies SYSOUT, the output file may be a standard SYSOUT subfile or a permanent SYSOUT file. If the output file is a standard SYSOUT subfile, no ASSIGN JCL statement is needed when the program is executed. However, if the output file is a permanent SYSOUT file an ASSIGN JCL statement must be used to assign the internal-file-name "H_PR" to the SYSOUT file.

For example:

```
ASSIGN H_PR,OUTFILE,DEVCLASS=MS/M402,MEDIA=DISOUT, FILESTAT=UNCAT;
```

In this example OUTFILE is an uncataloged sequential disk file.

For a standard SYSOUT subfile, records are output as a sequence of 120 column lines. For a permanent SYSOUT file the number of output records is variable and depends upon the maximum record length of the file.

If mnemonic-name specifies CONSOLE or CONSOLE-q, no ASSIGN JCL statement is needed. The following message will be displayed on the console when the program is executed:

```
hh:mm ron progid user-data...
```

where:

hh:mm	is the time at which the message was displayed.
ron	is the run-occurrence-number.
progid	is the program-id specified in the COBOL program.
user-data	is the data displayed by the user program.

Data will be displayed at 64 characters per line. Each display that is made on the console will be echoed in the JOR prefixed by the report code "CBL11".

If mnemonic-name specifies ALTERNATE-CONSOLE OR ALTERNATE-CONSOLE-q, data will be displayed on the alternate operator's console specified in the CONSOLE JCL statement (see the *JCL Reference Manual*). If no CONSOLE JCL statement is used, data will be displayed on the console that submitted the program. If the submitting console is no longer logged, the message is stored in the user’s mailbox. The format of the console dialogue is the same as that on the main console.

If mnemonic-name specifies TERMINAL and the load module is interactively executed from a terminal under the Interactive Operation Facility, data will be displayed prefixed by three blanks upon the terminal being used. If the load module is not executed from a terminal but is executed as part of a batch job, the DISPLAY will behave as if mnemonic-name specifies ALTERNATE-CONSOLE. If a break is entered on the console while a message is being output, the remaining part of the data to be output is lost.

When the device name suffix is -0, if WITH CONVERSION is specified and identifier-1 references an elementary numeric or Boolean data item, the data item is moved to a temporary data item (not defined by the user) with a USAGE DISPLAY, SIGN LEADING SEPARATE (if the item is numeric) and the same picture (if any) of the data item to be displayed. The temporary data item is then displayed, instead of the original user data item, in order to obtain a legible decimal output. If WITH CONVERSION is specified and identifier-1 references an elementary pointer data item, the 8-hexadecimal representation of the bit-coded value of the pointer data item is displayed. If WITH CONVERSION is not specified, data is displayed as in memory, without conversion. If the text to be displayed contains an unprintable character, and the DEBUG parameter is included in the STEP JCL statement when the program is executed, the hexadecimal value of this character is printed on the two lines below the erroneous character.

When the device name suffix is -X, the hexadecimal representation of the bit-coded value of the data item referenced by identifier-1 is displayed.

When the device name suffix is -1 or -2, the data item referenced by identifier-1 is converted to a legible format (if necessary) before being displayed (See the *COBOL 85 Reference Manual*, the legible equivalent).

11.4.3 Selection of the I/O Device

The variables governing the selection of the I/O device to be used for ACCEPT and DISPLAY statements are summarized in the table below.

Variables Governing the Selection of I/O Devices

Device Specified by Mnemonic-name in ACCEPT or DISPLAY	Is there a Console JCL statement in the Job Step?	Type of Job submission		
		BATCH	ROF	IOF
CONSOLE	No	M	M	M
	Yes	M	M	NA
Alternate- CONSOLE	No	M	T(1)	T
	Yes	User-name	User-name	NA
TERMINAL	No	M	T	T
	Yes	User-name	User-name	NA

Notes for the table "Variables Governing the Selection of I/O Devices":

M	The main system console is used for I/O.
User name	The terminal identified by user-name in the CONSOLE JCL statement is used for I/O.
T	The terminal that submitted the program for execution is used for I/O.
NA	Not available.
(1)	The main system console is used for I/O if the submitting ROF terminal is no longer logged.

Note that the use of TERMINAL is to be preferred to ALTERNATE-CONSOLE for interactive jobs submitted via IOF.

11.4.4 The STOP Literal Statement

The format of the STOP literal statement is:

```
STOP literal
```

This statement is used to suspend execution of the COBOL program.

When this occurs, the following message is displayed on the main operator's console:

```
nn/hh:mm ron progid STOP literal
```

where:

nn	is a message number that the operator must enter when replying to this message.
hh:mm	is the time at which the message was displayed.
ron	is the run-occurrence-number.
progid	is the program-id specified in the COBOL program.

In order to restart the program, the operator must enter the message number, one space and carriage-return.

Each STOP literal will be echoed in the JOR prefixed by the report code "CBL17".

Note that the effect of the STOP literal statement is the same as a DISPLAY literal UPON mnemonic-name statement followed by an ACCEPT dummy-data-name FROM mnemonic-name statement. Associating mnemonic-name with ALTERNATE-CONSOLE or TERMINAL enables the program to direct such a simulated STOP literal statement to the desired device, if it is not the main operator's console.

11.5 Using Diskettes

In a COBOL program diskette files are handled as UFAS sequential files. Furthermore, overriding rules enable users to process Diskette files in direct access or as Foreign files.

12. Miscellaneous Topics

This chapter includes various topics that are not discussed at length in this manual and therefore do not warrant individual chapters.

12.1 Sorting and Merging

The following paragraphs compare the use of the COBOL SORT and MERGE statements with the SORT and MERGE utilities. For further details concerning the use of the SORT and MERGE utilities see the *Sort/Merge User's Guide*.

12.2 COBOL SORT/MERGE and SORT/MERGE Utility

12.2.1 Making the Choice Between COBOL and the Utility for SORT/MERGE

The choice between using the COBOL SORT/MERGE statements and executing SORT/MERGE as separate utilities is basically the choice between flexibility and performance. The following points must be taken into account:

- Using input and output procedures with the COBOL SORT and MERGE statements makes it possible to combine the first and last phases of the sort or merge with processing of the released or returned record (e.g., record selection, editing).
- Execution of SORT/MERGE as a utility rather than as a COBOL statement saves central processor time. (The ratio COBOL SORT / SORT utility is approximately 1.2).
- The commands of the SORT utility introduce programmed functions such as: INCLUDE, OMIT, ARRANGE, etc.

Therefore, the SORT utility should be used whenever one of the following conditions is fulfilled:

- The input and output files do not need to be processed immediately before or after the sort, or,
- Processing of the input and output files can be done using the SORT utility commands.

If these conditions are not fulfilled, the processing of the input and output files and the sorting of the file should be combined in a single program. File processing could, of course, be carried out in two separate COBOL programs separated by the SORT utility. However, if these three operations are done in a single program, two file passes are saved: the intermediary files between the COBOL programs and the SORT utility do not have to be written or read.

The above rules for using the two forms of SORT can be applied to MERGE except that processing of input files (INPUT PROCEDURE) is not possible when merging.

12.2.2 "Millennium" and "Rule 61" Keywords in SORT Sub-routines

COBOL programmers wishing to SORT 2-digit decimal keys according to the sequence resulting from Rule 61 will write, for example,

```
SORT <SD name> {ASCENDING/DESCENDING {<data name> [DATE]}...}... ...,
```

The Rule 61 sequence is:

```
61  62  ...  99  00  01  ...  59  60
```

All COBOL programs calling a SORT sub-routine can modify parameters by passing a chain of options containing SORT(...). To change the beginning of the century dynamically, it suffices to pass a sub-chain of SORT sub-routine options such as SORT(...LOWYEAR=0 to 99...).

For more information refer to *SORT/MERGE Utilities User's Guide* and *How to Deal with the Year 2000 User's Guide*.

12.3 JCL for COBOL SORT

The following paragraphs describe the JCL for COBOL programs which use the SORT statement. The JCL for the COBOL MERGE statement is not discussed, as this only involves assigning the input and output files.

A SORT statement must refer to a file described in an SD in the File Section. This file must have a SELECT clause in the Input-Output Section. But, when the program is executed, no ASSIGN JCL statement is needed for this file. Note that the ORGANIZATION, ACCESS MODE, RESERVE, FILE STATUS and RECORD KEY phrases cannot be used in the SELECT clause for a file with an SD.

The sort process uses a work file in addition to the input and output files. The user can specify the work file on disk, what size disk file is to be used, and so on. This information can be specified in the optional SORTWORK statement that has the following format:

```
          { WKDISK  }
SORTWORK  {         } =
          { WKDISKS }

          { SIZE = digit5 [dvcmd]                          }
        ( { external-file-name                             } )
          {     [ { FILESTAT = CAT  CATALOG = digit1 } ]   }
          {     [ { [ FILESTAT = UNCAT ] [dvcmd]     } ]   }
```

where dvcmd is defined as follows:

```
{ RESIDENT                                                 }
{ { DEVCLASS }                                             }
{ {          } = device-class                              }
{ { DVC      }                                             }
{             { MEDIA }   { volume-name                  } }
{             {       } = {                               } }
{             { MD    }   { (volume-name [volume-name...])} }
```

The WKDISK parameters of the SORTWORK statement are used in exactly the same way as the WKDISK parameters of the SORT JCL statement. See the *Sort/Merge User's Guide* for an explanation of these parameters.

If the SORTWORK JCL statement is not used, a temporary work file will be allocated on a resident disk volume with the external file name H_SRTWKD. The size of this file will be calculated from the number of records specified following the SORTSIZE keyword in the OPTIONS parameter of the STEP JCL statement (see below). If the SORTSIZE keyword is not used, 10 cylinders will be allocated for this file.

If both the WKDISK parameter of the SORTWORK JCL statement and the SORTSIZE keyword are used, the amount of space needed in the work file will be calculated. If the work file is not large enough to sort the specified number of input records, the program will abnormally terminate before sorting begins.

The OPTIONS parameter of the STEP JCL statement can be used to pass a character string to a load module at the start of execution (see "Step Options", below). The OPTIONS parameter can also be used to pass a string of keywords to the sort process when the program being executed contains a COBOL sort. These keywords are also passed to the COBOL program as a normal user string. They are coded as follows:

```
STEP...
       OPTIONS = 'SORT ( [ AVERAGE = digits5 ]
                         [ MIN = digits5 ]
                         [ SORTSIZE = digits11 ]
                         [ SIZE = {512 | digits5} ]
                         [ NBSORT = {1 | digits2} ]
                         [ {REPORT | NREPORT} ]
                         [ MEMSORT ]
                         [ LOWYEAR = {61 | digits2} ]
                         [ REPEAT ]
                         [ DEBUG = digit1 ] );
                            optional-user-string'
```

The NBSORT parameter specifies the number of slave processes to be used in the subroutine sorts of the step, when performing parallel subroutine sorts. See 12.3.2 "JCL GSORTWRK and Parallel Sort".

The AVERAGE, MIN, SORTSIZE, REPORT and NREPORT keywords are used in exactly the same way as the AVERAGE, MIN, SORTSIZE, REPORT and NREPORT commands of the SORT utility. The SIZE keyword is used in exactly the same way as the SIZE parameter in the SORT JCL statement. The MEMSORT keyword specifies that a memory sort is required. The REPEAT keyword specifies that checkpoints are to be taken during the sort process. The DEBUG keyword specifies various levels of internal controls. See the *Sort/Merge User's Guide* for details. The keywords may appear in any order and they are all optional. However, the keywords must appear before any user data that may be included in the OPTIONS string.

The SIZE parameter, from GCOS 7-V7 onwards, can specify up to 64 Mbytes of memory (SIZE=65536 maximum) for a Large Memory Sort (see below).

The MEMSORT keyword implies a mandatory memory sort. When MEMSORT is not specified a memory sort will be done whenever possible.

The LOWYEAR parameter specifies the beginning of the century.

The workfile can be incremented by sort if exhausted in two cases:

- pre-allocated workfile with INCRSIZE specified at PREALLOC level;
- dynamic workfile assignment by sort (no JCL assignment).

In the latter case (dynamic workfile) the workfile size will depend on SORTSIZE:

- SIZE = 10 cylinders without SORTSIZE (INCRSIZE = 2);
- SIZE and INCRSIZE computed by sort if SORTSIZE is specified.

When REPEAT is used, the user program must not take any checkpoint inside the input and output procedures. Beware of GAC files open and CALL "H_GAC_UCOMIT" which imply checkpoint calls.

12.3.1 Large Memory Sort

GCOS 7-V7 introduced a Large Memory Sort which can use up to 64 Mbytes of memory. To use the 64 Mbytes memory while sorting from COBOL, you must do the following:

1. Re-link the COBOL program(s) using the following LINKER commands:

```
REPLACE = (H_SR_UCOBSRT, H_TD_TRID),
SEGTAB1 = (SHRLEVEL = 2, VSEG = 4),
SEGTAB2 = (SHRLEVEL = 3, VSEG = 4),
SEGTAB3 = (SHRLEVEL = 3, VSEG = 4),
```

2. Set the SIZE option at step level to the required value (lower than 65536 Kbytes). The default value is 512 Kbytes.

If you omit the REPLACE command, the run-time CU of the sort is included in the load module (by the LINKER). If the calling program creates too many segments itself and therefore leaves too few vacant entries for the sort (VMM RC = ENTRYOV), then the sort executes with the memory it is able to get (unless the JCL STEP option MEMSORT is specified).

The SEGTAB command enables the sort to use 12 large segments of 4 Mega-bytes each (8 of type 3, 4 of type 2). The sort will create small type 2 or small type 3 segments to use the rest of memory available to the sort. However, if the calling program creates too many segments itself and therefore leaves too few vacant entries for the sort (VMM RC = ENRTYOV), then the sort may still not be able to use 64 Mbytes of memory. In each case, the sort executes with the memory it is able to get. If there is more than 60 small segments entries of each type, then the sort leaves at least 60 vacant entries of each type for the requests of the caller and the GCOS 7 services.

12.3.2 JCL GSORTWRK and Parallel Sort

A new JCL statement, GSORTWRK, compatible with that of other GCOS 7 utilities, is introduced in GCOS 7-V7 and may be used as an alternative to SORTWORK, whether the sort is a parallel (multi-process) one or not.

The syntax of this JCL statement is as follows:

```
GSORTWRK
{WKFILE=(asg) [,WKALC=(alc)]; |
 [MODALC = (alc),] WKASG1 = (asg)[,WKALC1=(alc)]
                   WKASG2 = (asg) [,WKALC2=(alc)]
                     .
                     .
                   WKASGn = (asg) [,WKALCn=(alc)]; }
```

(asg) and (alc) — Indicate standard groupings of assign and allocate, defined in the *JCL Reference Manual* and the *JCL User's Guide.*

MODALC — Indicates the standard allocation parameters that are attributed to each of the WKASGi, thus avoiding attributing parameters explicitly i times.

If a WKASGi also has an explicit parameter value, it conserves this explicit value, even if there is a corresponding value in MODALC.

For a multi-process sort in a subroutine, the above syntax is used. For a mono-process sort in a subroutine, the following syntax is used:

```
GSORTWRK WKFILE = (asg) [,WKALC=(alc)];
```

The multi-process sort, or parallel sort, is described in the *Sort/Merge User's Guide.*

When using a multi-process sort as a subroutine, you must re-link it to the calling program with the following commands:

```
TASK = (H_SUB_SORT START=H_TD_CLONE OCNB=32),
SEGTAB1 = (SHRLEVEL=2 VSE6=4),
SEGTAB2 = (SHRLEVEL=3 VSE6=4),
SEGTAB3 = (SHRLEVEL=3 VSE6=4),
```

The parallel sort enables you to use more than 64Mbytes of memory, and to perform simultaneous input/output operations on several workfiles at the same time. This is particularly useful if the input and output files are read and written very quickly (as is the case when the batch booster facility is used. See 12.3.3.).

However, the following restrictions exist for the parallel sort:

- there is no nested parallel subroutine sort
- there are no checkpoints in the job (multi-process LM)
- the same set of workfiles is used for all the successive parallel subroutine sorts.

To request a multi-process sort in a subroutine, use the NBSORT parameter of the "OPTIONS =" string (see 12.3.1). The value "n" given to NBSORT will indicate the number of parallel processes to be used.

File assignation for a parallel sort may take one of the following forms:

```
1)  GSORTWRK WKFILE=(asg)[,WKALC=(alc)];

        where asg applies to at least n media.

2)  GSORTWRK     [MODALC=(alc),]
                WKASG1=(asg) [,WKALC1=(alc)]
                          .
                          .
                WKASGn=(asg) [,WKALCn=(alc)];
```

The workfile of a parallel sort has the same implicit values as the unique workfile of a mono-process sort.

NOTES:

1. A parallel sort uses at least 512Kbytes of memory for each process. The Declared Working Set (DWS) must be adjusted to take account of this (SIZE=xx in the STEP statement).

2. You are advised not to use the parallel sort as a subroutine, but to reserve it for very large stand-alone sorts.

   If you do use it as a subroutine, you are recommended to give the NBSORT parameter a value from 2 to 4 (2, 3 or 4), except for "all in memory" sorts.

   When an "all in memory sort" is to be executed, NBSORT has the exclusive role of reserving up to 32 x 32 Mbytes = 1 Gbyte of memory (32 Mbytes for each process, the value of NBSORT being limited to 32).

12.3.3 Batch Booster

GCOS 7-V6 (TS6150) introduced the Batch Booster or BPB facility. This is described in the manual *UFAS-EXTENDED User's Guide*. The BPB parameter exists in the JCL statement DEFINE (in the case of a sort called from a program STEP) and in the INFILE and OUTFILE parameter groups (stand-alone sort).

BPB and Large Memory Sort can be used together. In this case, you must specify a value of BPB greater than 1 and you must allow for the memory needed for the extra buffers (requested by BPB) in calculating the memory allocated to the sort (SIZE option at step level).

If the UFAS access method is called, the UFAS BPB function is executed (if BPB is available on your system).

12.3.4 Examples of a COBOL SORT

12.3.4.1 Using a Sort in a COBOL Program

```
$JOB CLS, USER=PJJC, PROJECT=LX28;

COMM  'THE NEXT STATEMENT EXECUTES THE COBOL COMPILER RESULTING
       IN THE TRANSLATION OF THE COBOL PROGRAM CONTAINED IN THE
       INPUT ENCLOSURE SAMPLESORT';

CBL    SOURCE = *SAMPLESORT, XREF;

COMM  'THE NEXT STATEMENT EXECUTES THE LINKER WHICH, USING THE
       OUTPUT OF THE COMPILATION, WILL BUILD A LOAD MODULE NAMED
       SAMPLESORT AND STORE IT IN A LIBRARY NAMED PLML';

LINKER SAMPLESORT, OUTLIB = (PLML, DEVCLASS=MS, MEDIA=BD54);

COMM  'THE FOLLOWING STEP ENCLOSURE EXECUTES THE LOAD MODULE
       SAMPLESORT. SINCE THE PROGRAM SAMPLESORT USES THE SORT
       VERB, A SORTWORK JCL STATEMENT IS REQUIRED WITHIN THE STEP.
       NOTE ALSO THAT TWO ASSIGN JCL STATEMENTS ARE PRESENT,
       ONE WITH INTERNAL FILE NAME SAMPLE, THE OTHER WITH INTERNAL
       FILE NAME SORTOUT.';

STEP   SAMPLESORT, (PLML, DEVCLASS=MS, MEDIA=BD54);

ASSIGN SAMPLE, INVMAST, FILESTAT=UNCAT, DEVCLASS=MT, MEDIA=TV46;
ASSIGN SORTOUT, PWFILE, FILESTAT=UNCAT, DEVCLASS=MS, MEDIA=BD22;

SORTWORK WKDISK = (PRISRTWK, DEVCLASS=MS, MEDIA=BD14);

ENDSTEP;

COMM  'THERE NOW FOLLOWS THE INPUT ENCLOSURE SAMPLESORT WHICH
       CONTAINS THE COBOL SOURCE PROGRAM';

$INPUT SAMPLESORT TYPE=COBOL;
       IDENTIFICATION DIVISION.
       PROGRAM-ID. SAMPLESORT.
       ENVIRONMENT DIVISION.
       INPUT-OUTPUT SECTION.
       FILE-CONTROL.
         SELECT SAMPLEFILE ASSIGN SAMPLE.
         SELECT SORTEDFILE ASSIGN SORTOUT.
         SELECT SORT-FILE ASSIGN H-SORT.
            .
            .
            .
```

```
        DATA DIVISION.
        FILE SECTION.
        FD SAMPLEFILE.
        01 INREC                PIC X(80).
        FD SORTEDFILE.
        01 SEQREC               PIC X(80).
        SD SORT-FILE.
        01 SORTREC.
           02 DATA-1            PIC X(10).
           02 KEYFLD            PIC X(5).
           02 DATA-2            PIC X(65).
               .
               .
        PROCEDURE DIVISION.
               .
               .
               .
        SORT-PARA.
           SORT SORT-FILE ON ASCENDING KEY KEYFLD
              USING SAMPLEFILE
              GIVING SORTEDFILE.
               .
               .
        END-PARA.
             STOP RUN.

$ENDINPUT;
$ENDJOB;
```

12.3.4.2 Using a Parallel Sort in a COBOL Subroutine

Parallel Sort with the SORTWORK Statement

```
LINKER SAMPLESORT, OUTLIB=USER.LMLIB,
                 , COMFILE=*MULTI-PROCESS;
$INPUT MULTI-PROCESS, PRINT;
ENTRY=SAMPLEENTRY,
TASK=(H_SUB_SORT START=H_TD_CLONE OCNB=32),
SEGTAB1 = (SHRLEVEL=2 VSE6=4),
SEGTAB2 = (SHRLEVEL=3 VSE6=4),
SEGTAB3 = (SHRLEVEL=3 VSE6=4),
$ENDINPUT;

STEP SAMPLESORT LIB=USER.LMLIB,
ASSIGN SAMPLE, INVMAST, FILESTAT=UNCAT, DEVCLASS=MT, MEDIA=TV46;
ASSIGN SORTOUT, PWFILE, FILESTAT=UNCAT, DEVCLASS=MS/FSA,
                        MEDIA=BD22;
```

```
OPTIONS='SORT (NBSORT=2, SIZE=2048)';
SORTWORK WKFILE=(A,MD=(VOL1,VOL2),DVC=MS/FSA),
         WKALC=(SIZE=40,UNIT=CYL);
SIZE 2048;
ENDSTEP;
```

In the above example, the sort subroutines of the SAMPLESORT program are executed with two slave processes (plus the master process P=0), requiring 2048 Kbytes of memory. One slave process uses a workfile of 20 cylinders on the VOL1 media, and the other slave process uses a workfile of 20 cylinders on the VOL2 media.

Parallel Sort with the GSORTWRK Statement

```
LINKER SAMPLESORT, OUTLIB=USER.LMLIB,
                 , COMFILE=*MULTI-PROCESS;
$INPUT MULTI-PROCESS, PRINT;
ENTRY=SAMPLEENTRY,
TASK=(H_SUB_SORT START=H_TD_CLONE OCNB=32),
SEGTAB1 = (SHRLEVEL=2 VSE6=4),
SEGTAB2 = (SHRLEVEL=3 VSE6=4),
SEGTAB3 = (SHRLEVEL=3 VSE6=4),
$ENDINPUT;

STEP SAMPLESORT LIB=USER.LMLIB,
ASSIGN SAMPLE, INVMAST, FILESTAT=UNCAT, DEVCLASS=MT, MEDIA=TV46;
ASSIGN SORTOUT, PWFILE, FILESTAT=UNCAT, DEVCLASS=MS/FSA,
               MEDIA=BD22;
OPTIONS='SORT (NBSORT=2, SIZE=2048)';
GSORTWRK WKASG1=(W1,MD=VOL1 DVC=MS/FSA FILESTAT=UNCAT),
         WKASG2=(W2,MD=VOL2 DVC=MS/FSA FILESTAT=UNCAT);
SIZE 2048;
ENDSTEP;
```

In this example, the sort subroutines called by the SAMPLESORT program are executed with two slave processes (plus the master process P=0), requiring 2048 Kbytes of memory. One slave process uses the pre-allocated file W1, and the other slave process uses the pre-allocated file W2.

12.4 User JCL Status

The system sets a status value, which can be used in a JUMP JCL statement in the event of an abnormal step termination (STATUS=10000), in an operator-requested end of step (STATUS=50000) or in case of system crash (STATUS=61000). The COBOL compiler also sets the status value at the end of compilation, according to errors detected (see "The Compiler", Chapter 2).

The user may also set the status value in his COBOL program, transmitting it to the run-time package routine H_CBL_USETST via a field described in the Working-Storage Section with usage COMP-1. Since COMP-1 is a binary half-word the user status value has a limit of 32767.

The following example shows how the status value can be set in a COBOL program:

```
     .
     .
WORKING-STORAGE SECTION.
01 STATE COMP-1.
     .
     .
PROCEDURE DIVISION.
     .
     .
   MOVE 64 TO STATE.
   CALL "H_CBL_USETST" USING STATE.
     .
     .
```

Execution of the job stream can then be modified by testing this status value:

```
$JOB...
.
.
STEP TEST01, TEMP, DUMP=DATA;
ENDSTEP;
JUMP LAB1,STATUS,EQ,64;
SEND 'STATUS DIFFERENT FROM 64';
JUMP LAB2;
LAB1: SEND 'STATUS = 64';
LAB2: SEND 'END OF TEST';
$ENDJOB;
```

The JOR will then show:

```
PROCESS GROUP TERMINATED STATUS = 64
```

if the CALL statement is executed.

12.5 Switches

Each COBOL program has access to 32 switches contained in the switch word assigned to the job in which the program is executed.

Switches are declared in the SPECIAL-NAMES paragraph, where they may be associated with mnemonic-names as well as with condition-names for ON STATUS and OFF STATUS.

A switch, once declared, may be turned ON or OFF by the COBOL SET statement, while its current status may be tested using the associated condition-name.

COBOL programs can use switches to communicate with steps that follow in the job, as well as with the job itself. JCL can also turn switches on and off (LET) and test them (JUMP). The operator may set switches when starting a job via the Start Job (SJ) command, or while the job is in execution via the MODIFY_JOB (MJ) command. The initial setting of the switch word is all zeros (i.e., all switches off) at the beginning of the job. If the job is initiated by operator action (RJ or SJ) or by another job (RUN) the SW parameter permits the switch word to be set to some other initial value.

The use of switches is shown in the following example:

```
      .
      .
CONFIGURATION SECTION.
      .
      .
SPECIAL-NAMES.
     SWITCH-2 IS SW2 ON STATUS IS CND2.
      .
      .
PROCEDURE DIVISION.
      .
      .
SET SW2 TO ON.
      .
      .
IF CND2 DISPLAY "SWITCH-2 ON".
      .
      .
```

Using switches is a way to handle the break mechanism, for a program running on an interactive terminal. The dialog on the terminal would be to press the break key and answer to "???" prompt:

```
??? MDJ ron ON=n;
```

12.6 Checkpoint, Restart and Journalization

The RERUN clause in the I-O-CONTROL paragraph allows the user to specify the conditions, if any, under which checkpoints (or commitments) are to be taken during program execution. If at the time that the RERUN condition arises, GAC-EXTENDED is fully effective for the step (locks may be applied to at least one opened file that is assigned with SHARE=MONITOR parameter), a commitment is taken. If GAC-EXTENDED is not fully effective, a checkpoint is taken. Checkpoints (or commitments) can be taken at each end of volume in a specified file or each time a specified number of records is read or written in a specified file. The checkpoint (or commitment) may be somewhat delayed depending on I/O events; usually, no checkpoint (or commitment) is taken for an I/O operation that does not return a "00" status.

Checkpoint data are placed in Backing Store. If the program aborts or there is a system crash, and the STEP statement contains the REPEAT parameter, the operator may call for the program execution to be restarted. If he does so, the program is restored to its state at the last checkpoint and execution continues from there. The REPEAT parameter of the $JOB statement can be used to request the restart of an entire job.

The user can also request checkpoints in the execution JCL. See the DEFINE JCL statement in the *JCL Reference Manual*.

At the price of introducing a non-standard element into his source program, the user may also directly call the system checkpoint procedure H_CK_UCHKPT, giving two parameters. For example:

```
CALL "H_CK_UCHKPT" USING RMODE, INFO.
```

RMODE is a user-defined USAGE COMP-2 field which indicates whether the current execution of the program is the first execution (RMODE = zero) or if the program has been restarted (RMODE not = zero). In the latter case, RMODE contains the JCL status value for the abnormal step termination, which also appears in the JOR.

INFO is a user-defined group item consisting of 32 one-character elements. Where all elements are zero after a checkpoint, the checkpoint was correctly executed. If not, those elements with the value 1 indicate what went wrong.

Regardless of whether a checkpoint is taken as a result of the RERUN clause or a programmed CALL, these values can be checked by coding:

```
CALL "H_CK_UMODE" USING RMODE INFO.
```

where RMODE and INFO have the same meaning as for H_CK_UCHKPT. This CALL also introduces a non-standard element into the user's source program, which will require alteration to run on any other system.

Associated with checkpointing is "journalization". This is a facility offered by Data Management which keeps a record of all file updates so that files can be reconstituted before a rerun is performed.
More information on the use of the above facilities are given in the *System Administrator's Manual* and the *JCL User's Guide*. The commitment mechanism and the ability for a COBOL user to call directly the system commitment procedure H_GAC_UCOMIT are fully described in the *GAC-EXTENDED User's Guide*.

12.7 Access to System Functions

The access to system functions through COBOL is described in the *System Calls From COBOL Manual*.

12.8 Alphabets

The following COBOL Language elements are discussed below:

In the SPECIAL-NAMES Paragraph

```
                              { STANDARD-1              }
                              { STANDARD-2              }
                              { NATIVE                  }
alphabet-name is              { ASCII                   }
                              { EBCDIC                  }
                              { JIS                     }
                              { GBCD                    }
                              { user-specified-alphabet }
```

In the OBJECT-COMPUTER Paragraph

```
                              { alphabet-name }
                              { STANDARD-1    }
                              { STANDARD-2    }
PROGRAM COLLATING SEQUENCE IS { NATIVE        }
                              { ASCII         }
                              { EBCDIC        }
                              { JIS           }
                              { GBCD          }
```

In the FILE Section

```
                          { alphabet-name }
                          { STANDARD-1    }
                          { STANDARD-2    }
CODE SET IS               { NATIVE        }
                          { ASCII         }
                          { EBCDIC        }
                          { JIS           }
                          { GBCD          }
```

In the SORT and MERGE Statements

```
                          { alphabet-name }
                          { STANDARD-1    }
                          { STANDARD-2    }
COLLATING SEQUENCE IS     { NATIVE        }
                          { ASCII         }
                          { EBCDIC        }
                          { JIS           }
                          { GBCD          }
```

Note that STANDARD-1, STANDARD-2, NATIVE, ASCII, EBCDIC, JIS and GBCD are not part of the ANS standard for the OBJECT-COMPUTER paragraph, File Section or SORT and MERGE statements. They are standard for the SPECIAL-NAMES paragraph only. See the *COBOL 85 Reference Manual* for an explanation of STANDARD-1, STANDARD-2, NATIVE, ASCII, EBCDIC, JIS and GBCD.

The alphabet-name clause provides a means of relating a name to a specified character code set and/or collating sequence. When alphabet-name is referenced in the PROGRAM COLLATING SEQUENCE clause of the OBJECT-COMPUTER paragraph or the COLLATING SEQUENCE phrase of a SORT or MERGE statement, the alphabet-name clause specifies a collating sequence. When alphabet-name is referenced in a CODE-SET clause in a file description entry, it specifies a character code set.

The collating sequence of each alphabet is given in an appendix of the *COBOL 85 Reference Manual*. This appendix shows the hexadecimal value, graphic symbol and symbolic character number for each character in the alphabet.

Whichever alphabet is specified, non-numeric data is always stored in memory in NATIVE (EBCDIC) form. If another alphabet is specified for comparison, collating or I/O, code conversion is carried out (by software). The circumstances under which code conversion is carried out are discussed below.

12.8.1 PROGRAM COLLATING SEQUENCE

PROGRAM COLLATING SEQUENCE in the OBJECT-COMPUTER paragraph indicates the collating sequence to be used for non-numeric comparisons involving the following operators:

```
IS  NOT  GREATER THAN
IS  NOT  LESS THAN
IS  NOT  >
IS  NOT  <
IS GREATER THAN OR EQUAL TO
IS >=
IS LESS THAN OR EQUAL TO
IS <=
EXCEEDS
```

The data to be compared is converted into the collating sequence indicated by PROGRAM COLLATING SEQUENCE before the comparison is made. PROGRAM COLLATING SEQUENCE has generally no effect on non-numeric comparisons involving the following operators:

```
IS  NOT  EQUAL TO
IS  NOT  =
IS UNEQUAL TO
EQUALS
```

These comparisons are made without prior conversion, unless the alphabet-name referenced by PROGRAM COLLATING SEQUENCE is described with the ALSO clause.

12.8.2 SORT and MERGE COLLATING SEQUENCES

COLLATING SEQUENCE in the SORT and MERGE statements has an effect similar to PROGRAM COLLATING SEQUENCE described above: the sort and merge keys will be converted according to the specified collating sequence before key comparison is made. This does not affect the record stored in the COBOL program.

12.8.3 CODE-SET

The CODE-SET clause enables data to be input from or output to files in code sets other than NATIVE (EBCDIC). These files must contain display items only and all signs must be specified as separate. CODE-SET operates in the following way.

- Immediately after a record is read, the record is converted from the code specified in CODE-SET into NATIVE code.
- Immediately before writing or rewriting a record, it is converted from NATIVE code into the code specified by the CODE-SET clause.
- The CODE-SET clause is ignored at execution time if code conversion is done by hardware (e.g., for cards or ANS magnetic tape).

12.8.4 HIGH-VALUE LOW-VALUE

The character with the highest ordinal position in the PROGRAM COLLATING SEQUENCE is used for the figurative constant HIGH-VALUE. The character with the lowest ordinal position in the PROGRAM COLLATING SEQUENCE is used for the figurative constant LOW-VALUE. These characters are shown below.

HIGH-VALUES and LOW-VALUES

COLLATING SEQUENCE	HIGH-VALUE	LOW-VALUE
STANDARD-1 (ASCII)	""256""	""1""
STANDARD-2	""256""	""1""
NATIVE (EBCDIC)	""256""	""1""
JIS	""256""	""1""
GBCD	"!"	"0"

Notes on the table "HIGH-VALUES and LOW-VALUES":

"0"	is zero
"!"	is exclamation mark
""1""	is hexadecimal 00
""256""	is hexadecimal FF

Values contained in two sets of quotation marks are "symbolic-character numbers". That is, they specify a particular hexadecimal value in the relevant collating sequence.

12.8.5 STEP OPTIONS

The OPTIONS parameter of the STEP JCL statement enables a character string to be passed to a load module at the start of execution.

The COBOL program accesses the character string from the OPTIONS parameter by including a Linkage Section in the main program of the load module. (The main program is the one named in the ENTRY parameter of the LINKER utility). The way in which the COBOL program should be written is shown in the following example.

```
    .
WORKING-STORAGE SECTION.
01 OPT1      PIC    X(20).
01 OPT2      PIC    X(20).
01 OPT3      PIC    X(20).
01 OPT4      PIC    X(20).
01 OPT5      PIC    X(20).
    .
LINKAGE SECTION.
01 SIGNED-LONG COMP-2.
01 TEXTE.
   02 ELEM PIC X OCCURS 1 TO 256 DEPENDING ON SIGNED-LONG.
    .
PROCEDURE DIVISION USING SIGNED-LONG TEXTE.
DEBUT.
   MOVE SPACE TO OPT1 OPT2 OPT3 OPT4 OPT5.
   IF SIGNED-LONG NOT = 0
      UNSTRING TEXTE DELIMITED BY "," INTO OPT1
      OPT2 OPT3 OPT4 OPT5.
    .
```

Suppose that the character string "123456,ABCDEFG,HIJK" is to be passed to the COBOL program. The STEP JCL statement would be:

```
STEP...OPTIONS = '123456,ABCDEFG,HIJK';
```

The above program has been written so that it can receive up to 5 twenty-character options with commas as delimiters. With the above STEP statement, this program will receive the following values:

```
OPT1: 123456
OPT2: ABCDEFG
OPT3: HIJK
```

Note that the OPTIONS parameter must be specified in the STEP JCL statement when the main program of the load module has the USING phrase in its Procedure Division header. Otherwise any reference to the parameter will result in an exception.

12.9 The Report Writer

The following paragraphs briefly describe the function of the Report Writer and provide advice on the use of Report Writer facilities. For a definition of the Report Writer statements see the *COBOL 85 Reference Manual*. See also The Report Writer in Chapter 6 of the current manual.

The Report Writer enables the programmer to produce reports by specifying the physical appearance of a report rather than by specifying the detailed procedures necessary to produce that report.

A hierarchy of levels is used in specifying the logical organization of a report. Each report is divided into report groups, which in turn are divided into sequences of items. Such a hierarchical structure enables explicit reference to other levels in the hierarchy. A report group contains one or more items to be output on one or more lines.

12.9.1 General Concepts

LINE-COUNTER is a special register that is generated for each report description (RD) entry in the Report Section of the Data Division. The implied description is that of an unsigned integer that must be capable of holding a range of values from 0 through 999999. The usage is COMP-2. The value in LINE-COUNTER is maintained by the Report Writer, and is used to determine the vertical positioning of a report. The value in LINE-COUNTER may be accessed by Procedure Division statements; however, only the Report Writer may change the value of LINE-COUNTER.

The reserved word PAGE-COUNTER is a name for a special register that is generated for each report description entry in the Report Section of the Data Division. The implicit description is that of an unsigned integer that must be capable of representing a range of values from 1 to 999999. The usage is DISPLAY. The value in PAGE-COUNTER is maintained by the Report Writer and is used to number the pages of a report. The value in PAGE-COUNTER may be altered by Procedure Division statements.

In the Report Section, neither a sum counter nor the special registers LINE-COUNTER and PAGE-COUNTER can be used as a subscript.

A report file is a sequential file and is subject to the following restrictions. An OPEN statement, specifying either the OUTPUT or EXTEND phrase, must have been executed prior to the execution of the INITIATE statement, and a CLOSE, without the REEL or UNIT phrase, must be executed for this file subsequent to the execution of the TERMINATE statement. No other input/output statement may be executed for this file.

Note that the CHANNEL-p IS mnemonic-name clause of the SPECIAL-NAMES paragraph cannot be associated with files written by the Report Writer. Also, any form control information specified in JCL statements for such files is ignored when the files are written. See Form Control, Chapter 11.

12.9.2 The Data Division

A REPORT clause is required in the FD entry to list the names of the reports to be produced.

In the Report Section the description of each report must begin with a report description entry (RD entry) and be followed by the entries that describe the report groups within the report.

In addition to naming the report, the RD entry defines the format of each page of the report by specifying the vertical boundaries of the region within each type of report group may be printed. The RD entry also specifies the control data items. When the report is produced, changes in the values of the control data items cause the detail information of the report to be processed in groups called control groups.

Each report named in the REPORTS clause of an FD entry in the File Section must be the subject of an RD entry in the Report Section. Furthermore, each report in the Report Section must be named in one and only one FD entry.

The report groups that will comprise the report are described following the RD entry. The description of each report group begins with a report group description entry; that is, an entry that has a 01 level number and a TYPE clause. Subordinate to the report group description entry, there may appear group and elementary entries that further describe the characteristics of the report group.

12.9.3 The Procedure Division

The INITIATE statement causes the Report Writer to begin the processing of a report.

The GENERATE statement directs the Report Writer to produce a report in accordance with the report description that was specified in the Report Section of the Data Division.

The SUPPRESS statement causes the Report Writer to inhibit the presentation of a report group.

The USE statement specifies Procedure Division statements that are executed just before a report group named in the Report Section of the Data Division is produced.

The TERMINATE statement causes the Report Writer to complete the processing of the specified report.

12.9.4 REPORT Clause in FD

A given report-name must appear in one and only one file description entry. The SELECT clause of a report file can only specify an SSF record format. If WITH SSF is not specified, it will be assumed. If neither VLR nor FLR is specified, WITH VLR is assumed except when the FD entry contains a format 1 RECORD clause (i.e. with neither the VARYING phrase nor the TO phrase). The RECORD CONTAINS clause in the FD entry of a report file is used to specify its record length. The default record length is 132 characters. For example:

```
      .
      .
ENVIRONMENT DIVISION.
      SELECT FILE-1 ASSIGN F1 WITH SSF FLR.
      SELECT FILE-2 ASSIGN F2.
      .
      .
DATA DIVISION.
FD    FILE-1 RECORD CONTAINS 121 CHARACTERS
             REPORT IS REPORT-A.
FD    FILE-2 REPORT IS REPORT-B.
      .
      .
```

In the above example FILE-2 is implicitly an SSF VLR file. The records for REPORT-A and REPORT-B will be written on FILE-1 and FILE-2 respectively. REPORT-A and REPORT-B cannot describe any line longer than 121 and 132 characters respectively.

12.9.5 Summing Techniques

The examples below show two coding techniques for the Report Section of the Data Division. Example 2 uses more complex statements than example 1 and will result in more efficient (faster) object code. The report description entry is as follows:

```
RD...CONTROLS ARE YEAR MONTH WEEK DAYE
```

EXAMPLE 1:

```
01   TYPE CONTROL FOOTING YEAR.
     05   SUM COST.
01   TYPE CONTROL FOOTING MONTH.
     05   SUM COST.
01   TYPE CONTROL FOOTING WEEK.
     05   SUM COST.
01   TYPE CONTROL FOOTING DAYE.
     05   SUM COST.
```

❑

EXAMPLE 2:

```
01   TYPE CONTROL FOOTING YEAR.
     05   SUM A.

01   TYPE CONTROL FOOTING MONTH.
     05   A SUM B.

01   TYPE CONTROL FOOTING WEEK.
     05   B SUM C.

01   TYPE CONTROL FOOTING DAYE.

     05   C SUM COST.
```

In example 2, one addition will be made for each day, one more for each week, and one for each month. In example 1, four additions will be made for each day.

❑

12.9.6 The Use of SUM

Unless each identifier is the name of a SUM counter in a TYPE CONTROL FOOTING report group at an equal or lower position in the control hierarchy, the identifier must be defined in the File, Working-Storage or Linkage Section. A SUM counter is algebraically incremented by the value of a SUM operand under the following circumstances.

- If the SUM operand is not a SUM counter and it is not associated with an UPON phrase, then the SUM counter is incremented just before the presentation of any TYPE DETAIL report group.
- If the SUM operand is not a SUM counter and it appears on the SUM clause with an UPON phrase, then the SUM counter is incremented just before the presentation of any TYPE DETAIL report group specified in the UPON phrase.
- If the SUM operand is a SUM counter, it is incremented just before presentation of the TYPE CF report group which contains this SUM counter.

In the following example, SUBTOTAL is incremented only when DETAIL-1 is generated.

EXAMPLE:

```
       .
FILE SECTION.
       .
   05  NO-PURCHASES PIC 99.
       .
REPORT SECTION.
RD...
01 DETAIL-1 TYPE DETAIL.
   05  COLUMN 30 PIC 99 SOURCE NO-PURCHASES.
      .
01 DETAIL-2 TYPE DETAIL.
      .
01 TYPE CONTROL FOOTING DAYE LINE PLUS 2.
      .
   05  SUBTOTAL COLUMN 30 PIC 999
       SUM NO-PURCHASES UPON DETAIL-1.
      .
01 TYPE CONTROL FOOTING MONTH
   LINE PLUS 2 NEXT GROUP NEXT PAGE.
      .
```

❑

12.9.7 SUM Routines

A SUM routine is generated by the Report Writer for each report. The SUM operands which are included for summing in this routine are those which are not SUM counters and which are associated with no UPON phrase.

A SUM routine is generated by the Report Writer for a DETAIL report group whose name is specified in at least one UPON phrase. The SUM operands included for summing in this routine are those which are associated with an UPON phrase which references this DETAIL report group.

A SUM routine is generated by the Report Writer for a CF report group which contains a SUM counter which is referenced in a SUM clause.

When a GENERATE detail-name statement is executed, the SUM routines for the report and the detail report group are executed in their logical sequence. When a GENERATE report-name statement is executed and the report contains one detail report group, the SUM routines are executed for the report and then for the DETAIL report group.

The following examples show the SUM routines which are generated by the Report Writer. In example 1 only one SUM routine is generated which is associated with the report. Example 2 illustrates how operands are selected when the UPON detail-name option is specified.

EXAMPLE 1:

The following statements are in the REPORT SECTION.

```
01 DETAIL-1 TYPE DE...
            .
            .
01 DETAIL-2 TYPE DE...
            .
            .
01 DETAIL-3 TYPE DE..
            .
            .
01 TYPE CF...
   05 TOTAL-1...SUM A, B, C.
            .
            .
01 TYPE CF...
   05 TOTAL-2...SUM B.
```

One SUM routine is generated for the report as follows:

```
ADD  A  TO  TOTAL-1.
ADD  B  TO  TOTAL-1.
ADD  C  TO  TOTAL-1.
ADD  B  TO  TOTAL-2.
```

❑

EXAMPLE 2:

In this example the same coding is used as in example 1, with one exception: the UPON detail-name option is used for TOTAL-1, as follows.

```
01  TYPE CF...
    05 TOTAL-1...SUM A, B, C UPON DETAIL-2.
```

The following SUM routines would be generated instead of those resulting from the calculations in example 1.

SUM routine for DETAIL-2:

```
ADD  A  TO  TOTAL-1.
ADD  B  TO  TOTAL-1.
ADD  C  TO  TOTAL-1.
```

SUM routine for the report:

```
ADD  B  TO  TOTAL-2.
```

❑

Page Breaks

The Report Writer page break procedure operates independently of the procedures that are executed after any control breaks (except that a page break will occur as the result of a NEXT PAGE option). Therefore, the programmer should be aware of the following:

- A control heading is not printed after a page heading except for first generation. If it is necessary to have the equivalent of a control heading at the top of each page, the information to be printed must be included as part of the page heading. However, as only one page heading may be specified for each report, the inclusion of control heading information in page headings should be done with care. This "control heading" will be the same for each page and may be printed at inopportune times.

- GROUP INDICATE items are printed after page and control breaks. The figure below contains a GROUP INDICATE clause and shows the run-time output.

```
REPORT SECTION
        .
        .
01 DETAIL LINE TYPE IS DETAIL LINE NUMBER IS PLUS
1.
   05  COLUMN IS 2 GROUP INDICATE PIC A(9)
       SOURCE IS MONTHNAME OF RECORD AREA (MONTH).
        .
        .
```

```
 (Execution output)
   . . . . . .
. . .

  FEBRUARY  15   A00...
                 A02...

  PURCHASES AND COST...
   . . . . . .
. . .

  FEBRUARY  21   A03...
                 A03...
```

Sample GROUP INDICATE Clause

12.9.8 WITH CODE Clause

When more than one report is being written on a file and these reports are to be selectively written, a unique two-character code known as the record identification code must be assigned to each of these reports. This is done using the WITH CODE clause. Note that if a report is written using the WITH CODE clause, this report should not be written in edited SYSOUT format (see Chapter 11) and should not be output directly to the printer.

When the WITH CODE clause is used, the code will be written as the first two characters of each record in the file. When the programmer wishes to print a report from this file, he must use a WRITER JCL statement specifying the desired code (See the *JCL Reference Manual*).

The following example shows how to create and print a report with a code. A Report Writer program contains the following statements.

```
   .
   .
ENVIRONMENT DIVISION.
   .
   .
DATA DIVISION.
FILE SECTION.
FD RPT-OUT-FILE RECORD CONTAINS 122 CHARACTERS
   REPORTS ARE REP-FILE-1 REP-FILE-2.
   .
   .
REPORT SECTION.
RD REP-FILE-1 CODE "AA"...
   .
   .
RD REP-FILE-2 CODE "BB"...
   .
   .
```

The RPT-OUT-FILE must be written on a tape or disk. A WRITER JCL statement could then be used to print only the report with code "AA", as follows.

```
WRITER (report-file-description), REPORT=AA, DATAFORM=SSF;
```

12.9.9 Control Footings and Page Format

Depending on the number and length of control footings (as well as the page depth of the report), it is possible that some of the specified control footings will not be printed on the same page if a control break occurs for a high level control. When a page-break condition is detected before all required control footings have been printed, the Report Writer will print the page footing (if specified), skip to the next page, print the page heading (if specified), and then continue to print control footings.

If it is necessary to print all the control footings on the same page the page must be formatted in the RD-Level entry for the report (by setting the FOOTING integer to a sufficiently low line number) to allow for the necessary space.

Note also the following example.

```
   .
   .
RD EXPENSE-REPORT CONTROLS ARE FINAL, MONTH, DAYE.
   .
   .
01 TYPE CONTROL FOOTING DAYE LINE PLUS 1
   NEXT GROUP NEXT PAGE.
   .
   .
01 TYPE CONTROL FOOTING MONTH LINE PLUS 1
   NEXT GROUP NEXT PAGE.
   .
   .

(execution output)

EXPENSE REPORT
   .
   .
MARCH   31.........36.40
   (output for CF DAYE)
MARCH TOTAL.........220.90
   (output for CF MONTH)
   .
   .
```

In the above example, the NEXT GROUP NEXT PAGE clause for the control footing DAYE is not activated when the control break is at MONTH level.

12.9.10 Floating First Detail Rule

The first presentation of a body group (CH, CF or DE) that contains a relative line as its first line, will have its relative line spacing suppressed and the first line will be printed on the line indicated either by FIRST DETAIL or INTEGER PLUS 1 of a NEXT GROUP clause from the preceding page. For example:

- If the body group shown below was the last to be printed on a page

  ```
  01 TYPE CF MONTH NEXT GROUP NEXT PAGE.
  ```

  then the following body group

  ```
  01 DETAIL-1 TYPE DE LINE PLUS 5.
  ```

  would be printed on value of FIRST DETAIL (in PAGE clause).

- If the following body group was the last one to be printed on a page

  ```
  01 TYPE CF MONTH NEXT GROUP 12.
  ```

  and after it was printed the value of LINE-COUNTER was 40, then the body group

  ```
  01 DETAIL-1 TYPE DETAIL LINE PLUS 5.
  ```

  would be printed on line 12 + 1 (i.e., line 13) on the next page.

12.9.11 Report Writer Routines

At the end of the analysis of a report description entry (RD), the Report Writer routines are generated, according to the contents of the RD. Each routine refers to the contents of the compiler-generated internal line number of its own respective RD.

12.10 Table Handling

12.10.1 Subscripts

If a subscript is a constant, the location of the subscripted data item within the table is resolved at compilation time.

If a subscript is held in a data item the location is resolved at execution time. The value contained in a data item used as a subscript is an integer that represents an occurrence number within a table. Every time a subscripted data item is referred to in a program, the compiler generates several instructions to calculate the correct displacement. Therefore, subscripts should be used with care to avoid an inefficient object program. See Chapter 8 for details. However, the compiler does optimize the calculation of displacements. If a subscripted data item is referred to more than once in the same statement, the displacement is calculated once only and is used each time the data item is referred to in that statement.

12.10.2 The SET Statement

The SET statement is used to assign values to index data items and index-names.

The SET statement can assign to an index-name the value of a literal, an identifier or an index-name from another table element. When this occurs, the index-name is set to an actual displacement from the start of the table element that corresponds with an occurrence number indicated by the second operand in the SET statement. The compiler performs all the required calculations. If the SET statement is used to assign an index-name to another index-name for the same table element, the compiler does not have to calculate the actual displacement value contained in the second operand.

However, when an index data item is set to another index data item or to an index-name, or when an index-name is set to an index data item, the compiler cannot change any existing displacement value because an index data item is not part of any table. Therefore, no conversion of values can be done. If the programmer forgets this, programming errors can occur.

For example, suppose that a table has been defined as:

```
01 A.
   02 B OCCURS 2 INDEXED BY A1, A5.
      03 C OCCURS 2 INDEXED BY A2, A6.
         04 D OCCURS 3 INDEXED BY A3, A4.
            05 E  PIC X(20).
            05 F  PIC 9(5).
```

```
                                                 Byte No.
                                 |---------|-|     0
                      { D(1,1,1)  |E        |F|
                      {           |---------|-|    25
          { C(1,1)    { D(1,1,2)  |E        |F|
          {           {           |---------|-|    50
          {           { D(1,1,3)  |E        |F|
  { B(1)  {                       |---------|-|    75
  {       {           { D(1,2,1)  |E        |F|
  {       {           {           |---------|-|   100
  {       { C(1,2)    { D(1,2,2)  |E        |F|
  {                   {           |---------|-|   125
  {                   { D(1,2,3)  |E        |F|
A {                               |---------|-|   150
  {                   { D(2,1,1)  |E        |F|
  {                   {           |---------|-|   175
  {       { C(2,1)    { D(2,1,2)  |E        |F|
  {       {           {           |---------|-|   200
  {       {           { D(2,1,3)  |E        |F|
  { B(2)  {                       |---------|-|   225
          {           { D(2,2,1)  |E        |F|
          {           {           |---------|-|   250
          { C(2,2)    { D(2,2,2)  |E        |F|
                      {           |---------|-|   275
                      { D(2,2,3)  |E        |F|
                                  |---------|-|   300
```

Sample Table Layout in Memory

The figure "Sample Table Layout in Memory" shows how the table is laid out in main memory. Suppose it is necessary to reference D (2, 2, 3). The following steps would be incorrect:

```
SET A3 TO 2.
SET INDX-DATA_ITM TO A3.
SET A2, A1 TO INDX-DATA-ITM.
SET A3 UP BY 1.
MOVE D (A1, A2, A3) TO WORKAREA.
```

The value contained in A3 following the first SET statement is 25, which represents the starting point (in bytes) of the second occurrence of D. When the second SET statement is obeyed, the value 25 is stored in INDX-DATA-ITM, and the third SET statement stores the value 25 in A2 and A1. The fourth SET statement augments the value in A3 to 50. The calculation of the address of D (A1, A2, A3) would then be as follows:

```
(address of D (1, 1, 1))+25+25+50 = (address of D (1, 1, 1))+100
```

where D (1, 1, 1) represents the first occurrence of D. This is not the address of D (2, 2, 3).

The following steps will determine the correct address:

```
SET A3 TO 2.
SET A2,  A1 TO A3.
SET A3 UP BY 1.
```

In this case the first SET statement stores the value 25 in A3. Since the compiler can calculate the lengths of B and C, the second SET statement stores the value 75 in A2 and the value 150 in A1. The third SET statement stores the value 50 in A3. The correct address calculation will be:

```
(address of D (1, 1, 1))+150+75+50 = (address of D (1, 1, 1))+275
```

The rules for the SET statement are shown below.

Receiving	Sending		
	Index-name	Index Data Item	Identifier or literal
Index-name	Set to value corresponding to occurrence number (note A)	Move without conversion	Set to value corresponding to occurrence number
Index Data Item	Move without conversion	Move without conversion	Not applicable
Identifier	Set to occurrence number represented by index-name	Not applicable	Not applicable
Note A: If the index-names refer to the same table element, the move is made without conversion.			

Rules for the SET Statement

12.10.3 The SEARCH Statement

Only one level of a table (a table element) can be referenced in one SEARCH statement. Note that SEARCH statements can be nested: an imperative statement must follow the WHEN condition and the SEARCH statement, when terminated by the END-SEARCH scope terminator, is an imperative statement.

The SEARCH statement has two formats.

Format 1

Format 1 SEARCH statements carry out a serial search of a table element. If the programmer knows that the "found" condition will occur after some intermediate point in the table element, to speed up execution, the SET statement can be used to set the index-names at that point and search only part of the table element. If the table element is large and must be searched from the first occurrence to the last, the use of Format 2 (SEARCH ALL) is more efficient than Format 1, as it uses a binary search technique; however the table must then be ordered.

In Format 1 the VARYING phrase allows the programmer to:

- Vary an index-name other than the index-name stated for this table element. So, with two SEARCH statements each using a different index-name, reference can be made to more than one value in the same table element for comparisons etc.
- Vary an index-name from another table element. In this case, the first index-name specified for this table element is used for the search and the index-name specified in the VARYING phrase is incremented at the same time. Thus it is possible to step through two table elements at once.

In Format 1, the WHEN condition can be any relation condition and can be multiple. If multiple WHEN conditions are specified, the implied logical connective is OR. That is, if any one of the WHEN conditions is satisfied, the imperative statement following the WHEN condition is executed. If it is necessary that all conditions of the SEARCH statement be satisfied, a compound WHEN condition with an AND logical connective must be used.

Format 2

In Format 2 (SEARCH ALL) the table must be ordered on the key(s) named in the OCCURS clause. Any key can be named in the WHEN condition, but all preceding names in the KEY phrase must also be tested. The test must be an "equal to" (=) condition and the KEY data-name must either be the subject or the object of the condition, or the name of a conditional variable with which the tested condition-name is associated. The WHEN condition can also be a compound condition, consisting of one of the simple conditions listed above, with AND as the only logical connective. The key and its object of comparison must be compatible.

To write a series of statements that will search the three dimensional table discussed under "The SET Statement" above, the programmer could write the following:

```
01 ERROR-INDICATOR1 PIC 1 VALUE B"1".
88 ERROR1                 VALUE B"1".
77 COMPARAND1          PIC X(5).
77 COMPARAND2          PIC 9(5).
01 A.
   05 B OCCURS 2 INDEXED BY A1 A5.
      10 C OCCURS 2 INDEXED BY A2 A6.
         15 D OCCURS 3 INDEXED BY A3 A4.
            20 E       PIC X(5).
            20 F       PIC 9(5).
              .
(set-up values for COMPARAND1 and COMPARAND2)
              .
   PERFORM VARYING A1 FROM 1 BY 1 UNTIL A1 > 2
     PERFORM VARYING A2 FROM 1 BY 1 UNTIL A2 > 2
      SET A3 TO 1
      SEARCH D WHEN E (A1, A2, A3) = COMPARAND1
                AND F (A1, A2, A3) = COMPARAND2
                SET A5 TO A1
                SET A6 TO A2
                SET A2 TO 3
                SET A1 TO 3
                SET ERROR1 TO TRUE
      END-SEARCH
     END-PERFORM
   END-PERFORM.
   IF NOT ERROR1 THEN GO TO ENTRY-PROCESSING1.
ERROR-RECOVERY1.
              .
ENTRY-PROCESSING1.
   MOVE E (A5, A6, A3) TO OUT-AREA1.
   MOVE F (A5, A6, A3) TO OUT-AREA2.
              .
```

The PERFORM statements vary the indexes (A1 and A2) associated with table elements B and C. The SEARCH statement varies A3, which is associated with table element D.

The values of A1 and A2 that satisfy the WHEN conditions of the SEARCH statement are stored in A5 and A6. A1 and A2 are then set to 3 via the SET statement, so that when returning from the SEARCH statement control will fall through the PERFORM statement to the GO TO statement.

Later references to the desired occurrence of table elements E and F use the index-names A5 and A6 in which the correct value was stored.

For example, suppose that the following table was defined:

```
01 TABLEA.
   05 ENTRY-IN-TABLEE OCCURS 90 TIMES
      ASCENDING KEY1, KEY2
      DESCENDING KEY3
      INDEXED BY INDEX-A.
      10  PART-1  PIC 99.
      10  KEY-1   PIC 9(5).
      10  PART-2  PIC 9(6).
      10  KEY-2   PIC 9(4).
      10  PART-3  PIC 9(33).
      10  KEY-3   PIC 9(5).
```

A search of the entire table could be made with the following:

```
SEARCH ALL ENTRY-IN-TABLEE AT END GO TO NOFIND
       WHEN KEY-1   (INDEX-A)  =  VALUE-1
        AND KEY-2   (INDEX-A)  =  VALUE-2
        AND KEY-3   (INDEX-A)  =  VALUE-3
        MOVE PART-1 (INDEX-A) TO OUTPUT-AREA.
```

These instructions will result in a search on the above table TABLEA which contains 90 elements of 55 bytes and 3 keys. The primary and secondary keys (KEY-1 and KEY-2) are in ascending order but the least significant key (KEY-3) is in descending order. If an entry is found in which the three keys are equal to the given values (VALUE-1, VALUE-2, VALUE-3) PART-1 of that entry will be moved to OUTPUT-AREA. If no matching key is found in any of the entries in TABLEA, the NOFIND routine is entered.

If there is a match between a table entry and the given values, the index (INDEX-A) is set to a value indicating the relative position within the table of the matching entry. If a match is not found, the final value of the index is unpredictable.

Note that if KEY entries within the table do not contain valid values, the results of the binary search will be unpredictable.

12.10.4 Building Tables

When reading in data to build an internal table the following points should be borne in mind.

- Ensure that the data does not exceed the space allocated for the table.
- If the data must be in sequence, check the sequence in the program.
- If the data contains a subscript determining its position in the table, check that the subscript does not exceed the bounds of the table.

When testing for the end of a table, use a data item containing the item count, rather than use a literal. Then, if the table must be expanded, only one value need be changed, instead of all references to the literal (in addition to changing the number of occurrences in the OCCURS clause). Both changes can be effected using the REPLACE statement.

12.11 Intermediate Results

The compiler breaks down arithmetic statements into a succession of simpler operations and reserves locations in memory to contain the results of these operations. The handling of these "intermediate results" is discussed in the following paragraphs.

For an arithmetic statement containing only one pair of operands, no intermediate result is generated. Intermediate results may be generated in the following cases.

- In an ADD or SUBTRACT statement which contains several operands immediately following the verb.
- In a COMPUTE statement which specifies a series of arithmetic operations.
- In arithmetic expressions which are contained in IF or PERFORM statements.

In such cases, the compiler treats the statement as a series of operations. For example, the following statement:

```
COMPUTE Y = A + B * C - D / E + F ** G
```

is replaced by:

```
**F            BY G       GIVING ir1
MULTIPLY B     BY C       GIVING ir2
DIVIDE E       INTO D      GIVING ir3
ADD A          TO ir2      GIVING ir4
SUBTRACT ir3   FROM ir4    GIVING ir5
ADD ir5        TO ir1      GIVING Y
```

Where ir1 through ir5 are successive intermediate results.

In the following discussion "decimal floating-point format" is referred to. In this format, 18 most significant digits are retained. If the LEVEL=NSTD parameter is included in the CBL JCL statement, the TEMP=parameter is not included in the CBL JCL statement, and the TEMP clause is not used in the Default Section of the Control Division, 30 most significant digits are retained. If the TEMP=parameter is included in the CBL JCL statement it may specify a different number of significant digits. If the TEMP=parameter is not included in the CBL JCL statement, but the TEMP clause is used in the Default Section of the CONTROL DIVISION, it may specify the number of significant digits.

12.11.1 Length of Intermediate Result Fields

Based upon the length of the operands or intermediate results to be operated upon, the compiler allocates intermediate result fields of a particular length. The algorithm for doing this is explained below. The following abbreviations are used in this explanation.

ip	the number of integer places to be stored in the intermediate result.
dp	the number of decimal places to be stored in the intermediate result.
dmax	either, the maximum number of decimal places defined for any operand, or, the number of decimal places needed for the final result field (plus 1 if rounding is required), whichever is larger in a particular statement.
op1	the first operand in a generated arithmetic statement.
op2	the second operand in a generated arithmetic statement.
d1,d2	the number of decimal places specified for op1 and op2.
i1,i2	the number of integer places specified for op1 and op2.
ir	the intermediate result produced by an arithmetic operation. ir1, ir2 etc. represent successive intermediate results.

The compiler calculates the number of integer places in an "ir" in the following way. The maximum value that an "ir" can contain is determined by performing the statement in which the "ir" occurs:

- If an operand in the statement is a data-name, the value used for this operand is the largest value that can be stored in the data item. For example, PIC 9V99 would result in a value 9.99.
- If an operand is a literal the actual value of the literal is used.
- If an operand is an intermediate result, the value determined for the intermediate result in a previous calculation is used.
- If the operation is division:
    a. If op2 is a data-name, the value used for op2 is the smallest non-zero value that can be stored in the data item. For example, PIC 9V99 would result in a value of 0.01.
    b. If op2 is an intermediate result, the smallest non-zero value that can be stored in the intermediate result field is used.
    c. If a further divide, multiply or exponentiation is to be performed for the same COBOL statement and either the TEMP= parameter of the CBL JCL statement does not specify NS, or it is not specified and the TEMP IS NOT STANDARD clause is not used in the Default Section of the Control Division, decimal floating-point format will be used.
- If the operation is exponentiation and op2 has a literal value of 2 or 3 normal multiplication will be performed. Otherwise decimal floating-point format will be used.

When the maximum value of an "ir" is determined in the above manner, "ip" is set equal to the number of integers in this value.

The compiler calculates the number of decimal places in an "ir" in the following way:

```
Operation       Decimal Places

+ or            d1 or d2, whichever is greater.
*               d1 + d2
/               (d1-d2) or (d1+i2) or dmax, whichever is greater.
```

The table below indicates the length allocated to "ir" based upon the values calculated for "ip" and "dp".

Length of the Intermediate Result Fields

Value of	Not Standard TEMP specified	Value of ip + dmax	Length allocated for ir
< 32	Not relevant	Any value	Ip integer places and dp decimal places are allocated for ir.
> 31	Used	< 32	Ip integer places and 31-ip decimal places are allocated for ir.
	Used	> 31	Decimal floating-point format is used.
	Not used	Any value	

12.11.2 Fixed Binary Data Items

If an operation involving fixed binary operands requires an intermediate result greater than the equivalent of 10 decimal digits, the operands are converted into packed decimal before performing the operation. If the result field is fixed binary, the result will then be converted from packed decimal into binary.

If an intermediate result will not be greater than the equivalent of 9 decimal digits, the operation will be performed most efficiently on fixed binary data fields.

12.11.3 COBOL Run-Time Package

12.11.3.1 Arithmetic Intermediate Results

If a decimal multiplication requires an intermediate result greater than 31 decimal digits, a COBOL run-time package procedure is used to perform the calculation in decimal floating point format.

A COBOL run-time package procedure will be used to perform division if the number of decimal places of the dividend plus the number of decimal places of the quotient plus the number of integer places of the quotient is greater than 30.

If an arithmetic operation requires an intermediate result greater than 31 decimal digits, decimal floating-point format will be used for the operation. The number of digits in this intermediate result is given by the TEMP IS clause of the Control Division (the Control Division is not part of the ANS standard) the default value is 30 if LEVEL = NSTD is specified in the CBL JCL statement, otherwise the default value is 18.

12.11.3.2 H_CBL_UGETG4

Please refer to "Return Code", chapter 9.

12.11.3.3 H_CBL_USETST

Please refer to "User JCL Status", below.

12.11.3.4 H_CBL_UGETPN

This can be used in the Procedure Division of an externally compiled program to return the program's name (as specified in the PROGRAM-ID paragraph). To do this, use the following statement:

```
CALL "H_CBL_UGETPN" USING data-name-1
```

There must be only one parameter. This parameter (data-name-1) must be an alphanumeric data item of 30 characters. Data-name-1 is set to the name of the program.

EXAMPLE:

```
IDENTIFICATION DIVISION.
PROGRAM-ID. THIS-PROGRAM-NAME.
...
DATA DIVISION.
...
01 CURRENT-PROG-NAME PIC X(30).
...
PROCEDURE DIVISION.
...
    CALL "H_CBL_UGETPN" USING CURRENT-PROG-NAME.
...
```

The result of the above CALL statement is exactly the same as if one had entered the following:

```
MOVE "THIS-PROGRAM-NAME" TO CURRENT-PROG-NAME
```

NOTE:

H_CBL_UGETPN may be called from a contained program. In this case, the name returned is that of the externally compiled container program, not that of the contained program. See the example below:

❑

EXAMPLE:

```
IDENTIFICATION DIVISION.            Outermost Container Program (ABC)
PROGRAM-ID. ABC.
...
CALL "XYZ"
...
IDENTIFICATION DIVISION.                        Contained Program (XYZ)
PROGRAM-ID. XYZ.
...
CALL "H_CBL_UGETPN" USING CURRENT-PROG-NAME.
...
END PROGRAM XYZ.
...
END PROGRAM ABC.
```

The result in CURRENT-PROG-NAME is "ABC" not "XYZ".

❑

12.11.4 The ON SIZE ERROR Phrase

Apart from division by zero or exponent overflow in floating point format, the ON SIZE ERROR phrase applies only to final results and not to intermediate results, i.e., it applies only when the final results are stored in the receiving data items.

12.11.5 Communication Programs

Communications programming is not discussed in this manual. This subject is covered in the *Communications Processing Facility Manual*, which includes a discussion of the following Message Control System verbs.

- SEND
- RECEIVE
- ACCEPT
- DISABLE
- ENABLE
- PURGE

12.12 INSPECT and EXAMINE

The INSPECT statement has been added to the COBOL language standard to replace the EXAMINE statement. As EXAMINE has been removed from the ANS standard, it is advisable to use INSPECT rather than EXAMINE in all new programs.

The main advantages of INSPECT are as follows:

- Groups of characters can be tallied and/or replaced by a single INSPECT statement (the EXAMINE statement can only tally and/or replace a single character).
- Several different groups of characters can be tallied and/or replaced in a single INSPECT statement.
- INSPECT can tally and/or replace groups of characters before or after a specified group of characters.

Examples of EXAMINE statements and an equivalent INSPECT statement are shown below.

Comparison of INSPECT and EXAMINE

INSPECT or EXAMINE Statement	Value of X		Value of TALLY	Value of UPTOA	Value of ONES
	Before	***After***			
EXAMINE X TALLYING UNTIL FIRST A. *EXAMINE X TALLYING ALL 1.* *EXAMINE X REPLACING FIRST B BY C.* *EXAMINE X REPLACING LEADING 2 BY 3*	*21BA2AB* *21BA2AB* *21BA2AB* *21CA2AB*	*Same* *Same* *21CA2AB* *31CA2AB*	*3* *1* *same* *same*		
INSPECT X TALLAYING UPTOA FOR. *CHARACTERS BEFORE INITIAL A,* *ONES for all 1,* *REPLACING FIRST B BY C,* *LEADING 2 BY 3*	*21BA2AB*	*31CA2AB*	-	*3*	*1*

Further examples of the use of the INSPECT statement are given in the *COBOL 85 Reference Manual*.

13. COBOL 85 Extensions and Changes

This chapter deals with the major aspects of the changes due to the introduction of the COBOL 85 Standard and to the GCOS 7 extensions to this Standard. Some of these changes are new features and some are modifications.

An exhaustive list of these changes is given in the *COBOL 85 Reference Manual*. Also note that the Reference Manual contains a list of the COBOL 85 obsolete elements, which means elements that will be deleted from the next revision of the COBOL Standard and therefore are preferably not to be used when developing new programs.

13.1 Structured Programming

13.1.1 What is Structured Programming?

Programming has always been a difficult task. The most difficult thing about it is often to maintain programs that were written by someone else.

Developing programs in a structured way gives them better legibility and maintainability. It also speeds up development time and optimizes execution.

The main concepts of structured programming are: modularity, use of simple nested structures that closely follow flowcharts, avoiding as much as possible transfers of control by GO TO statements. (The intensive use of GO TO statements quickly leads to difficulties in following the internal logic of the program and increased risk of bad segmentation, and thus time loss.)

It has been demonstrated for instance that structured programming increases the debugging-speed of new programs by a factor of 3 or 4.

It is sometimes difficult to follow a method, if the programmer sees this as being too mechanical. But this is not true; following rules that give a program the simplest possible structure liberates one from struggling with a language to find some clever way to solve a problem. It gives the programmer a means to concentrate on the real question rather than on the way he/she can code it.

Structured programming is based on the notion of "clean programs": easy to read, easy to understand, easy to maintain. Programs may be split into simple modules composed of simple structures: linear, selective (IF THEN ELSE), or iterative (loop with test before or after entering it).

The following very simple example will give an idea of the philosophy of structuring:

```
  IF A < B COMPUTE C=D-1.
           GO TO FOLLOW-1.
  IF A = B COMPUTE C=D-2
           GO TO FOLLOW-1.
  COMPUTE C=D-3.
FOLLOW-1.
```

Could be structured this way:

```
IF A < B THEN COMPUTE C=D-1
          ELSE IF A = B THEN COMPUTE C=D-2
                   ELSE COMPUTE C=D-3.
```

Where nested IF statements are used, together with the new COBOL 85 THEN statement which is optional but gives a better legibility.

New features have been added to the COBOL language to facilitate structured programming, mainly:

- scope terminators such as END-ADD, END-IF,...;
- NOT-conditions such as NOTAT END, NOT ON SIZE ERROR, ...;
- THE CONTINUE statement, meaning no operation;
- the new in-line PERFORM statement: PERFORM without procedure-name i.e. PERFORM imperative statements END-PERFORM;
- the EVALUATE statement.

Languages such as PASCAL and C, which are relatively new compared to COBOL, were from the beginning developed as structured languages. COBOL 85 gives the same facilities (apart from the recursivity), while keeping all the COBOL advantages of a "Business Oriented Language" and ensuring compatibility with old programs.

The following paragraphs give some hints on how to use the new COBOL 85 features to structure programs.

13.1.2 The NOT-Conditions and Scope Terminators

The COBOL 85 Standard introduces the following optional phrases:

```
NOT INVALID KEY
NOT AT END
NOT ON SIZE ERROR
NOT ON EXCEPTION
NOT ON OVERFLOW
NOT END-OF-PAGE
```

followed by imperative statements to be executed if the NOT-condition is true.

EXAMPLE:

```
ADD  identifier-1 TO identifier-2
     ON SIZE ERROR imperative-statement-1
     NOT ON SIZE ERROR imperative-statement-2
```

Scope terminators are END-verb statements that explicitly terminate the scope of an instruction. Scope terminators exist for the following verbs:

```
ADD           READ
CALL          RECEIVE
COMPUTE       RETURN
DELETE        REWRITE
DIVIDE        SEARCH
EVALUATE      START
IF            STRING
MULTIPLY      SUBTRACT
PERFORM       UNSTRING
              WRITE
```

The above example could contain an END-ADD statement. Another example:

```
DIVIDE A BY B GIVING C
       ON SIZE ERROR DISPLAY "SIZE ERROR ON DIVIDE"
       MULTIPLY A BY -1 GIVING C
         ON SIZE ERROR DISPLAY "SIZE ERROR ON MULTIPLY"
         END-MULTIPLY
       ADD 1 TO ERROR-FLAG
       END-DIVIDE.
```

In this example, scope terminators enhance the program's legibility. Indenting source-program lines is also a good habit. It allows statement nesting and clearly shows the levels of nesting.

❑

13.1.3 The CONTINUE Statement

The CONTINUE statement is a "no operation" statement. It only gives control to the next executable statement. It may be used to clarify the structure of a portion of program.

13.1.4 The COBOL 85 PERFORM Statement

In COBOL 85, a PERFORM statement need not refer to an external procedure known by "procedure-name". A PERFORM ... END-PERFORM delimits an internal procedure, possibly iterative if UNTIL is used.

EXAMPLE:

```
MOVE 1 TO S, I
PERFORM VARYING I FROM 1 BY 1 UNTIL I = 10000
        COMPUTE S = S + 1 / I
        END-PERFORM
```

Another enhancement brought to PERFORM structures is the PERFORM TEST BEFORE...UNTIL... or PERFORM TEST AFTER...UNTIL... . For those who are familiar to the PASCAL language, this is similar to the DO UNTIL or DO WHILE statements.

The TEST BEFORE option means that the first test will be done before the execution of the statements contained in the scope of the PERFORM, as in the following example:

```
PERFORM WITH TEST BEFORE
        VARYING I FROM 1 BY 1
        UNTIL TAB (I) = SPACE OR I = 100
        WRITE OUTREC FROM TAB (I)
        END-PERFORM
```

If the first TAB (I) contains SPACE, control is immediately given to the sentence that follows the END-PERFORM. No record is written.

If TEST AFTER had been used in place of TEST BEFORE, a first PERFORM loop would have been executed before the test TAB (I) = SPACE and a first record would have been written to the output file.

Also note that the number of AFTER...BY...UNTIL phrases allowed in the VARYING phrase of a PERFORM statement is unlimited in COBOL 85 whereas it was limited to two in COBOL 74.

❑

The EVALUATE Statement

The COBOL 85 EVALUATE statement is the equivalent of the PASCAL CASE OF statement. It gives the decision table construct to COBOL.

A few simple examples will show its use.

EXAMPLE 1:

```
EVALUATE ANSWER
       WHEN "y"
       WHEN "Y" DISPLAY "YOU ARE RIGHT"
       WHEN "n"
       WHEN "N" DISPLAY "WHY NOT?"
       WHEN OTHER DISPLAY "PLEASE ANSWER Y OR N"
       END-EVALUATE
```

❑

EXAMPLE 2:

```
EVALUATE    SCORE
       WHEN 15 THRU 20 MOVE "EXCELLENT" TO APPRECIATION
       WHEN 10 THRU 14 MOVE "GOOD" TO APPRECIATION
       WHEN 5 THRU 9 MOVE "MUST PROGRESS" TO APPRECIATION
       WHEN 0 THRU 4 MOVE "BAD RESULT" TO APPRECIATION
       WHEN OTHER MOVE "*****" TO APPRECIATION
       END-EVALUATE
```

❑

EXAMPLE 3:

```
EVALUATE    RAINY ALSO WARM
       WHEN TRUE ALSO FALSE DISPLAY "BAD WEATHER"
       .
       .
       .
       END-EVALUATE
```

❑

In the second example, the THRU (equivalent to THROUGH) clauses define ranges for evaluation.

In the third example, the identifiers RAINY and WARM may take the values TRUE and FALSE. They are connected by an ALSO clause. If RAINY contains TRUE and WARM contains FALSE, "BAD WEATHER" will be displayed.

The EVALUATE statement permits simpler structures than multiple-nested IF statements, which could be used for the same evaluation.

Note that GCOS 7 COBOL 85 allows Boolean expressions to be used within the scope of EVALUATE statements (not Standard).

13.2 Inter-Program Communication

13.2.1 COBOL 85 New Features

COBOL 85 introduces several new inter-program communication capabilities, including internal sub-programs (contained programs), external and global data, and passing parameters by reference or by content.

New statements have been added to the COBOL language, such as: END PROGRAM, BY CONTENT, BY REFERENCE, COMMON, INITIAL, GLOBAL and EXTERNAL.

Note that in COBOL 85, the parameters passed in a CALL USING need not be 01 or 77 level data items.

13.2.2 Contained Programs

A contained program is a program that is contained within another COBOL program. It is delimited by IDENTIFICATION DIVISION and END PROGRAM headers.

EXAMPLE:

```
IDENTIFICATION DIVISION.
PROGRAM-ID. OUTER.
.
.
.
PROCEDURE DIVISION.
.
.
.
IDENTIFICATION DIVISION.
PROGRAM-ID. INNER.
.
.
.
END PROGRAM INNER.
.
.
.
END PROGRAM OUTER.
```

Contained programs may be nested.

GCOS 7 COBOL 85 allows contained programs anywhere a COBOL statement is allowed in the Procedure Division. It also allows the word SAME to be used as the segment-number in procedure-name headers of contained programs, to specify the same segment-number as in the section which contains it (extensions to the Standard).

The following example illustrates these extensions:

```
.
.
.
SECTION 15.
   IF A > B ...
      CALL INNER
IDENTIFICATION DIVISION.
PROGRAM-ID. INNER.
.
.
.
SECTION SAME.
.
.
.
END PROGRAM INNER.
      ELSE...
      END-IF.
.
.
.
```

The program INNER is contained within the scope of an IF statement (allowed in GCOS 7 COBOL 85, not in COBOL 85 Standard). The CALL statement is necessary because the program INNER, as any subprogram, must be called to execute. The extension that allows contained programs anywhere a COBOL statement is permitted is useful in conjunction with a COPY statement: it enables a program to be copied together with the statement which calls it.

The segment-number SAME means here that the actual segment-number will be 15 (segment-number of the section containing the program).

❑

13.2.3 Call by Reference and Call by Content

In COBOL 85, parameters may be passed in CALL USING phrases either by reference, which is the same as in COBOL 74 (default) or by content, in which case the parameters cannot be modified by the called program (any modification will not be returned to the calling program).

EXAMPLE:

```
IDENTIFICATION DIVISION.
PROGRAM-ID. OUTER.
DATA DIVISION.
WORKING-STORAGE SECTION.
01 COINS.
   02 QUARTERS     PIC 99 VALUE 10.
   02 DIMES        PIC 99 VALUE 8.
01 TOTAL           PIC 999V99.
PROCEDURE DIVISION.
MAIN-OUTER.
   CALL "INNER" USING BY CONTENT QUARTERS, DIMES,
                      BY REFERENCE TOTAL.
   DISPLAY "QUARTERS: ", QUARTERS, " DIMES: ",DIMES.
   DISPLAY "TOTAL: ", TOTAL.
   STOP RUN.
IDENTIFICATION DIVISION.
PROGRAM-ID. INNER.
DATA DIVISION.
LINKAGE SECTION.
01 QUARTERS        PIC 99.
01 DIMES           PIC 99.
01 TOTAL           PIC 999V99.
PROCEDURE DIVISION USING QUARTERS, DIMES, TOTAL.
MAIN-INNER.
   DISPLAY "ENTERING INNER".
   COMPUTE TOTAL = QUARTERS * .25 + DIMES *.10.
   MOVE ZEROS TO QUARTERS, DIMES.
   DISPLAY "QUARTERS: ", QUARTERS, " DIMES: ",DIMES.
   DISPLAY "TOTAL: ", TOTAL.
   DISPLAY "EXITING FROM INNER".
   EXIT PROGRAM.
END PROGRAM INNER.
END PROGRAM OUTER.
```

In the above example, QUARTERS, DIMES, and TOTAL are all modified in the called "INNER" program. But as QUARTERS and DIMES are called by content, their values are not modified in the "OUTER" program.

Thus, the following lines will be displayed:

```
ENTERING INNER
QUARTERS: 00 DIMES: 00
TOTAL: 003.30
EXITING FROM INNER
QUARTERS: 10 DIMES: 08
TOTAL: 003.30
```

Note that GCOS 7 COBOL 85 allows expressions or literals in CALL BY CONTENT when calling a contained program (extension to the Standard).

❑

COMMON Clause

Adding IS COMMON to the name of a contained program in its PROGRAM-ID entry extends its visibility to any other program contained in the same main program.

EXAMPLE:

```
IDENTIFICATION DIVISION.
PROGRAM-ID. A.
PROCEDURE DIVISION.
A-LABEL.
    CALL "B".
    CALL "C".
    STOP RUN.

IDENTIFICATION DIVISION.
PROGRAM-ID. B.
PROCEDURE DIVISION.
B-LABEL.
    .
    .
    CALL "C".
END PROGRAM B.

IDENTIFICATION DIVISION.
PROGRAM-ID. C IS COMMON.
C-LABEL.
    .
    .
END PROGRAM C.
    .
    .
END PROGRAM A.
```

The main program A may call any program contained in it. As program C contains the COMMON clause, it may be called by any program contained in A. Thus, for instance, B and any program contained in B may call C. But C may not call B, as B is not common.

Note that recursive calls are not allowed. A program may not call itself nor call a program in which it is contained.

❑

13.2.4 INITIAL Clause

Adding IS INITIAL to the name of a program in its PROGRAM-ID entry causes all data within it to be initialized on each entry to the procedure. This means that all data will contain the same initial values as when the program was first called in the run unit.

EXAMPLE:

```
IDENTIFICATION DIVISION.
PROGRAM-ID. MAIN.
    .
    .
    .
    CALL "A".
    .
    .
    .
    CALL "A".
    .
    .
    .
    STOP RUN.
IDENTIFICATION DIVISION.
PROGRAM-ID. A IS INITIAL.
DATA DIVISION.
WORKING-STORAGE SECTION.
01  QUARTERS    PIC 99 VALUE 10.
01  DIMES       PIC 99 VALUE 20.
01  TOTAL       PIC 999V99 VALUE 0.
PROCEDURE DIVISION.
MAIN-A.
    COMPUTE TOTAL=TOTAL+QUARTERS*.25+DIMES*.10.
    MOVE 1 TO QUARTERS DIMES.
    .
    .
    .
    EXIT PROGRAM.
END PROGRAM A.
    .
    .
    .
END PROGRAM MAIN.
```

In this example, although 1 is moved to QUARTERS and DIMES during the first execution of the contained program A, the second execution will give the same result, because the initial values are reset, A being declared INITIAL. If IS INITIAL had not been stated, the second call would have given another result.

Note that COMMON and INITIAL are not exclusive.

❑

13.2.5 EXTERNAL and GLOBAL Clauses

The EXTERNAL clause specifies that a data item or file defined in a given program is available to any other program contained in the same run-unit.

The GLOBAL clause specifies that a data item or file is available to every program contained in the one which declares it as global.

EXAMPLE:

```
IDENTIFICATION DIVISION.
PROGRAM-ID. A.
DATA DIVISION.
WORKING-STORAGE SECTION.
01 VAR-1    PIC  999 IS GLOBAL.
01 VAR-2    PIC 999 IS EXTERNAL.
01 VAR-3    PIC 999 IS GLOBAL, EXTERNAL.
PROCEDURE DIVISION.
A-LABEL.
    MOVE 100 TO VAR-1, VAR-2, VAR-3.
    CALL "B".
    .
    .
IDENTIFICATION DIVISION.
PROGRAM-ID. B.
PROCEDURE DIVISION.
B-LABEL.
    DISPLAY "VAR-1= ", VAR-1.
    DISPLAY "VAR-3= ", VAR-3.
    .
    .
    .
END PROGRAM B.
END PROGRAM A.
```

In this example, VAR-1 and VAR-3 are global and thus directly available to B. VAR-2 is external but not global and thus not visible to B.

❑

GCOS 7 COBOL 85 brings the following extensions to the standard:

- GLOBAL is allowed on paragraph-names and section-names of the Procedure Division. A whole paragraph or section may be global.
- GLOBAL is allowed on Communication Description entries and on Sort-Merge Description entries as well as on File Description entries.

13.3 Data Definition and Manipulation

13.3.1 SIGN Clause

The SIGN clause is allowed in Report Description entries.

Multiple SIGN clauses may be specified in the hierarchy of a data description entry. In this case, the specification at the subordinate level takes precedence over the specification at group level.

13.3.2 Table Handling

In COBOL 85, a table may have up to 7 dimensions. In GCOS 7 COBOL 85, this number is raised to 48 which means practically no limit.

Two new possibilities exist for table element referencing:

- Mixing subscripts and indexes. Indexes and data-name subscripts may both be written in a single set of subscripts used to reference an individual occurrence of a multi-dimensional table.
- Referencing elements by relative subscripting, such as TAB(N+2) or MAT(I-3). This gives a facility similar to relative indexing.

In COBOL 85, the SYNCHRONIZED clause may be specified for an elementary data item with USAGE IS INDEX.

The VALUE clause may be used in a data description entry that contains an OCCURS clause or is subordinate to an entry containing an OCCURS clause.

13.3.3 INSPECT Statement

The CONVERTING phrase has been added to the INSPECT statement. An INSPECT CONVERTING is equivalent to an INSPECT REPLACING with ALL being used for each character. (Examples are given in the *COBOL 85 Reference Manual*).

In COBOL 85, multiple occurrences of the BEFORE/AFTER phrase are allowed.

13.3.4 SET Statement

In COBOL 85, the SET statement may be used to change the value of a conditional variable.

In GCOS 7 COBOL 85, the SET statement may also be used to store a value in a pointer data item or to associate an address with a data item declared in the Linkage Section. (Refer to the *COBOL 85 Reference Manual*).

13.3.5 Reference Modification

Reference modification is a new method for referencing data items. A portion of a data item is referenced by its first position in the data item and by its length.

EXAMPLE:

```
.
.
.
DATA-NAME PIC X(120).
.
.
.
MOVE DATA-NAME(27:5) TO ...
```

Positions 27 through 31 will be moved.

Note that for the purpose of reference modification, the character colon (:) has been added to the COBOL character set.

❑

13.3.6 De-editing Numeric Data Items

A numeric edited data item may be moved to a numeric data item. In this case, de-editing takes place.

13.3.7 PICTURE Clause

The insertion characters "." or "," may be used as the last character of a PICTURE character string, provided it is immediately followed by the separator period which terminates the data description entry.

13.3.8 User-defined Words

A user may define words with the same name as system-names. The compiler uses the context to determine which type it is. This facility does not concern COBOL reserved words.

Symbolic-characters are user-defined words that define user-defined figurative constants.

A user-defined word need not be unique or capable of being made unique unless referenced.

Note that 50 levels of qualification are allowed in COBOL 85 whereas 5 was the COBOL 74 limit.

13.3.9 Figurative Constant ZERO

In COBOL 85, the figurative constant ZERO is allowed in arithmetic expressions.

13.3.10 Lower-case Letters and Non-numeric Literals

Lower-case letters may be used in character strings. Except when used in non-numeric literals, each lower-case letter is equivalent to the corresponding upper-case letter.

The ALPHABETIC test gives a true value for alphabetic upper and lower-case characters whereas the ALPHABETIC-UPPER and ALPHABETIC-LOWER tests give a true value respectively for alphabetic upper-case characters and alphabetic lower-case characters.

Non-numeric literals may have up to 160 characters in length whereas 120 was the limit in COBOL 74.

13.3.11 Support of New Data Types

The COBOL 85 Standard adds the BINARY and PACKED-DECIMAL usages.

GCOS 7 COBOL 85 also supports BIT, COMP-5 (DPS8 packed-decimal data with ASCII sign), COMP-15 (internal floating-point quadruple precision) and POINTER usages.

These are described in Chapter 5 of the present manual. Also refer to the *COBOL 85 Reference Manual*, description of the USAGE clause.

13.4 Input-Output Related Modifications

13.4.1 SELECT OPTIONAL Phrase

The SELECT OPTIONAL... phrase, which allows the selected file not to be present at run time, may be used in COBOL 85 with Indexed or Relative files as well as Sequential files.

13.4.2 Padding Character

The padding character can be requested for filling out a physical block when the valid records do not completely fill that block.

GCOS 7 COBOL 85 adds a NO PADDING clause in order to speed up reading of sequential files. This clause bypasses the checking of records filled with padding characters.

13.4.3 RECORD DELIMITER Clause

The RECORD DELIMITER clause in File-Control entry is used to specify the mechanism that will be used for determining the length of a variable-length record on a medium associated with a file. STANDARD-1 specifies a standard mechanism. IMPLIED, which is the default value, specifies that explicit or implicit organization qualifier contained in the ORGANIZATION clause be used.

13.4.4 REWRITE with Modified Length

A record of different length may replace another record if the file organization is indexed or relative.

13.4.5 SORT and MERGE

Multiple file-names are allowed in the GIVING phrase of a SORT or MERGE statement. This feature permits as many copies as desired of the sorted or merged file.

The input and output procedures of a SORT or MERGE statement may contain explicit transfers of control to points outside the input or output procedure. The remainder of the Procedure Division may contain transfers of control to points inside the input or output procedure.

13.4.6 New Input-Output Status Key Values

A number of new input-output status key values have been defined in the COBOL 85 Standard. Their purpose is to give a means of testing many I/O conditions that could not be tested under the COBOL 74 Standard.

These new status key values are described in the *COBOL 85 Reference Manual*. The user is recommended to be careful when recompiling existing programs that test status key values.

Note that GCOS 7 COBOL 85 allows in the Default Section a clause COBOL 1974 FOR FILES which enables the same values to be kept as in COBOL 74 for input-output status keys.

13.4.7 LINAGE Clause

Within the LINAGE clause, data-names may be qualified.

13.4.8 CLOSE of Files

The REEL/UNIT of the CLOSE statement can be applied to a single reel/unit file and is specifically permitted for a report file produced by the Report Writer.

The FOR REMOVAL phrase of the CLOSE statement is allowed for a sequential reel/unit file.

The NO REWIND phrase cannot be specified in a CLOSE statement having the REEL/UNIT phrase.

The CANCEL statement as well as the STOP RUN statement close all open files.

13.4.9 GCOS 7 Input/Output Facilities

The following features are GCOS 7 COBOL 85 facilities:

- dynamic file assignment;
- dynamic assignment of queued file members;
- ability to scan members from a queued file.

These facilities are described in the Chapter 9 of the present manual.

13.4.10 Enforced Checks on Indexed Files

COBOL 85, as well as COBOL 74, requires that all the keys of an indexed file are specified in each program using the file. The check that these keys are specified is enforced in COBOL 85 compiler.

13.5 Communications Facilities

13.5.1 The PURGE Statement

The PURGE statement is a new COBOL 85 statement that causes the Message Control System to eliminate any partial message that has been released by one or more SEND statements.

13.5.2 New Communication Status and Error Keys

New communication and error key values have been added so that the user can check for these values and take corrective action, if appropriate.

These values are described in the *COBOL 85 Reference Manual*.

13.6 Syntax and Miscellaneous Features

13.6.1 Features Made Optional

The following words or clauses have been made optional:

- ORGANIZATION IS in the File-Control entry;
- ACCESS MODE IS in the File-Control entry;
- LABEL RECORDS clause;
- word RECORD in DATA RECORDS and LABEL RECORD clauses (GCOS 7 extension);
- DATA DIVISION and PROCEDURE DIVISION.

13.6.2 Order of Clauses

The order of clauses has been made immaterial in the I-O-CONTROL paragraph and in the Communication Description entry.

13.6.3 REPLACE Statement

The REPLACE statement is a new COBOL 85 statement that replaces program text at compile time.

It may be very useful when some of the new COBOL 85 reserved words were user-defined words in COBOL 74 programs that are to be recompiled.

EXAMPLE:

```
REPLACE ==OTHER== BY ==OT-1==.
```

The scope of a REPLACE statement may be terminated by a REPLACE OFF statement.

❑

13.6.4 DAY-OF-WEEK Phrase of the ACCEPT Statement

The DAY-OF-WEEK phrase provides access to an integer representing the day of week: 1 represents Monday, 2 represents Tuesday,..., 7 represents Sunday.

13.6.5 EXIT PROGRAM Statement

The EXIT PROGRAM statement need not be the only statement in a paragraph; but if it appears in a consecutive sequence of imperative statements within a sentence, it must appear as the last statement in that sequence.

13.6.6 GO TO procedure-name DEPENDING ON Statement

The number of procedure-names required in GO TO ... DEPENDING ON statement has been reduced to one.

13.6.7 Default Legible Equivalent

The current COBOL 85 language allows the representation of numeric data items under four different formats on the external medium from which they are ACCEPTed or upon which they are DISPLAYed. These formats are called legible equivalents.

The legible equivalent is specified through the suffix -0, -1, -2, or -X appended to the device name referenced in an ACCEPT or DISPLAY statement. When no suffix is explicitly specified, the default is -1 for SYSIN or SYSOUT and -2 in other cases. The corresponding rules are different from those of the COBOL 74 compiler.

To get the same results as with the COBOL 74 compiler, the suffix -0 must be used. This is done by specifying -0 either in the definition of a mnemonic-name or in a Control Division Default Section clause.

EXAMPLE:

```
DEFAULT SECTION.
    ACCEPT TERMINAL IS TERMINAL-0
    SYSOUT IS SYSOUT-0
    .
    .
    .
SPECIAL-NAMES.
    SYSIN-0 IS DEVICE-1
    TERMINAL-0 IS DEVICE-2
    .
    .
    .
    ACCEPT ... FROM DEVICE-1.
    ACCEPT ... FROM DEVICE-2.
    DISPLAY IDENTIFIER-1.
    DISPLAY ... UPON DEVICE-2.
```

❑

13.6.8 GCOS 7 Compiler Options

The new GCOS 7 compiler options are described in the Chapter 2 of the present manual.

Note that:

- external line numbers may now be shown in the compilation listing;
- warnings and observations diagnostics may be placed before or after the source listing;
- cross-reference listings may be restricted to those names which are actually referenced.

The COVLIB parameter invokes the test coverage facility. It is described in the following section.

13.7 The Test Coverage Facility

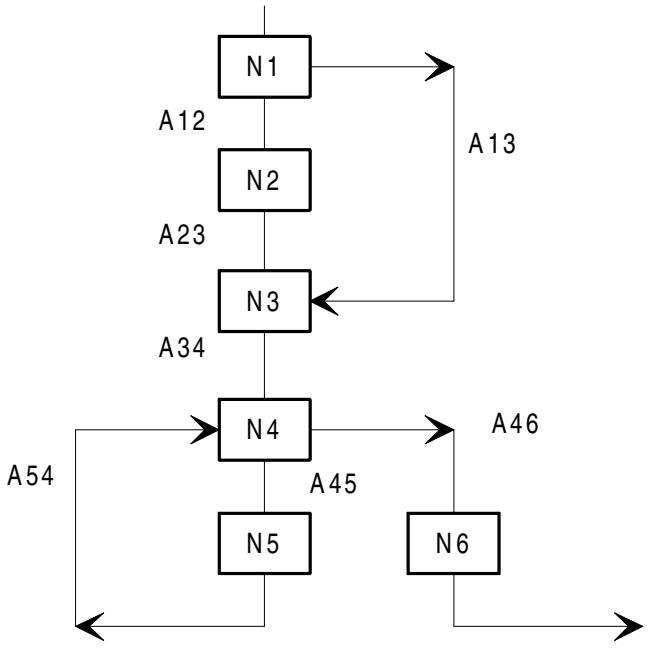
As experience shows, every newly written program contains errors. At execution, these errors are costly. Therefore, it is current practice to place a verification phase after software development.

One should theoretically ensure that the developed program does exactly what the specifications state, and nothing else, but since no mathematical method may prove the correctness of a software in real cases, one usually tests the program's functions as well. However, the number of tests that should be done is infinite, and because of lack of time (and also lack of imagination to think of all possible cases), it is illusory to pretend to test them all.

E. Miller's coverage method progressively builds a set of tests that, following certain criteria, avoid redundancy and reach a measurable degree of completeness.

The program to be tested is represented by a graph where nodes correspond to disjointed portions of executable program, and arcs represent transfers of control between them. The coverage of the program by a given test is then represented by a path way within the graph.

EXAMPLE:



If a test execution corresponds to the path:

```
N1, N2, N3, N4, N5, N4, N6
```

this test ensures the so-called C0 coverage, as every node is traversed at least once.

If two tests respectively correspond to the following paths:

```
N1, N2, N3, N4, N6 and N1, N3, N4, N5, N4, N6
```

this set of tests ensures the C1 coverage, as every node and every arc are traversed at least once.

❑

Miller's method specifies other levels of coverage, up to Cinfinite, which would be reached when all possible paths have been passed.

The COVLIB compiler option provides for information that is needed to ensure coverage C1, or to measure the degree of coverage C1.

In the following description, nodes are linear sequences as defined hereafter.

The Procedure Division is split totally into disjointed contiguous linear sequences in an exclusive manner:

- an end of linear sequence is any point from which a transfer of control to a point not in sequence may take place;
- a beginning of linear sequence is any point to which a transfer of control may take place from a point which is not in the immediate sequence of the program preceding that point.

The transfers of control that are taken into account are those whose destination is a COBOL source instruction, not those internal to COBOL instructions such as INSPECT, for instance.

The program of the figure "ROMAN-TO-ARABIC Program" can be split into linear sequences as in the graph given for example above.

The correspondence between the graph and the source program lines is the following:

node	linear sequence
N1	lines 14 through 16
N2	lines 17 through 18
N3	line 19
N4	lines 20 through 21
N5	lines 24 through 26
N6	lines 22 through 23

This program accepts a Roman number from 1 to 10, translates it into Arabic notation, and prints the result.

```
1       IDENTIFICATION DIVISION.
2       PROGRAM-ID.    ROMAN TO ARABIC.
3       ENVIRONMENT DIVISION.
4       DATA DIVISION.
5       WORKING-STORAGE SECTION.
6       01  ROMAN          PIC X(4)  VALUE SPACES.
7       01  ROMAN-TABLE.
8           02              PIC X(20) VALUE "I    II  III IV V   ".
9           02              PIC X(20) VALUE "VI   VII VIIIIX X   ".
10      01  REDEFINES ROMAN-TABLE.
11          02  ROMAN-NB    PIC X(4)   OCCURS 10.
12      01  ARABIC          PIC 99.
13      PROCEDURE DIVISION.
14      BEGIN.
15          ACCEPT ROMAN.
16          IF ROMAN IS NOT ALPHABETIC-UPPER
17          THEN INSPECT ROMAN CONVERTING "ivx" TO "IVX"
18          END-IF.
19          MOVE 1 TO ARABIC.
20      COMPARE.
21          IF ROMAN = ROMAN-NB (ARABIC)
22          THEN DISPLAY ROMAN "=" ARABIC
23               STOP RUN
24          ELSE ADD 1 TO ARABIC
25               GO TO COMPARE
26          END-IF.
```

ROMAN-TO-ARABIC Program

The JCL for compilation and link may be:

```
CBL SOURCE=ROMAN-TO-ARABIC INLIB=MY_SLLIB COVLIB=MY_COVLIB;
```

```
LINKER ROMAN-TO-ARABIC OUTLIB=MY_LMLIB;
```

The COVLIB option requires that the code needed to ensure C1 coverage measuring must be included in the compile unit. The coverage library (MY_COVLIB in the example) must be of the SL type. This library need not be present at compile time. Its use will be explained later.

During execution of the load module, the number of times each linear sequence executes is counted. The result is printed at the end of the test, under one of the following four forms, depending on the options selected by the user:

1. The SHORT LISTING option gives a histogram of all paths. The figure below is an example of SHORT LISTING. Each linear sequence is represented by its first line number. 16# represents no linear sequence (implicit ELSE). The unused paths are signaled by three stars ("***") immediately to the right of their line number;
2. The LONG LISTING option gives, for each instruction, the type of verb and the percentage of execution number concerning that instruction. An example of LONG LISTING is given later.
3. The UNUSED PATH LISTING gives the list of paths that were not used. An example of UNUSED PATH LISTING is given later.
4. The VERB USAGE ARRAY gives the percentage of execution number for each type of verb. An example of VERB USAGE ARRAY is given later.

To enable this figure to fit in this manual, some spaces were suppressed.

```
PROC-NM  XLN      EXECUTIONS % .  .  .  .  * .  .  .  .  10 .  .  .  .  * .  .  .  .  20

BEGIN                                                       .                             .
          14               1 *************************************************************
          17  ***                                           .                             .
          16#              1 *************************************************************
          19               1 *************************************************************
COMPARE                                                     .                             .
          20               1 *************************************************************
          22               1 *************************************************************
          24  ***                                           .                             .

TOTAL . . . . .            5
```

SHORT LISTING Example

To enable this figure to fit in this manual, some spaces were suppressed.

```
PROC-NM         XLN       EXECUTIONS    %       -> VERB

BEGIN                                  (1)          -> paragraph
                 15                     1  16.666      ACCEPT
                 16                     1  16.666      IF
                 17                     0           -> INSPECT
                 16                    (1)          -> otherwise (#)
                 19                     1  16.666   -> MOVE
COMPARE                                (1)          -> paragraph
                 21                     1  16.666      IF
                 22                     1  16.666   -> DISPLAY
                 23                     1  16.666      STOP
                 24                     0           -> ADD
                 25                     0              GO

TOTAL . . . . . .                      6
```

LONG LISTING Example

```
UNUSED PATHS LIST

17  24
```

UNUSED PATH LISTING Example

```
PROC-NM                EXECUTIONS    %      EXEC # STAT #

ACCEPT                          1  16.666       1       1
ADD                             0               0       1
DISPLAY                         1  16.666       1       1

GO                              0               0       1
IF                              2  33.333       2       2
INSPECT                         0               0       1

MOVE                            1  16.666       1       1
STOP                            1  16.666       1       1

TOTAL . . . .                   6               6       9
```

VERB USAGE ARRAY Example

Options are specified in the options string of the JCL STEP statement. This options string must contain the word "COVLIB", and only the options after "COVLIB" and before the next ";" or the end of the string will be taken into account. If the load module is to receive parameters for its own use, it must eliminate this part of the options string.

The SHORT LISTING, LONG LISTING, UNUSED PATH LISTING and VERB USAGE ARRAY are specified respectively by the letters: "S", "L", "U" and "V".

EXAMPLE 1:

U	for the UNUSED PATH LISTING alone
SUV	for the SHORT LISTING, UNUSED PATH LISTING and VERB USAGE ARRAY.

As the load module may contain several programs compiled with the COVLIB option, the option string must indicate for each of them the type wanted.

❑

EXAMPLE 2:

```
P1 U, P2 U, P3 SUV
```

P1, P2, P3 are the names of the programs as they appear in the PROGRAM-ID entry. The options chosen for the different programs are separated by commas.

A star ("*") may be used in place of program names. It means "any program which is not explicitly referenced". In the above example, if we assume that only P1, P2, and P3 are compiled with the COVLIB option, the same result will be obtained with:

```
* U, P3 SUV
```

No listing is produced for a program which is not called up in the current test.

It is possible to specify the accumulation of the counters calculated during the present execution with those obtained from the previous tests. This gives a means to observe how the test coverage evolves, to decide whether the test currently performed is of utility or not, and, if necessary, to prepare other tests to verify the part of the program that is not yet covered. The accumulation is obtained by a character "-" (minus) immediately following the current test options.

❑

EXAMPLE 3:

```
P2 U-
```

requires the UNUSED PATH LISTING for the current test, and the accumulation of the execution numbers for the program P2;

```
P2 -
```

requires the accumulation only.

It is also possible to specify a cumulated listing by introducing characters "S", "L", "U", "V" after the "-" character.

❑

EXAMPLE 4:

```
P2 S-S
```

requires the histogram of the current test and of all tests already executed (including the present one). These two listings differ by their page header which contain "EXECUTION STATISTICS FOR THE CURRENT RUN" and "SUMMED EXECUTION STATISTICS", respectively.

In order to obtain the cumulating described above, it is necessary to register somewhere the values of the counters. This is done in a member of the library specified by the COVLIB compilation parameter. This member is named "program-name_S", "program-name" being the name of the program as present in the PROGRAM-ID paragraph.

The internal-file-name (ifn) of the library that contains "program-name_S" is H_COVLIB.

Nothing prevents assigning to H_COVLIB another library than that specified in the compilation JCL. In this case, the explicitly assigned library will be used by all the programs of the load module being tested.

❑

EXAMPLE 5:

```
CBL SOURCE=P1 COVLIB=LIB1...
.
.
.
STEP P1 OPTIONS='COVLIB P1 -"...
.
.
.
ENDSTEP;
STEP P1 OPTIONS='COVLIB P1 -'...
ASSIGN H_COVLIB LIB2;
.
.
.
ENDSTEP;
```

During the first execution of P1 (first STEP statement), the results are registered in member P1_S of LIB1; during the second execution of P1 (second STEP statement), the results are registered in member P1_S of LIB2.

❑

If the assigned library already contains a member whose name is "program-name_S", this member is first saved in another member named "program-name_X". The accumulation with the values registered in program-name_S will take place only if "program-name" has not been recompiled nor re-linked. In case no member "program-name_S" exists, or if it corresponds to an old version of "program-name", only those values calculated during the present execution are registered in program-name_S, which is created by the coverage utility if necessary.

Coming back to the ROMAN-TO-ARABIC example, let us suppose that the first test asks for the translation of the Roman number I. Requiring the option "* SU-" causes the listing shown in the figure "Translation of I with Option *-U".

This figure shows that the paths 17 and 24 (N2 and N5) have not been used.

As can also be seen on this figure, recapitulative information is given at the end of every listing:

- on the last-but-one page, the number of unused paths (nodes and arcs) and the number of paths used for the first time (by the current test);
- on the last page, the number of program executions during the current test and during the set of tests already done; and a recall of the options that were asked for.

If the second test asks for the translation of II, with the option "* -U", the figure "Translation of II with Option *-U" is obtained, which shows that path 17 has still not been used.

If the third test uses III, still with "* -U" as an option, the listing obtained in the figure "Translation of III with Option *-U" shows that no new path has been used. This test may therefore be removed from the test coverage set.

Looking more closely at the program structure, it appears that the last path will be used when translating "i". As a result of this test, the figure "Translation of i with Option *-U" shows "0 unused paths".

To enable the figure below to fit in this manual, some spaces were suppressed.

```
PROC-NM      XLN       EXECUTIONS % .  .  .  .  *  .  .  .  .  10  .  .  .  .  *  .  .  .  .  20

BEGIN                                                             .                             .
              14                1 ************************************************************
              17  ***                                             .                             .
              16#               1 ************************************************************
              19                1 ************************************************************
COMPARE                                                           .                             .
              20                1 ************************************************************
              22                1 ************************************************************
              24  ***                                             .                             .

TOTAL . . . . . . .             5

UNUSED PATHS LIST

17  24

2 unused paths (out of 7)

This run used 5 new paths in ROMAN-TO-ARABIC


     PROG-NAME                          CURRENT     SUMMED

     ROMAN-TO-ARABIC                          1          1


Explicit options were:

COVLIB * SU-
```

Translation of I with Option "* -U"

```
UNUSED PATHS LIST
17

1 unused path (out of 7).
This run used 1 new path in ROMAN-TO-ARABIC.
     PROG-NAME                          CURRENT     SUMMED
     ROMAN-TO-ARABIC                          1          2
Explicit options were:
COVLIB * -U
```

Translation of II with Option "* -U"

```
UNUSED PATHS LIST
17
1 unused path (out of 7).
This run did not use any new path in ROMAN-TO-ARABIC.
     PROG-NAME                          CURRENT     SUMMED
     ROMAN-TO-ARABIC                          1          3

Explicit options were:
COVLIB * -U
```

Translation of III with Option "* -U"

```
UNUSED PATHS LIST

     --- NONE ---

0 unused paths (out of 7).

This run used 1 new path in ROMAN-TO-ARABIC.

     PROG-NAME                              CURRENT     SUMMED

     ROMAN-TO-ARABIC                              1          4

Explicit options were:

COVLIB * -U
```

Translation of i with Option "* -U"

Leaving the ROMAN-TO-ARABIC example, below is the complete syntax of the options string:

```
                                  { program-name }
OPTIONS = '[...]COVLIB[nnn]{, {                  }
                              {        *         }

    [S] [U] [L] [V] [-[S] [U] [L] [V] ] }...[;...]'
```

With the following rules:

- nnn is the length of the histogram line (option S). The default value is 120 characters;
- letters "S", "U", "L", and "V", when present, may be placed in any order. They may also be repeated, which will produce the corresponding listings that many times;
- the "*" character may be used only once;
- options may be replaced by more mnemonic strings:

  ```
  "NONE"         for no peculiar option
  "STORE"        for "-"
  "UNUSED"       for "-U"
  "HISTOGRAM"    for "-S"
  "PROFILE"      FOR "-V"
  "FULL"         FOR "SULV-SULV"
  "STANDARD" or "STD" for "S-SUV"
  ```

  If the options string is not present or does not contain the word "COVLIB", the default options string is '120,* S-SUV'.

Listings may be directed to a file rather than immediately printed at the end of the step. For this, the internal-file-name H_COVPR must be assigned to the desired file.

EXAMPLE:

```
STEP...
ASSIGN H_COVPR MY-COVPR MB=TEST-RESULT;
.
.
.
ENDSTEP;
```

In this case, the listings will not be sent to SYSOUT but will be stored in the member TEST-RESULT of the library MY-COVPR.

❑

A. Sample COBOL Program and LINKER Listings

To enable the listings in Appendix A to fit in this manual, some non-significant characters were suppressed and some lines were folded.

```
********************************************************************************************
********************************************************************************************
**** GCOS7                                                                              ****
****                        C  O  B  O  L                                               ****
****                                                  VERSION: 01     DATED: DEC 01, 1987 ****
********************************************************************************************
********************************************************************************************

PROGRAM: FIND-DAY

USER: CASSIRAME

PROJECT: COBL

DATE: MAR 23, 1988

TIME: 18:32:24

COMPILER VERSION: INTERNAL V-01

USER OPTIONS: SOURCE=FIND-DAY COMFILE LIB=0 LIB PRTLIB LEVEL=NSTD DCLXREF XREF

ACTIVE OPTIONS: OBJ, NDEBUG, WARN, OBSERV, NMAP, DCLXREF, XREF, LIST, CKSEQ, CARDID, CASEQ,
                EXPSIZE, DIAGIN, NCODAPND, NOPT, NDEBUGMD, LFATAL, OOBSERV, XLN, NSILENT,
                NDDLIST, NREFMDCK, COBOL85, PSEGMAX=4096(BYTES), DSEGMAX=4096(BYTES),
                ISEGMAX=(65536(BYTES), 0, 16384(BYTES)), TEMP=30, CODE=OBJA.

COMPILATION LEVEL: NOT STANDARD
```

```
COMPILER INPUT:
  ALTER FILE
    ALTER-DAYS IN COBL.DOC (H_ALTER)
    CD=01/23/78 CT=10:35:24 MD=01/23/78 MT=10:35:24 SL=DAT MN=00 NM=ALTER-DAYS
  SOURCE FILE
    FIND-DAY IN COBL.DOC (H_INLIB)
    CD=01/23/78 CT=10:35:24 MD=03/23/88 MT=14:54:36 SL=DAT MN=14 NM=FIND-DAY
  COPY FILE (COPY STATEMENT ON LINE 37.0, COPIED TEXT ON 12 LINES)
    DAYS IN COBL.DOC (H_INLIB)
    CD=01/23/78 CT=10:35:24 MD=01/23/78 MT=10:35:24 SL=DAT MN=00 NM=DAYS

     A.1 ----->     C: COMPILE;
     A.2            R: R FIND-DAY
     A.3            R: /DATA/S/DIVISION/&./
     A.4            R: /01  DITWEEK-TAB/,/SUNDAY/C             COMMENT

                                                1
   * 1  1-44  Text follows the 'A', 'C', 'I' or 'Q' command on the line.  Text is ignored.

     A.5            I:            COPY DAYS
     A.6            I:              REPLACING == PIC X(8) == BY == PIC X(10) ==.
     A.7            I: [F

         1          IDENTIFICATION DIVISION.
         2         *                       THIS ROUTINE, STARTING FROM A DATE, GIVES
         3         *                       THE DAY IN THE WEEK CORRESPONDING TO THE
         4         *                       DATE
         5          PROGRAM-ID. FIND-DAY.
         6         *
         7          ENVIRONMENT DIVISION.
         8          CONFIGURATION SECTION.
         9          SOURCE-COMPUTER. DPS7.
        10          OBJECT-COMPUTER. DPS7.
        11         *
       *12          DATA DIVISION.
        13         *
        14          WORKING-STORAGE SECTION.
        15         *                       TEMPORARIES
        16          01  X PICTURE 9(10).
        17          01  Y PICTURE 9(5).
        18         *                       TOTAL NUMBER OF DAYS PRECEDING THE MONTH
        19         *                       (SHOWN BY ITS ORDINAL NUMBER IN THE LIST)
        20         *                       IN THE YEAR
```

```
        21         01  PREC-D-TAB.
        22             02  FILLER PIC 999 VALUE   0.
        23             02  FILLER PIC 999 VALUE  31.
        24             02  FILLER PIC 999 VALUE  59.
        25             02  FILLER PIC 999 VALUE  90.
        26             02  FILLER PIC 999 VALUE 120.
        27             02  FILLER PIC 999 VALUE 151.
        28             02  FILLER PIC 999 VALUE 181.
        29             02  FILLER PIC 999 VALUE 212.
        30             02  FILLER PIC 999 VALUE 243.
        31             02  FILLER PIC 999 VALUE 273.
        32             02  FILLER PIC 999 VALUE 304.
        33             02  FILLER PIC 999 VALUE 334.
        34         01  PREC-D-TAB-RED REDEFINES PREC-D-TAB.
        35             02  PRECEDING-DAYS PIC 999 OCCURS 12.
        36        *                      TABLE GIVING THE NAME OF THE DAYS IN THE
        37        *                      WEEK
         .             COPY DAYS
         .               REPLACING == PIC X(8) == BY == PIC X(10) ==.
       ..1         01  OTHER-UNUSED.
       ..2             02 FILLER PIC X.
       ..3             02 FILLER COMP-1 SYNC.

                         1
   * 1  2-199 A 1 byte type 2 FILLER item was allocated to align this synchronized item
(see reference manual).

       ..4         01  DITWEEK-TAB.
      *..5             02  FILLER                       PIC X(10)
      *..5                                 VALUE "LUNDI   ".
       ..6             02  FILLER PIC X(10) VALUE "MARDI   ".
       ..7             02  FILLER PIC X(10) VALUE "MERCREDI".
       ..8             02  FILLER PIC X(10) VALUE "JEUDI   ".
       ..9             02  FILLER PIC X(10) VALUE "VENDREDI".
      ..10             02  FILLER PIC X(10) VALUE "SAMEDI  ".
      ..11             02  FILLER PIC X(10) VALUE "DIMANCHE".
       -46         01  DITWEEK-TAB-RED REDEFINES DITWEEK-TAB.
        47             02  DAY-IN-THE-WEEK PIC X(10) OCCURS 7 TIMES.
        48        *                      AREA FOR DATE SPLITTING INTO YEAR, MONTH,
        49        *                      AND DAY
        50         01  SPLIT-DATE.
        51             02  CENTURY       PIC 99.
        52             02  SHORT-DATE.
        53             03  FILLER        PIC 99.
        54             03  MONTH         PIC 99.
        55             03  DAY-OF-MONTH  PIC 99.
        56             03  DAY-OF-MONTH-X REDEFINES DAY-OF-MONTH PIC XX.
```

```
57          01 YEAR REDEFINES SPLIT-DATE PIC 9(4).
58         *                       ORDINAL NUMBER OF THE DAY TAKEN INTO
59         *                       CONSIDERATION WITHIN THE DAYS OF THE
60         *                       CHRISTIAN ERA
61          01  DAYS-IN-THE-ERA PIC 9(10).
62         *
63          LINKAGE SECTION.
64         *                       DATE FOR WHICH THE DAY OF WEEK IS LOOKED
65         *                       FOR, UNDER THE FORM YYYYMMDD OR YYMMDDBB
66         *                       (WHERE B MEANS BLANK)
67          01  FULL-DATE PIC X(8).
68         *                       RETURNED ORDINAL NUMBER OF THE DAY IN THE
69         *                       WEEK (1 IS MONDAY, 2 TUESDAY ... )
70          01  DAY-OF-THE-WEEK PIC 9.
71         *                       RETURNED DAY IN THE WEEK ITSELF
72          01  DAY-ITSELF PIC X(10).
73         /
74          PROCEDURE DIVISION USING FULL-DATE DAY-OF-THE-WEEK DAY-ITSELF.
75         *
76          BEGIN.
77              PERFORM COMPUTE-DAY-OF-THE-WEEK.
78          THE-END.
79              EXIT PROGRAM.
80         *
81         *
82         *
83         *
84          COMPUTE-DAY-OF-THE-WEEK.
85              MOVE FULL-DATE TO SPLIT-DATE.
86              IF DAY-OF-MONTH-X = SPACE
87                MOVE SPLIT-DATE TO SHORT-DATE

                                     1
** 1  5-148 The receiving item may be truncated on the right.
** 1  5-264 Sending and receiving fields overlap.
 * 1  5-184 This is a group move and operands do not have the same size.
```

```
        88               MOVE 19 TO CENTURY.
        89        *                      LET US COMPUTE THE NUMBER OF DAYS SPENT
        90        *                      SINCE THE BEGINNING OF THE CHRISTIAN ERA
        91             COMPUTE DAYS-IN-THE-ERA =
        92                           DAY-OF-MONTH
        93                         + PRECEDING-DAYS (MONTH)
        94                         + (YEAR - 1) * 365.
        95        *                      LET US ADD 1 FOR EACH LEAP-YEAR, INCLUDING
        96        *                      THE YEAR OF THE PROCESSED DATE IF THE
        97        *                      MONTH IS LATER THAN FEBRUARY
        98             IF MONTH < 3 COMPUTE YEAR = YEAR - 1.
        99             DIVIDE YEAR BY 4 GIVING X.
       100             ADD X TO DAYS-IN-THE-ERA.
       101             DIVIDE YEAR BY 100 GIVING X.
       102             SUBTRACT X FROM DAYS-IN-THE-ERA.
       103             DIVIDE YEAR BY 1000 GIVING X.
       104             ADD X TO DAYS-IN-THE-ERA.
       105        *                      NOW THE REMAINDER OF THE DIVISION BY 7 OF
       106        *                      THE DAYS-IN-THE-ERA, AUGMENTED OF THE
       107        *                      PROPER CONSTANT, IS THE ORDINAL NUMBER OF
       108        *                      THE DAY IN THE WEEK
       109             DIVIDE DAYS-IN-THE-ERA BY 7 GIVING X REMAINDER Y.
       110             IF Y > 4
       111               SUBTRACT 4 FROM Y
       112             ELSE
       113               ADD 3 TO Y.
       114             MOVE Y TO DAY-OF-THE-WEEK.

                                1
** 1  5-156 Possible left truncation.

       115             MOVE DAY-IN-THE-WEEK (Y) TO DAY-ITSELF.
```

```
  FIND-DAY                                PROGRAM-NAME   1      NOREF SEPARATELY-COMPILED

01 X                          1:000055  DISPLAY 9(10)  16     99* 100 101* 102 103* 104 109*

01 Y                          1:00005F  DISPLAY 9(5)   17     109* 110 111* 113* 114 115

01 PREC-D-TAB                 1:000064  GROUP   X(36)  21     34

01 PREC-D-TAB-RED             1:000064  GROUP   X(36)  34     NOREF
02 PRECEDING-DAYS (+ 0)       1:000064  DISPLAY 9(3)   35     93

01 OTHER-UNUSED               1:000088  GROUP   X(4)   37.0.1 NOREF

01 DITWEEK-TAB                1:00008C  GROUP   X(70)  37.0.4 46
```

```
01 DITWEEK-TAB-RED              1:00008C  GROUP   X(70)  46     NOREF
02 DAY-IN-THE-WEEK (+ 0)        1:00008C  DISPLAY X(10)  47     115

01 SPLIT-DATE                   1:0000D4  GROUP   X(8)   50     57 85* 87
02 CENTURY (+ 0)                1:0000D4  DISPLAY 99     51     88*
02 SHORT-DATE (+ 2)             1:0000D6  GROUP   X(6)   52     87*
03 MONTH (+ 4)                  1:0000D8  DISPLAY 99     54     93 98
03 DAY-OF-MONTH (+ 6)           1:0000DA  DISPLAY 99     55     56 92
03 DAY-OF-MONTH-X (+ 6)         1:0000DA  DISPLAY XX     56     86

01 YEAR                         1:0000D4  DISPLAY 9(4)   57     94 98+* 99 101 103

01 DAYS-IN-THE-ERA              1:0000DC  DISPLAY 9(10)  61     91* 100* 102* 104* 109

01 FULL-DATE                 1  000000    DISPLAY X(8)   67     74 85

01 DAY-OF-THE-WEEK           2  000000    DISPLAY 9      70     74 114*

01 DAY-ITSELF                3  000000    DISPLAY X(10)  72     74 115*



   BEGIN                                  PARAGRAPH-NAME 76     NOREF
   THE-END                                PARAGRAPH-NAME 78     NOREF
   COMPUTE-DAY-OF-THE-WEEK                PARAGRAPH-NAME 84     77
   BEGIN                                  PARAGRAPH-NAME 76     NOREF
02 CENTURY (SPLIT-DATE + 0)     1:0000D4  DISPLAY 99     51     88*
   COMPUTE-DAY-OF-THE-WEEK                PARAGRAPH-NAME 84     77

02 DAY-IN-THE-WEEK (DITWEEK-TAB-RED + 0)
                                1:00008C  DISPLAY X(10)  47     115

01 DAY-ITSELF                3  000000    DISPLAY X(10)  72     74 115*

03 DAY-OF-MONTH (SPLIT-DATE + 6)
                                1:0000DA  DISPLAY 99     55     56 92

03 DAY-OF-MONTH-X (SPLIT-DATE + 6)
                                1:0000DA  DISPLAY XX     56     86

01 DAY-OF-THE-WEEK           2  000000    DISPLAY 9      70     74 114*
01 DAYS-IN-THE-ERA              1:0000DC  DISPLAY 9(10)  61     91* 100* 102* 104* 109
01 DITWEEK-TAB                  1:00008C  GROUP   X(70)  37.0.4 46
01 DITWEEK-TAB-RED              1:00008C  GROUP   X(70)  46     NOREF
   FIND-DAY                               PROGRAM-NAME    1     NOREF SEPARATELY-COMPILED
01 FULL-DATE                 1  000000    DISPLAY X(8)   67     74 85
03 MONTH (SPLIT-DATE + 4)       1:0000D8  DISPLAY 99     54     93 98
01 OTHER-UNUSED                 1:000088  GROUP   X(4)   37.0.1 NOREF
01 PREC-D-TAB                   1:000064  GROUP   X(36)  21     34
01 PREC-D-TAB-RED               1:000064  GROUP   X(36)  34     NOREF
```

```
02 PRECEDING-DAYS (PREC-D-TAB-RED + 0)
                                1:000064  DISPLAY 9(3)    35    93

02 SHORT-DATE (SPLIT-DATE + 2)  1:0000D6  GROUP   X(6)    52    87*
01 SPLIT-DATE                   1:0000D4  GROUP   X(8)    50    57 85* 87
   THE-END                                PARAGRAPH-NAME 78    NOREF
01 X                            1:000055  DISPLAY 9(10)   16    99* 100 101* 102 103* 104 109*
01 Y                            1:00005F  DISPLAY 9(5)    17    109* 110 111* 113* 114 115
01 YEAR                         1:0000D4  DISPLAY 9(4)    57    94 98+* 99 101 103



121 SOURCE LINES



SUMMARY OF ERRORS

      *            3        ON LINES   A.4      37.0.3    87
    * *            3        ON LINES    87   114
  * * *            0
* * * *            0

N S T D            1



CU PRODUCED ON LIBRARY ;106134.TEMP.CULIB



SEGMENT NAME        TYPE      BYTES  (CODE)      ADDRESS

FIND-DAY.0          ..L         112              ?BR7.50->
FIND-DAY.1          .D.         236              ?BR7.28->
FIND-DAY.2          C..         508  (508)       ?BR7.5C->

STACK                            91



RUN TIME PACKAGE PROCEDURES INVOKED

NONE
```

```
*****************************************************************************************
*****************************************************************************************
**** GCOS7                                                                           ****
****                                   L I N K E R                                   ****
****                                              VERSION: 90.00  DATED: JUN 30,1986  ****
*****CALENDAR_K********************************************** 16  -2*********************
*****************************************************************************************

ADDITIONAL INFO:     4   5

    1              CODE (DEFAULT):      OBJA

      ****************************** LINKER CONTROL STATEMENTS ******************************

    2     ENTRY=CALENDAR,
    3     LIST=XREF,

*********************************************TASK=MAIN*************************************

    4     ENTRY POINT = CALENDAR          LOCATION:  8.10.000008     IN CU: CALENDAR
    5    0.CU= CALENDAR           FROM:INLIB  CREATED  ON  4/19/88 AT 16:46:12 BY: COB82  1.0
    6                                                                          O: ...........E.
    7     -SYMREFS                  TYPE   LOCATION     MATCH. DEF IN
    8     1.PRT                     DATA    8.10.000058 **__BLANK**/ 8.14.000000
    9     1.PRT                     DATA    8.11.00005A **__BLANK**/ 8.14.000000
   10     1.FIND-DAY                PROC    8.10.0000AA FIND-DAY/ 8.15.000008
   11     1.CU= FIND-DAY           FROM:INLIB  CREATED  ON  4/19/88 AT 16:45: 0 BY: COB82  1.0
   12                                                                          O: ...........E.


   13 ====================================GROUP INFORMATION ================================

   14                                    GLOBAL SEGMENTS
   15     SEGNAME            SEG NUM   CONTAINS
   16     H_U_BIFN            9. A                          LOCATION                 LOCATION
   17                                 H_CBL_DRTPD2             000000
   18     H_CBL_DRTP          8.13                          LOCATION                 LOCATION
   19                                 H_CBL_DRTPD1             000000
   20     __BLANK             8.14                          LOCATION                 LOCATION
   21                                 PRT                      000000   H_PRT_CFD       000008
```

```
22                                      SEGMENT  LIST
23     SEG.   IN CU.ISN           TYPE SH RF RD WR EX WP EP  G  S    SIZE MAXSIZE  CONT.P.
24      8.10  CALENDAR.0          ..L  3  3  3  3  3                  256           0
25      8.11  CALENDAR.1          .D.  3  3  3  3  3  W               1472           0
26      8.12  CALENDAR.2          C..  3  3  3  3  3     E            2592           0
27      8.13  H_CBL_DRTP          .D.  3  3  3  3  3  W                608    8192   0
28      8.14  __BLANK             .D.  3  3  3  3  3  W                 80           0
29      8.15  FIND-DAY.0          ..L  3  3  3  3  3                  112           0
30      8.16  FIND-DAY.1          .D.  3  3  3  3  3  W                240           0
31      8.17  FIND-DAY.2          C..  3  3  3  3  3     E            512           0
32     .....
33      9. 0  PGCR                CD.  2  3  3  0  3  W  E           4576
34      9. 4  TERMINATION         .D.  2  3  3  0  0  W         S       96
35      9. A  H_U_BIFN            .D.  2  3  3  3  3  W                64
36      9. E  SEMPH. POOL         .D.  2  3  3  1  1  W         S     3536
37     .....

====================================LIST OF CU  (S) ========================================

         CALENDAR             INLIB      CREATED 16:46:12  APR 19, 1988   BY:   COB82   1.0
                  CU OPTION :  EOD
         FIND-DAY             INLIB      CREATED 16:45:00  APR 19, 1988   BY:   COB82   1.0
                  CU OPTION :  EOD
```

```
**************************************CROSS-REFERENCE LIST  ********************************

-EXTERNAL NAME- -   TYPE  -  -LOCATION- -1ST DEF IN CU - -                REFERENCED IN :

CALENDAR                      LINE:    5
               PROC SYMDEF  08.10.0008  CALENDAR   **ENTRY POINT**
FIND-DAY                      LINE:   11
               PROC SYMDEF  08.15.0008  FIND-DAY
                                                     9    CALENDAR
H_CBL_DRTP                    LINE:    5
               SEGT NAME    08.13.0000  CALENDAR
H_CBL_DRTPD1                  LINE:    5
               GLOBAL DATA  08.13.0000  CALENDAR
                                                     5    CALENDAR             5    CALENDAR
H_CBL_DRTPD2                  LINE:    5
               GLOBAL DATA  09.0A.0000  CALENDAR
                                                     5    CALENDAR             5    CALENDAR
H_CBL_UCLOSE                  LINE:    9
               PROC SYMREF  **NOLINK**
                                                     9    CALENDAR
H_CBL_UCLOSI                  LINE:    9
               PROC SYMREF  **NOLINK**
                                                     9    CALENDAR
H_CBL_UDMRC                   LINE:   11
               PROC SYMREF  **NOLINK**
                                                    11    CALENDAR
H_CBL_UFCHK                   LINE:    9
               PROC SYMREF  **NOLINK**
                                                     9    CALENDAR
H_CBL_UOPEN                   LINE:    9
               PROC SYMREF  **NOLINK**
                                                     9    CALENDAR
H_CBL_USTOP                   LINE:    9
               PROC SYMREF  **NOLINK**
                                                     9    CALENDAR
H_PRT_CFD                     LINE:    5
               GLOBAL DATA  08.14.0008  CALENDAR
                                                     8    CALENDAR
H_S_PRT                       LINE:    5
               GLOBAL DATA  09.0B.0000  CALENDAR
                                                     9    CALENDAR             9    CALENDAR
H_STND2_MEPT                  LINE:    8
               DATA SYSDEF
                                                     8    CALENDAR            11    FIND-DAY
H_STND2_TDF1                  LINE:    9
               DATA SYSDEF
                                                     9    CALENDAR
H_STND2_UDMA                  LINE:    8
```

```
               DATA SYSDEF
                                                    8    CALENDAR
H_U_BIFN                        LINE:    5
               SEGT NAME    09.0A.0000  CALENDAR
PRT                             LINE:    5
               GLOBAL DATA  08.14.0000  CALENDAR
                                                    5    CALENDAR            8    CALENDAR


************************************************LINKAGE REPORT*****************************

                NO ERRORS DETECTED
                -------------------------
.  OUTPUT MODULE PRODUCED ON LIBRARY   ;102160.TEMP.LMLIB
MODULE IS OF CLASS  (CODE):          0


*************************************L*I*N*K*E*R*****************************************

********************************** END OF SESSION*************************************LAST

PERCENTAGE OF SPACE USED           1
```

B. COBOL Compiler Diagnostics

In the following diagnostic codes:

- the concatenation of the first two columns gives the error number;
- the third column gives the gravity of the error:
  - observation
  - warning
  - serious error
  - fatal error
  - warning for the current release - will be a serious error at next release
  - observation or fatal error depending on which of the LOBSERV or LFATAL parameters of the $CBL JCL statement is active
  - COBOL 85 obsolete feature;
- the right-most column contains the error message itself. In the text of the message, variable information is shown by the symbol ^.

1 -1 3
Illegal character. Replaced by blank.

1 -2 3
Too long a PICTURE character-string. PICTURE character-string is truncated.

1 -3 3
End delimiter missing in a literal. Delimiter is assumed.

1 -4 3
Too long a literal. Literal is truncated.

1 -5 3
Illegal continuation of a non-numeric literal. Column 7 ignored.

1 -6 3

Debugging line disallowed in pseudo-text preceding "BY". Line accepted.

1 -7 2

Sequence error or non-numeric line number. Line is accepted.

1 -8 5

Area A of a continuation line must be blank. Area A is ignored.

1 -9 3

Continuation line not allowed after a debugging or a blank line, within a comment-entry or as the first line of source or copied text. Column 7 is ignored.

1 -10 3

First word after "COPY" or "REPLACE" processing, if any, is neither "CONTROL" nor "IDENTIFICATION", or it does not begin in Area A.

1 -11 1

Syntax checking discontinued.

1 -12 1

Syntax checking resumed.

1 -13 4

Implementation restriction. No room enough to accommodate "REPLACE", "COPY ... REPLACING ..." and/or statement scanning.

1 -14 3

Zero length or too long word after replacement. Replacement did not take place.

1 -15 2

This "BY" phrase cannot participate to replacement because of a previous "BY" phrase.

1 -16 3

"COMPILE" command or terminating semi-colon thereof assumed to be missing. Line is reprocessed.

1 -17 3

Empty pseudo-text to the left of "BY". This "BY" phrase will not participate to replacement.

1 -18 3
Illegal character in column 7. The line is processed as a blank line.

1 -19 4
No "COMPILE" command found in the alter file.

1 -20 3
Duplicate or out of sequence division header.

1 -21 3
^ Division missing.

1 -22 4
Illegal delimiter for the regular expressions of an "S" command.

1 -23 3
This word in Area A is not a user-defined word.

1 -24 3
This word is reserved for future implementation.

1 -25 3
Illegal character in symbolic-character. End of literal is ignored.

1 -26 6
This feature is a ^ feature, not included in the current compilation level.

1 -27 3
The use of this reserved word has been restricted by this installation.

1 -28 4
Too complex substitute string.

1 -29 3
Zero length PICTURE character string. "PICTURE X" is assumed.

1 -30 3
Zero length non-numeric literal. "SPACE" is assumed.

1 -31 3
Too long a source line. Line is truncated.

1 -32 1
First word of text replaced (or deleted).

1 -33 1
Last word of text replaced (or deleted).

1 -34 1
Word replaced (or deleted).

1 -35 4
Line too long after alter substitution.

1 -36 4
Unknown or illegal alter request.

1 -37 4
The first request of the alter enclosure is not an "R" request without address expression.

1 -38 4
An "R" request is allowed only as the first request of an alter enclosure.

1 -39 4
Address expression missing before "," or ";".

1 -40 4
Address expression missing after "," or ";".

1 -41 4
Address range is not followed by a proper request.

1 -42 4
End of request missing in alter line.

1 -43 4
Relative address value missing in alter command.

1 -44 1
Text follows the "A", "C", "I" or "Q" command on the line. Text is ignored.

1 -45 4
Dollar must not be the first address of an address range.

1 -46 4
Dollar must not be followed by a relative address.

1 -47 4
Syntax error in regular expression.

1 -48 4
Numeric addresses are meaningful only with source program in SSF format.

1 -49 3
Operand following "BY" is illegal or missing.

1 -50 3
Impossible to note on ^ where to start from at next compilation.

1 -51 4
Impossible to recognize the last line in ^.

1 -52 3
^ is referenced, but it is not assigned.

1 -53 4
Impossible to open ^.

1 -54 4
Impossible to open ^.

1 -55 4
Impossible to initiate a new compilation. Repositioning on ^ cannot be done.

1 -56 3
Debugging lines are allowed only after the "OBJECT-COMPUTER" paragraph. Line is ignored.

1 -57 3
First group of contiguous non blank characters assumed to be "COMPILE".

1 -58 3
Text follows semi-colon. Text is ignored.

1 -59 3
Illegal option or value. Ignored from this point on.

1 -60 4
Semi-colon missing at the end of the "COMPILE" command.

1 -61 4
^ is not a source library.

1 -62 1
^ (console message)

1 -63 4
Illegal COBOL option string.

1 -64 1
Too many percent lines. Line is ignored.

1 -65 4
The command does not specify a member, and none is currently implied.

1 -66 3
Text follows the command. Text is ignored.

1 -67 4
Library-name specified in the command is too long.

1 -68 4
Member-name missing in the command.

1 -69 4
Member-name specified in the command is too long.

1 -70 4
Undefined regular expression.

1 -71 4
Maximum regular expression length exceeded.

1 -72 3
Copy text in ^ not exhausted.

1 -73 3
Alter text in ^ not exhausted.

1 -74 4
More than one library may match ^.

1 -75 3
More than one library may match ^.

1 -76 4
Syntax error in regular expression.

1 -77 3
Illegal continuation of a name. Column 7 is ignored.

1 -78 1
This non standard spelling is reserved to system-names.

1 -79 3
The "REPLACING" phrase of this "COPY" statement does not apply to the copied "REPLACE" statement.

1 -80 3
Illegal character in a Boolean literal. Character is ignored.

1 -81 3
Impossible to close ^.

1 -82 3
Illegal or missing symbolic-character in a non-numeric literal. The highest position of the native collating sequence is assumed.

1 -83 3
Right-most character missing in symbolic-character. "ZERO" is assumed.

1 -84 3
Impossible to close ^.

1 -85 3
Separator missing before the word. Blank is assumed.

1 -86 4
Referenced line not found or already passed.

1 -87 3
Punctuation character is not followed by a blank.

1 -88 3
Illegal continuation of a word. Column 7 is ignored.

1 -89 3
Illegal continuation of a non-numeric literal. Missing quote is assumed.

1 -90 3
End quote missing in a non-numeric literal. Missing quote is assumed at the end of the line or of the last continuation line, if any.

1 -91 3
Too long a numeric literal. Integral part is truncated.

1 -92 3
Too long a numeric literal. Fractional part is truncated.

1 -93 3
Error while purging ^.

1 -94 1
Some errors on this line may indeed apply to one of the first lines following the current copied text (if any).

1 -95 1
The end of this line has been moved after the copied text (if any).

1 -96 1
Too short a record on ^ to be an SSF record. Line is ignored and is not shown in the listing.

1 -97 3
PICTURE character-string ends with a period or a comma. The period(s) and/or comma(s) terminating the PICTURE character-string are ignored.

1 -98 3

Too long a name. Name is truncated.

1 -99 3

Name terminates with a hyphen.

1 -100 1

The apostrophe is used instead of the quote to delimit literals in this program.

1 -101 3

Period missing after the "REPLACE" statement.

1 -102 3

Nested "COPY" statement. "COPY" statement is not applied.

1 -103 3

Text-name missing in "COPY" statement. "COPY" parse is terminated.

1 -104 3

Library-name missing in "COPY" statement. "IN" or "OF" are ignored.

1 -105 3

Period missing after this "COPY" statement.

1 -106 3

Period missing after this "COPY" statement. Though not repeated below in the source listing, the word following the statement will be properly taken care of.

1 -107 2

Unexpected SSF control record in ^. Record is ignored and is not shown on the source listing.

1 -108 3

Too long a record in ^ to be an input line. Record is ignored and is not shown on the source listing.

1 -109 3

Abnormal termination of the source while processing an alter insert, change or append enclosure.

1 -110 2
End of line considered as comment.

1 -111 4
No alter data available. ^ is empty.

1 -112 4
The alter enclosure in ^ does not contain data.

1 -113 3
End of source program reached while seeking for a line specified in an alter command.

1 -114 4
^ not found in assigned or specified input libraries, if any.

1 -115 3
^ not found in assigned or specified input libraries.

1 -116 4
None of the specified input libraries is ^.

1 -117 3
None of the specified input libraries is ^.

1 -118 4
^ not assigned.

1 -119 3
^ not assigned.

1 -120 4
^ not assigned.

1 -121 4
^ not found.

1 -122 3
^ not found.

1 -123 4
SSF format for ^ is neither COBOL, nor COBOLX, nor DATASSF.

1 -124 3

SSF format for ^ is neither COBOL, nor COBOLX, nor DATASSF.

1 -125 4

SSF format for ^ is not DATASSF.

1 -126 3

Empty continuation line. Line is ignored.

1 -127 3

Missing closing bracket in identifier. This "BY" phrase will not participate to replacement.

1 -128 3

Qualifier missing in identifier. This "BY" phrase will not participate to replacement.

1 -129 3

Replacement specification missing.

1 -130 4

Error while reading ^.

1 -131 3

The expected word was "BY" or the ending period is missing after the previous "COPY" or "REPLACE" statement.

1 -132 3

Subscript missing in identifier. This "BY" phrase will not participate to replacement.

1 -133 3

Relative index missing in identifier. This "BY" phrase will not participate to replacement.

1 -134 4

No source program available. ^ is empty.

1 -135 4

No source program available. ^ (specified in the "R" command) is empty.

1 -136 3

Ending pseudo-text delimiter missing. This "BY" phrase will not participate to replacement.

1 -137 3

Illegal or missing exponent. Zero is assumed.

1 -138 3

Search for substitution failed.

1 -139 4

Next (or first) source in ^ cannot be accessed.

1 -140 3

"COPY" word found within a "COPY" or a "REPLACE" statement.

1 -141 3

Source text in ^ not exhausted.

1 -142 4

The "[B" request is not included in the set of alter commands.

1 -143 4

^ is assumed to be an external-file-name and as such is not allowed in an "R" command. Only "INLIB", "INLIB1", "INLIB2", "INLIB3" and "LIB" are allowed as subfile qualifier.

1 -144 4

Only the semi-colon is allowed in a range whose first address is a compound address.

1 -145 2

Punctuation character is not followed by a blank.

1 -146 3

"COPY" statement not fully contained in a debugging line.

1 -147 2

This option is LEVEL-62 specific. The entire section is scanned off.

1 -148 3
A "USE FOR DEBUGGING ON ALL PROCEDURES" has been previously met.

1 -149 1
^

1 -150 1
^

1 -151 1
Error messages about the current "COPY" statement have been lost from this point on.

1 -152 6
Error messages about the current "COPY" statement have been lost from this point on.

1 -153 2
This feature is a LEVEL-62 feature.

1 -154 3
This item may only be referenced in a paragraph of a "USE FOR DEBUGGING" section.

1 -155 1
A line may be lost.

1 -156 2
LEVEL-62 specific "DEBUG-ITEM" reference.

1 -157 3
^ is assumed to be in SSF format.

1 -158 2
LEVEL-62 specific column 7. The line is processed as a blank line.

1 -159 6
This feature (^2) is a ^1 feature, not included in the current compilation level.

1 -160 3

More than 30 characters are specified in a PICTURE character-string.

1 -161 3

Illegal character is specified in the PICTURE character-string.

1 -162 3

Illegal combination of characters is specified in the PICTURE character-string.

1 -163 3

The length of the editing character-string must not exceed 256 characters.

1 -164 3

No receiving characters are specified in the PICTURE character-string.

1 -165 1

^ Division missing.

1 -166 3

Illegal data type or library-name.

1 -167 3

Unable to provide altered source.

1 -168 4

Missing member-name(s).

1 -169 3

No positions specified for the exponent in a floating-point PICTURE character-string.

1 -170 3

Sign missing in the exponent.

1 -171 1

This word may be reserved in a compilation whose level is other than "DSA".

1 -172 2

This word is a reserved word in COBOL-85.

1 -173 2

Word allowed in Area A as LEVEL-62 feature.

1 -174 4

Too many source lines.

1 -175 1

This "BY" phrase does not participate to replacement.

1 -176 3

Continuation line not allowed after a debugging line or as first line of source or copied text. Column 7 is ignored.

1 -177 2

The absence of a blank after a punctuation is allowed as a LEVEL-62 feature.

1 -178 4

The file assigned to ^ is a library.

1 -179 4

The file assigned to ^ is not a library.

1 -180 4

Compiler error: loop in LEX.

1 -181 3

Program-name missing or incorrectly spelled in this "END PROGRAM" header.

1 -182 4

Too many levels of contained program nesting.

1 -183 3

The name of this program is not a unique program-name.

1 -184 3

"END PROGRAM ^" missing.

1 -185 3

Ambiguous reference to a program.

1 -186 3
Period missing after this "END PROGRAM" header.

1 -187 2
This program may be recursively called.

1 -188 2
This program is not called.

1 -189 1
Segment-number ^ is assumed.

1 -190 1
There are too many levels of calls of contained programs to permit space re-use or to warn possibly recursive calls.

1 -191 1
There are too many calls of contained programs to permit space re-use or to warn possibly recursive calls.

1 -192 1
The sign is superfluous with a literal whose value is zero.

1 -193 4
"SKIP" must not be specified when no source program is waiting.

1 -194 4
No commands are specified in the alter enclosure.

1 -195 3
Text follows the "END PROGRAM" header of the outermost program.

1 -196 3
The Procedure Division does not end with a period.

1 -197 3
"END" in Area A must be followed by "DECLARATIVES" or "PROGRAM".

1 -198 3
Program-name is blank.

1 -199 5

Program-name contains embedded blanks.

1 -200 3

Program-name exceeds 31 characters in length.

1 -201 2

^ Division missing in the preceding program or in the program terminating at this point.

1 -202 1

^.

1 -203 3

Incorrect reference modification. This "BY" phrase will not participate to replacement.

1 -204 1

^.

1 -205 2

This feature (^) is a LEVEL-62 feature.

1 -206 3

Program-name must not be a reserved word.

1 -207 7

This feature is temporarily kept in the COBOL Standard. It will disappear from a future version of the Standard.

1 -208 4

"SOURCE=" is missing though an input library is specified.

1 -209 1

Some of the error(s) on this line may actually apply to a piece of text that is replaced or deleted as a result of, or before, the "COPY ... REPLACING" and/or "REPLACE" processing.

1 -210 3

The decimal point used in the literal is not the one in use in this program.

1 -211 3

A floating-point literal whose mantissa has a zero value must have a zero exponent, and must not contain a minus sign.

1 -212 4

Illegal command.

1 -213 4

"LIB" is not specified or the source program does not comprise a correct member or file.

1 -214 4

The source program does not comprise a complete member or file; the "Z" or "W" command is not satisfied.

1 -215 1

Some of the error(s) may actually apply to another line.

1 -216 3

A "REPLACE" statement in the Substitution Section is not allowed with other "REPLACE" statements. It is ignored from this point on.

1 -217 3

Period missing after the "REPLACE" statement. The first word after the "REPLACE" statement may not have been replaced even if the "REPLACE" processing is applicable.

1 -218 3

The preceding "REPLACE" statement is not immediately followed by the Identification Division header.

1 -219 3

"COPY ... REPLACING" or "REPLACE" processing causes a debugging line to be continued.

1 -220 1

This feature (^2) is a ^1 feature, not included in the current compilation level.

1 -221 3

"DB-CXT", "DB-PARAMETERS" and "DB-REGISTERS" can only be used in the "USING" phrase of the "PROCEDURE DIVISION" header or of the "CALL" statement.

1 -222 3

Too long a text-word. Text-word is truncated.

1 -223 3

Illegal continuation of a text-word.

1 -224 4

This command is available only in interactive mode.

1 -225 3

Compiler error: lexical analysis delivered a text-word.

1 -226 2

This text-word contains one or more illegal characters.

1 -227 2

In this program, the "USAGE" of at least one data item is "COMP-6", "COMP-7", "COMP-11", "COMP-12", "COMP-13" or "COMP-14". Its representation is that of a "COMP-2", "COMP-1", "COMP-9", "COMP-10", "COMP-9" or "COMP-10" data item, respectively.

1 -228 3

Separator missing before the text-word. Blank is assumed.

1 -229 3

The default "REPLACING" option (initiated by an "&" in the continuation area of the first copied line) is not terminated at the end of the copied text. A period is missing or the pseudo-text delimiters are unbalanced.

1 -230 3

A text-word that does not obey the rules for formation of a COBOL word or a literal must be specified within a pseudo-text.

1 -231 3

Zero length Boolean literal. "ZERO" is assumed.

1 -232 1

This word or literal is suffixed by "^".

1 -233 2

The next character, which is an "=" sign, should have been on this line in order that a pseudo-text delimiter be recognized.

1 -234 3

This word is not preceded by a punctuation period. A "REPLACE" statement cannot be recognized.

1 -235 3

First word after "COPY" or "REPLACE" processing, if any, is neither "CONTROL" nor "IDENTIFICATION" nor "SELECT" nor the level-number 01, or it does not begin in the proper area.

1 -236 4

Ending non-numeric literal delimiter missing in regular expression.

1 -237 4

The default regular expression has been lost because a "Y" command changed the non-numeric literal delimiter since the default has last been taken.

1 -238 3

First group of contiguous non-blank characters assumed to be "EXTRACT".

1 -239 4

No source member specified in the "EXTRACT" command.

1 -240 1

This word in a comment-entry is not the beginning of a "COPY" statement.

1 -241 1

This word in a comment-entry is not the beginning of a "REPLACE" statement.

1 -242 3

The submitted source is not consistent with the "EXTRACT" parameter.

1 -243 2

In this "BY" phrase, matching may have been achieved based on the internal value of a literal rather than based on its representation in the source program.

1 -244 2

Debugging line in a "COPY" statement, but outside or not wholly contained in a pseudo-text. The "D" in continuation area is taken care of only for pseudo-text, if any, not preceded by a pseudo-text delimiter in this line.

1 -245 2

Debugging line in a "REPLACE" statement, but outside or not wholly contained in a pseudo-text.

1 -246 3

Impossible to note on ^1 the command following the "EXTRACT" command.

1 -247 3

No object member specified in the "EXTRACT" command.

1 -248 4

Closing parenthesis missing in "EXTRACT" command.

1 -249 4

The object member already exists.

1 -250 4

File or library assigned to ^1 is not known.

2 -1 2
This reserved word should begin in Area A.

2 -2 3
The reserved word "DIVISION" is missing.

2 -3 2
Period expected after the previous word.

2 -4 3
The reserved word "PROGRAM-ID" is missing.

2 -5 3
The program-name is missing or incorrectly specified in PROGRAM-ID paragraph.

2 -6 3
A division, section, or paragraph header is missing.

2 -7 2
This Identification Division paragraph has appeared previously.

2 -8 1
Identification Division paragraphs appear in incorrect order.

2 -9 3
Conflicting clauses in this file control entry: ^1 and ^2.

2 -10 3
Invalid or missing computer-name.

2 -11 6
^ is missing, though mandatory when compiling at "DSA" level.

2 -12 3
The reserved word "SECTION" should appear here.

2 -13 3
The word "DEBUGGING" should appear here.

2 -14 3
The reserved word "MODE" should appear here.

2 -15 3
A too high value is specified.

2 -16 3
This clause has already appeared.

2 -17 3
An integer should appear here.

2 -18 2
The memory size is incorrectly specified.

2 -19 2
The specified size of memory is larger than available.

2 -20 3
The "SEGMENT-LIMIT" clause has appeared previously.

2 -21 2
The reserved word "IS" should appear here.

2 -22 3
The segment limit cannot be greater than 49 nor can it be 0.

2 -23 3
Invalid "SEGMENT-LIMIT" clause.

2 -24 3
Invalid "ASSIGN" clause.

2 -25 3
The currency sign literal is invalid.

2 -26 3
The reserved word "COMMA" should appear here.

2 -27 3
The status of this switch was not specified.

2 -28 3
The status of this switch has already been specified.

2 -29 3
Invalid condition-name.

2 -30 3
This file has been selected previously.

2 -31 3
This file control entry does not specify an Internal File Name in an "ASSIGN" clause.

2 -32 3
Internal File Name incorrectly specified.

2 -33 3
The word "REEL" or "UNIT" should appear here.

2 -34 3
An integer should appear here.

2 -35 3
Invalid padding literal.

2 -36 3
The "PADDING" clause cannot be applied on this file.

2 -37 2
Duplicate "PADDING" clause for this file; first clause accepted.

2 -38 3
Invalid banner character.

2 -39 3
The "BANNER" clause cannot be applied on this file.

2 -40 3
A reserved word has been used as a user word or program-name is missing.

2 -41 6
This feature is a ^1 feature, not included in the current compilation level.

2 -42 1
The computer-name should be "LEVEL-64".

2 -43 3
Invalid option in a file control entry.

2 -44 3
Duplicate character in alphabet-name specification.

2 -45 1
This word may be reserved in a compilation whose level is other than "DSA".

2 -46 3
Invalid input-output technique.

2 -47 3
Invalid key-name.

2 -48 3
Invalid file-name.

2 -49 3
The reserved word "ON" should appear here.

2 -50 3
Illegal use of plural.

2 -51 3
The reserved word "CHECKPOINT-FILE" should appear here.

2 -52 1
Syntax checking is resumed at this point, current paragraph is assumed to be "^".

2 -53 3
Number of alternate keys is limited to a maximum of 15.

2 -54 3
Number of secondary keys is limited to a maximum of 8.

2 -55 3
The file referenced in the "RERUN EVERY END OF REEL/UNIT" clause is not accessed sequentially.

2 -56 3
Invalid "RERUN" clause.

2 -57 3
Only one file-name was specified in the previous "SAME" clause.

2 -58 1
No syntax checking from the last diagnostic to this point.

2 -59 3
Invalid or missing mnemonic-name.

2 -60 3
The reserved word "FILE" should appear here.

2 -61 3
This file-name has appeared in a previous "SAME AREA" clause.

2 -62 3
This file-name has appeared in a previous "SAME RECORD AREA" clause.

2 -63 3
Another file is assigned to the same internal-file-name as this file.

2 -64 3
Invalid access mode.

2 -65 3
The reserved word "SEGMENT" should appear here.

2 -66 3
Invalid file-name in this file control entry.

2 -67 1
Incorrect order of clauses in this file control entry.

2 -68 3
The reserved word "MULTIPLE" should appear here.

2 -69 3
Invalid "ORGANIZATION" clause.

2 -70 3
The reserved word "STATUS" should appear here.

2 -71 6
This feature (^2 in ^3) is a ^1 feature, not included in the current compilation level.

2 -72 2
Duplicate "BANNER" clause for this file; first clause accepted.

2 -73 3
The "OPTIMIZE" clause cannot be applied on this file.

2 -74 2
Duplicate "NO-RESIDENT-INDEX" clause for this file.

2 -75 3
A "KEY" clause is required for this file.

2 -76 6
This feature (^2) is a ^1 feature, not included in the current compilation level.

2 -77 1
Incorrect order of I-O-Control clauses.

2 -78 1
Incorrect order of Object-computer clauses.

2 -79 1
Incorrect order of Special-names clauses.

2 -80 3
Unrecognized or misplaced "DECIMAL-POINT" clause.

2 -81 2
The file was already referenced in a previous "RERUN" clause; first clause accepted.

2 -82 3
The alphabet-name is missing or incorrectly specified.

2 -83 3
This file has already appeared in a "MULTIPLE FILE" clause.

2 -84 3
A too high position number is specified for this file.

2 -85 3
Invalid data-name qualifier.

2 -86 3
Invalid implementor-name.

2 -87 3
Invalid "COLLATING SEQUENCE" clause.

2 -88 3
The reserved word "RECORD" should appear here.

2 -89 3
The word "1974" should appear here.

2 -90 1
Incorrect order of Source-computer clauses.

2 -91 3
Invalid "SEGMENT SIZE" clause.

2 -92 3
The reserved word "FILE" or "COMMUNICATION" should appear here.

2 -93 3
This feature is a ^1 feature, not included in the current compilation level; the clause is not accepted.

2 -94 3
Error at character ^2 in assign literal (^1).

2 -95 3
Illegal use of singular.

2 -96 3
Configuration Section is not allowed in the Environment Division of a contained program.

2 -97 3
This clause cannot be recognized.

2 -98 3
The reserved word "SELECT" should appear here.

2 -99 3
This feature is not implemented.

2 -100 2
Non standard internal-file-name suffix.

2 -101 2
This clause is used for documentation only.

2 -102 6
The JCL "^1" option is a NON STANDARD feature, not included in the current compilation level.

2 -103 3
Syntax error.

2 -104 3
The data segment size may not exceed 4M bytes.

2 -105 3
Alphabet-name already declared.

2 -106 3

The alphabet-name referenced in the program collating sequence clause is not declared hereafter.

2 -107 3

The Substitution Section has not been executed.

2 -108 5

Duplicate "SYMBOLIC QUEUE" clause.

2 -109 5

Duplicate "TEMP" clause; first clause accepted.

2 -110 5

Duplicate "ACCEPT" clause; first clause accepted.

2 -111 5

Duplicate "DISPLAY" clause; first clause accepted.

2 -112 3

The reserved word "DATA" should appear here.

2 -113 3

The reserved word "TIMES" should appear here.

2 -114 3

The value in "TEMP" clause must range between 13 and 30 inclusively.

2 -115 4

Compiler error: subroutine stack overflow.

2 -116 3

Index file is incorrectly specified.

2 -117 3

Invalid file for "MULTIPLE FILE" clause.

2 -118 2

^ is the internal-file-name given to this file.

2 -119 3
The members of a multiple file must have the same device and organization.

2 -120 3
More than one member of a multiple file are given the same position.

2 -121 3
The reserved word "INDEX" should appear here.

2 -122 3
Optional file must be of sequential organization.

2 -123 2
Non sequential organization optional file is a LEVEL-62 feature.

2 -124 2
This may be a LEVEL-62 internal device designator.

2 -125 2
This is a LEVEL-62 feature.

2 -126 2
This LEVEL-62 feature is ignored.

2 -127 3
The reserved word "CONSOLE" should appear here.

2 -128 3
The reserved word "STANDARD" should appear here.

2 -129 2
A LEVEL-62 switch-status-name assumed, the RERUN clause is ignored.

2 -130 3
This LEVEL-62 feature is not implemented.

2 -131 4
The interactive mode is not available on your site, please contact supplier.

2 -132 4

Implementation restriction. Too many files are referenced in this "SAME" clause; syntax checking cannot be fulfilled.

2 -133 3

Text is not allowed after the program-name in the PROGRAM-ID paragraph.

2 -134 2

Period is missing after the previous paragraph header.

2 -135 3

The FILE-CONTROL paragraph header should appear here.

2 -136 3

The optional "COMMON" clause may be used only if the program is contained within another program.

2 -137 7

This feature is temporarily kept in the COBOL Standard. It will disappear from a future version of the Standard.

2 -138 3

Invalid alphabet-name.

2 -139 3

A suffixed hardware-name is missing or in error.

2 -140 3

Only "PROGRAM-ID" is allowed as a paragraph-name in the Identification Division of a contained program.

2 -141 3

The procedure segment size may not exceed 32K bytes.

2 -142 3

The procedure segment size may not exceed 64K bytes.

2 -143 3

Inconsistent values in the "BLOCK CONTAINS" clause.

2 -144 1

For compatibility reason the record description should be given using the FD entry "RECORD" clause.

2 -145 3

This file-name appeared more than once in this clause.

2 -146 3

The word "CONSOLE" possibly suffixed should appear here.

2 -147 3

Unrecognized or misplaced "CURRENCY SIGN" clause ignored.

2 -148 3

Non numeric literal or figurative constant expected here.

2 -149 3

Duplicate Internal File Name.

2 -150 1

The size of this data item is less than the size of the data item which it redefines.

2 -151 3

Block size must equal maximum record size.

2 -152 2

The specified block size is too small to contain the largest record of this file; the "BLOCK" clause will be ignored.

2 -153 1

The record format for this file is permitted only on tape.

2 -154 1

The record format for this file is permitted only on tape or disk.

2 -155 3

A "RECORD...DEPENDING..." clause is not permitted with this record format.

2 -156 1

A data record for this file is too large for the specified device.

2 -157 3

Invalid record format for CPL file.

2 -158 2

Record prefix incompatible with display to sysout.

2 -159 5

The "CODE-SET" clause given for an SSF file must specify either EBCDIC or ASCII alphabet.

2 -160 3

The "UNBANNERED" clause applies to H-2000 odd parity tape files only.

2 -161 2

Duplicate I-O techniques applied to this file; first clause accepted.

2 -162 3

File characteristics are different from the one of another file with the same IFN.

2 -163 3

Invalid catalogue-name.

2 -164 3

The reserved word "TEMPORARY" or "PERMANENT" should appear here.

2 -165 3

The "NO-SORTED-INDEX" clause applies only to indexed files described with alternate keys.

2 -166 5

Duplicate "DISPLAY SIGN IS" clause, the first clause was accepted.

2 -167 3

"LEADING" or "TRAILING" must be specified in the "DISPLAY SIGN IS" clause.

2 -168 5

Duplicate "COMP" clause, the first clause was accepted.

2 -169 3

The "COMP" clause is improperly stated.

2 -170 2

A FILLER item is missing to accommodate synchronization in the current redefinition: FILLER item is provided.

2 -171 2

The synchronization cannot be accommodated for all occurrences of this item.

2 -172 2

This feature is not implemented.

2 -173 2

The 01 or 77 level item is allocated in a segment whose size exceeds the specified or implied maximum segment size.

2 -174 2

Implementation restriction: too many items subordinate to this item or redefining it, space is only allocated for the size of the redefined 01 or 77 level item.

2 -175 3

The 01 level item has not the same length as the CD it implicitly redefines.

2 -176 2

The initial value will be entirely or partially overridden if the program is scheduled by the MCS ("CD FOR INITIAL INPUT").

2 -177 2

A "VALUE" clause cannot be specified for an 01 level entry that does not immediately follow a CD entry in the Communication Section.

2 -178 2

The size of the record might be too small if the file is assigned to a tape.

2 -179 3

Implementation restriction: too large 01 or 77 level item.

2 -180 2

The length of this record is greater than the maximum specified in the "RECORD" clause.

2 -181 2

The length of this record is inconsistent with the "RECORD" clause.

2 -182 3

The organization of this file contradicts the variable record format implied by the following definition.

2 -183 3

The specified code-set is not allowed with the file organization.

2 -184 2

The specified code-set (IBCD) is meaningful only if the file actually assigned at object-time is a tape file.

2 -185 3

The specified code-set is not implemented.

2 -186 2

The number of characters specified in the "BLOCK CONTAINS" clause is not a multiple of the record size.

2 -187 3

The "LINAGE" clause may only be used for an SSF file.

2 -188 5

A sort file cannot be referenced in a "MULTIPLE FILE" clause.

2 -189 1

The "CODE-SET IS IBCD" clause may only be used for a tape file when its organization is H-2000 sequential.

2 -190 2

The longest record of an SSF file must be at least 71 characters in length.

2 -191 3

Implementation restriction: "INDEXED BY" must not be used when either the element size or the repetition number is greater than 65500.

2 -192 2

The "CODE-SET" clause is not allowed with the file organization.

2 -193 4

Implementation restriction: no space enough to process the synchronization.

2 -194 3

Internal-file-name "H-SORT" is reserved for sort files.

2 -195 3

The file control entry for a sort file can only contain the "ASSIGN" (mandatory) clause with an internal-file-name possibly suffixed by -MSD and the non standard "FLR/VLR" option.

2 -196 3

The internal-file-name given for this file is not allowed for a sort file.

2 -197 2

The FILLER item inserted for synchronization was not taken care of in the redefinition.

2 -198 1

A ^1 ^2 type 1 FILLER item was added at the end of this item (see the *COBOL 85 Reference Manual*).

2 -199 1

A ^1 ^2 type 2 FILLER item was allocated to align this synchronized item (see the *COBOL 85 Reference Manual*).

2 -200 2

This "SELECT" has no corresponding "FD".

2 -201 2

This file has been opened but not closed.

2 -202 2

This file has been closed but not opened.

2 -203 2

This file was not opened in "INPUT" or "I-O" mode though it is referenced in a "READ" or a "START" statement.

2 -204 2

This file was not opened in the proper mode to be referenced in a "WRITE" statement.

2 -205 2

This file was not opened in "I-O" mode though it is referenced in a "REWRITE" or a "DELETE" statement.

2 -206 2

This file is not referenced in a "READ" statement though it is referenced in a "REWRITE" or a "DELETE" statement, and it is in sequential access.

2 -207 2

Only input files can be optional.

2 -208 2

The (maximum) size in the "RECORD" clause is greater than the size of the largest record described for this file; the former is taken as the maximum record size.

2 -209 3

The "LABEL" clause of this multiple file member is not the same as that of another member.

2 -210 2

"CODE-SET" clause illegal on non-sequential file.

2 -211 3

"CODE-SET" clause illegal on non-sequential file.

2 -212 3

This sort-file is referenced in more than one "SAME SORT" or "SAME SORT-MERGE" clause.

2 -213 3

This clause must contain at least one sort-file-name.

2 -214 3

A file-name of this "SAME AREA" clause appears in a "SAME RECORD", "SAME SORT" or "SAME SORT-MERGE" clause but all file-names in the "SAME AREA" clause do not appear in that other clause.

2 -215 3

The size of this data item must not exceed the size of the data item which it redefines.

2 -216 3

This is a COBOL C reserved word.

2 -217 3

An identifier must not appear more than once in a "USING" phrase.

2 -218 5

A sort-file must not appear in a "SAME" clause when the "SORT", "SORT-MERGE" or "RECORD" phrase is not used.

2 -219 1

Previous definition of this item cannot be referenced.

2 -220 3

Both this item and a previous definition thereof are referenced in the "USING" phrase of the Procedure Division header.

2 -221 1

The size of this data item should not exceed the size of the data item that it redefines.

2 -222 2

This item is not referenced in the "USING" phrase of the Procedure Division header.

2 -223 4

Implementation restriction. No room enough for the allocation phase.

2 -224 3

The (implicit or explicit) first stack page is not large enough.

2 -225 3

This "SELECT" has no corresponding "FD".

2 -226 2

This file whose organization is "QUEUED" should be referenced in an "ASSIGN ... TO MEMBER" statement since it is referenced in an "OPEN" statement.

2 -227 2

Output files cannot be optional.

2 -228 3

The data description entry for this name must not contain a "REDEFINES" clause.

2 -229 3

The same external file was defined with two different IFN.

2 -230 3

Duplicate "ACCEPT ALTERNATE-CONSOLE" clause, the first clause was accepted.

2 -231 3

Duplicate "ACCEPT CONSOLE" clause, the first clause was accepted.

2 -232 3

Duplicate "SYSIN" clause, the first clause was accepted.

2 -233 3

Duplicate "ACCEPT TERMINAL" clause, the first clause was accepted.

2 -234 3

Duplicate "DISPLAY ALTERNATE-CONSOLE" clause, the first clause was accepted.

2 -235 3

Duplicate "DISPLAY CONSOLE" clause, the first clause was accepted.

2 -236 3

Duplicate "SYSOUT" clause, the first clause was accepted.

2 -237 3

Duplicate "DISPLAY TERMINAL" clause, the first clause was accepted.

2 -238 3

Duplicate "COBOL 1974" clause, the first clause was accepted.

2 -239 3

A report file may be referenced in a "SAME" clause only if the "RECORD" phrase is not specified.

2 -241 1

^.

2 -242 2
^.

2 -243 3
^.

2 -244 4
^.

2 -245 5
^.

2 -246 6
^.

2 -247 7
^.

2 -249 3
If the "RECORD" clause is not specified for an external file, all associated Record Descriptions must be the same length.

2 -250 3
The length of this record is inconsistent with the maximum specified in the "RECORD" clause.

2 -251 3
The length of this record is inconsistent with the minimum specified in the "RECORD" clause.

2 -252 3
The "RECORD DELIMITER" clause may be specified only for files whose record type is explicitly or implicitly variable length.

2 -253 3
The class name is missing or incorrectly specified.

2 -254 3
Integer from 1 to 256 or non numeric literal expected here.

2 -255 1
Duplicate character specified in class definition.

2 -256 3

Integer from 1 to 256 or 1-character non-numeric literal expected here.

2 -257 3

A user-defined word should appear here.

2 -258 3

Alphabet name expected here.

2 -259 3

Symbolic character already declared.

2 -260 3

There is no character at this position in the NATIVE character set.

2 -261 3

There is no character at this position in the specified character set.

3 -1 3
A valid section name was expected in area A.

3 -2 2
Reserved word "SECTION" is missing.

3 -3 2
Period expected after the previous word.

3 -4 3
Redundant File Section was detected, only one File Section is allowed per program.

3 -5 3
Sections precedence syntax error is detected, check the *COBOL 85 Reference Manual* for correction.

3 -6 3
Unrecognizable File Section level indicator has occurred, it must be "FD" or "SD".

3 -7 5
Only right justification may be specified.

3 -8 3
The reserved word "GLOBAL" should appear here.

3 -9 3
In the present data entry, the following data properties are inconsistent with ^1: ^2.

3 -10 3
File-name is not defined in Environment Division.

3 -11 3
Unrecognizable FD clause is encountered or period is missing.

3 -12 3
The reserved word "TERMINAL" should appear here.

3 -13 3
The reserved word "VARYING" should appear here.

3 -14 3
File recording mode name is in error.

3 -15 1
"CHARACTERS" option is assumed for the "BLOCK" clause.

3 -16 3
Maximum block size (unsigned positive integer) is missing or incorrectly specified.

3 -17 3
Maximum record size (unsigned positive integer) is missing or incorrectly specified.

3 -18 2
Reserved word "RECORD" is missing.

3 -19 2
Reserved word "OF" is missing.

3 -20 3
Data-name is missing or in error.

3 -21 3
Literal (unsigned positive integer) or data-name is missing or incorrectly specified.

3 -22 3
Redundant "TOP" phrase is specified for "LINAGE" clause.

3 -23 3
LINAGE specification is in error.

3 -24 3
Duplicate name in "RECORDS" or "REALMS" clause.

3 -25 3
Literal (unsigned positive integer) is missing or incorrectly specified.

3 -26 3
Redundancy of Sub-Schema Section is detected, only one is allowed per program.

3 -27 3
Redundancy of Working-Storage Section is detected, only one is allowed per program.

3 -28 3
Unrecognizable level or section indicator has occurred.

3 -29 3
The "REDEFINES" clause when used must immediately follow the subject of "REDEFINES".

3 -30 3
A too high value is specified.

3 -31 3
The subject of an "OCCURS" with the "DEPENDING" option must be the last group or elementary item in the record, it cannot be followed by an item of equal or less level number.

3 -32 3
The object of "REDEFINES" is not found at equal level, or is itself the subject of a "REDEFINES" clause.

3 -33 3
This word should begin in area A.

3 -34 3
The 66 level item cannot follow a 77 level item.

3 -35 3
An unrecognizable data attribute is encountered, or period is missing.

3 -36 3
The condition-name must immediately follow the 88 level number.

3 -37 3
The value specified for the condition-name is in error.

3 -38 3
The level number for this data item is improper, it should be 77 or 01.

3 -39 3
A 88 level condition-name item cannot be associated with a usage DB-KEY data item.

3 -40 3
A 88 level condition-name item cannot be associated with an index data item.

3 -41 6
This feature is a ^1 feature, not included in the current compilation level.

3 -42 3
The object of "REDEFINES" data-name is not specified.

3 -43 3
Redundant "REDEFINES" clause is detected, only one is allowed per data item.

3 -44 3
Redundant "PICTURE" clause is detected, only one is allowed per data item.

3 -45 3
Redundant "USAGE" clause is detected, only one is allowed per item.

3 -46 3
Redundant "VALUE" clause is detected, only one is allowed per item.

3 -47 3
"OCCURS" clause cannot be declared on a level 01 or 77 item, nor can it be redundant.

3 -48 3
Redundant "JUST" clause is detected, only one is allowed per item.

3 -49 3
Redundant "BLANK WHEN ZERO" clause is detected, only one is allowed per data item.

3 -50 3
Redundant "SYNC" clause is detected, only one is allowed per data item.

3 -51 3
Redundant "SIGN" clause is detected, only one is allowed per data item.

3 -52 3
Redundant "RENAMES" clause is detected, only one is allowed per item.

3 -53 3
Sign is leading or trailing is not specified.

3 -54 3
The object of "REDEFINES" may not have an "OCCURS" clause.

3 -55 3
The object of "REDEFINES" data-name cannot be an item of variable length.

3 -56 3
The subject of "REDEFINES" in File Section or Communication Section cannot be a 01 level item, the redefinition is implied.

3 -57 3
Level indicator DB is missing or in error.

3 -58 3
Schema-name is missing or in error.

3 -59 3
The subject of "RENAMES" is not specified.

3 -60 3
The "RENAMES" clause is missing for the 66 level item.

3 -61 3
An unrecognizable DB clause is encountered or period is missing.

3 -62 3
Redundant "DB-DESCRIPTIONS" clause is detected, only one is allowed per DB entry.

3 -63 3

Redundant "RECORDS" or "REALMS" clause is detected, only one of "RECORDS" or "REALMS" is allowed per DB entry.

3 -64 3

Reserved word "THRU" is missing.

3 -65 3

"LABEL" clause is missing in the current FD entry.

3 -66 3

The PICTURE character-string is missing.

3 -67 3

"REPORT" clause and "DATA RECORD" clause are mutually exclusive.

3 -68 3

The initial value is redundantly specified; when the group item already has initial value specified, the subordinate item cannot have additional initial value.

3 -69 3

Reserved word "WORKING-STORAGE" or "LINKAGE" is missing.

3 -70 3

The usage of a subordinate item must be consistent with that of the group item.

3 -71 3

When the group data item has initial value, the subordinate item cannot have usage other than DISPLAY.

3 -72 3

Only one Sub-Schema Section with one DB entry is allowed.

3 -73 3

When the group data item has initial value, the subordinate item may not have the "JUSTIFIED" clause.

3 -74 1

"SYNC LEFT" is assumed.

3 -75 3

When the group data item has initial value, the subordinate item cannot contain "SYNC" clause.

3 -76 6

This feature (^2) is a ^1 feature, not included in the current compilation level.

3 -77 3

The "SIGN" clause is redundantly specified; when the group item has "SIGN" clause, it is implied to the subordinate item.

3 -78 4

Compiler error: working space exhausted.

3 -79 3

Reserved word "ZERO" is missing.

3 -80 3

When the group item is associated with 88 condition items, the subordinate items cannot contain "JUST" clause.

3 -81 3

When the group item is associated with 88 condition items, the subordinate items cannot contain "SYNC" clause.

3 -82 3

When the group item is associated with 88 condition items, the subordinate items cannot have usage other than DISPLAY.

3 -83 3

The dimension of "OCCURS" cannot exceed 3.

3 -84 3

The occurrence times is not correctly specified.

3 -85 3

The occurrence times cannot be 0.

3 -86 3

The maximum occurrences must be greater than the minimum occurrences.

3 -87 3
"REPORT" clause and "LINAGE" clause are mutually exclusive.

3 -88 3
When the group item contains "OCCURS" clause, the subordinate item cannot be of variable length.

3 -89 3
The index-name is missing.

3 -90 3
The option "LINKAGE" is not allowed in a contained program.

3 -91 3
The record name must be specified for a record associated with a File Description entry that contains the "EXTERNAL" or "GLOBAL" clause.

3 -92 3
Redundant "INDEXED BY" clause is detected, only one is allowed per data item.

3 -93 2
The qualification of object of "REDEFINES" data-name is for documentation only.

3 -94 3
Communication Section precedence error.

3 -95 4
Implementation restriction. No room enough to hold the index-name table.

3 -96 5
Redundant Communication Section is detected, only one is allowed per program.

3 -97 3
Level indicator CD is missing or in error.

3 -98 3
CD-name is missing.

3 -99 3
"INPUT" or "OUTPUT" option must be specified for each CD entry.

3 -100 3
Only one "INITIAL" clause is allowed in the Communication Section.

3 -101 3
When neither option is used, the CD entry must be followed by one or more 01 record descriptions.

3 -102 3
Excess data-names are specified for the current CD entry.

3 -103 3
An unrecognizable CD level indicator or section header is encountered.

3 -104 3
Unrecognizable input CD clause is encountered.

3 -105 3
Redundant input CD "SYMBOLIC SUBQUEUE-1" clause.

3 -106 3
Redundant input CD "SYMBOLIC SUBQUEUE-2" clause.

3 -107 3
Redundant input CD "SYMBOLIC SUBQUEUE-3" clause.

3 -108 3
Redundant input CD "SYMBOLIC QUEUE" clause.

3 -109 3
Redundant input CD "MESSAGE DATE" clause.

3 -110 3
Redundant input CD "MESSAGE TIME" clause.

3 -111 3
Redundant input CD "TEXT LENGTH" clause.

3 -112 3
Redundant input CD "END KEY" clause.

3 -113 3
Redundant input CD "STATUS KEY" clause.

3 -114 3
Too many occurrences specified.

3 -115 3
Redundant input CD "SYMBOLIC SOURCE" clause.

3 -116 3
Redundant input CD "MESSAGE COUNT" clause.

3 -117 3
Unrecognizable output CD clause is encountered.

3 -118 3
The maximum occurrence number must be numeric unsigned integer greater than 0.

3 -119 3
Redundant output CD "DESTINATION COUNT" clause.

3 -120 3
Redundant output CD "TEXT LENGTH" clause.

3 -121 3
Redundant output CD "STATUS KEY" clause.

3 -122 5
Redundant output CD "DESTINATION TABLE" clause.

3 -123 3
Redundant output CD "ERROR KEY" clause.

3 -124 3
Redundant output CD "SYMBOLIC DESTINATION" clause.

3 -125 3
Unrecognizable input-output CD clause is encountered.

3 -126 2
Destination table may occur only 1 time in this implementation.

3 -127 3
Redundant input-output CD "MESSAGE DATE" clause.

3 -128 3
Redundant input-output CD "MESSAGE TIME" clause.

3 -129 3
Redundant input-output CD "SYMBOLIC TERMINAL" clause.

3 -130 5
Data-name is missing or in error.

3 -131 3
Redundant Linkage Section is detected, only one is allowed per program.

3 -132 3
Redundant input-output CD "TEXT LENGTH" clause.

3 -133 3
Redundant Constant Section is detected, only one is allowed per program.

3 -134 3
Redundant input-output CD "END KEY" clause.

3 -135 3
The usage specified is unrecognizable.

3 -136 3
Redundant input-output CD "STATUS KEY" clause.

3 -137 3
"CODE-SET" clause redundant in FD or illegal in SD.

3 -139 3
"RECORDING MODE" clause redundant in FD or illegal in SD.

3 -140 3
"BLOCK CONTAINS" clause redundant in FD or illegal in SD.

3 -141 3
"RECORD" clause redundant in FD or SD.

3 -142 3
"LABEL RECORD" clause redundant in FD or illegal in SD.

3 -143 3
"VALUE OF" clause redundant in FD or illegal in SD.

3 -144 3
"DATA RECORD" clause redundant in FD or SD.

3 -145 3
"REPORT IS" clause redundant in FD or illegal in SD.

3 -146 3
"LINAGE IS" clause redundant in FD or illegal in SD.

3 -147 3
The "EXTERNAL" clause in an FD entry is not compatible with the non standard "EXTERNAL" clause in the related file control entry.

3 -148 3
The specification in the "LABEL RECORD" clause is unrecognizable.

3 -149 3
Duplicate name in "REPORT" clause.

3 -150 3
The current section is assumed to be File Section.

3 -151 3
The current section is assumed to be Working-Storage Section; please disregard the irrelevant diagnostics if any.

3 -152 3
When the clause "DB-DESCRIPTIONS IN LINKAGE" is present, "DB-CXT" must appear in the "USING" phrase of the Procedure Division header.

3 -154 1
Syntax checking is resumed at this point.

3 -155 3

A report file must not have the "DEPENDING" option in the "RECORD" clause.

3 -156 1

Record description is missing, syntax checking is resumed at this point.

3 -157 3

A "LINAGE" or "REPORT" clause cannot apply to a file whose organization is not sequential.

3 -158 3

A report file should not have record description.

3 -159 3

The data description clauses in this entry have fatal error.

3 -160 3

Reserved word "KEY" is missing.

3 -161 1

SSF format is assumed with a "LINAGE" or a "REPORT" clause.

3 -162 3

The literal following the "THRU" option is missing.

3 -163 3

The level-66 entry is not properly positioned, it must immediately follow the last data entry of the logical record.

3 -164 3

Area-name is not defined.

3 -165 3

SD-name is not defined.

3 -166 3

The record description for the previous file description entry is missing.

3 -167 3

File-name is missing or in error.

3 -168 3

The record prefix specified in the file control entry conflicts with a "LINAGE" or "REPORT" clause.

3 -169 3

A file referenced in a "SAME" clause cannot be external.

3 -170 3

The "GLOBAL" clause must not be specified when the file is referenced in a "SAME RECORD AREA" clause.

3 -171 3

The "VARYING" phrase in the "RECORD" clause is not allowed for a Report file.

3 -172 3

Unrecognizable attribute in CD entry is encountered.

3 -173 3

Reserved word "LENGTH" is missing.

3 -174 3

Reserved word "TOP" or "BOTTOM" is missing.

3 -175 3

Redundant "BOTTOM" phrase is specified for "LINAGE" clause.

3 -176 3

Redundant "FOOTING" phrase is specified for "LINAGE" clause.

3 -177 3

Literal (unsigned integer) or data-name is missing or incorrectly specified.

3 -178 3

Reserved word "FOOTING" is missing.

3 -179 3

The footing integer must not be greater than the body integer in "LINAGE" clause.

3 -180 1

The "THRU" option should be used for a binary floating point numeric item.

3 -181 2

In the given "VALUE" clause, second value is not greater than first.

3 -182 2

In the given "VALUE" clause, value may be longer than length of data item.

3 -183 2

In the given "VALUE" clause, unsigned item has signed value.

3 -184 3

In the given "VALUE" clause, numeric data item has non-numeric value.

3 -185 3

In the given "VALUE" clause, non-numeric data item has numeric value.

3 -186 3

Invalid code-set specified.

3 -187 3

Redundant "GLOBAL" clause is detected, only one is allowed per "DB" entry.

3 -188 3

When FD has JIS code-set, signed numeric data must have "SIGN IS SEPARATE" clause.

3 -189 2

"RECORD" clause specifies variable length records, but file control entry specifies "FLR".

3 -190 3

The "SIGN" clause must be associated with at least one numeric item with PICTURE character-string containing "S".

3 -191 1

Record description assumed to be data record for the preceding FD.

3 -192 2

A label record specified in the File Section was not defined by a record description entry.

3 -193 2

A data record specified in a "DATA RECORD" clause was not subsequently defined by a record description entry.

3 -194 2

Label records for H_RD, H_PR must be standard and are so assumed.

3 -195 3

Integer cannot be signed.

3 -196 3

Integer cannot be signed nor can it be zero.

3 -197 3

Reserved word "DEPENDING" is missing.

3 -198 1

In compliance with Standard: "CODE-SET" clause on FD should be accompanied by "SIGN IS SEPARATE" for signed numeric data.

3 -199 3

When FD has JIS code-set, data usage must be DISPLAY.

3 -200 1

In compliance with Standard: "CODE-SET" clause on FD should be accompanied by all data usage DISPLAY.

3 -201 3

Only numeric literals are allowed in the "LINAGE" clause for an external file.

3 -202 2

A level number must be the first word of the line.

3 -203 3

A reserved word has been used as a user word or data-name is missing.

3 -204 1
"VALUE OF" clause is used for documentation.

3 -205 3
"ALL" may not be used with a numeric literal.

3 -206 2
This word should begin in area A.

3 -207 5
Illegal use of singular.

3 -208 5
Inconsistent use of singular and plural.

3 -209 3
This feature is not implemented.

3 -210 3
Illegal use of plural.

3 -211 2
Too many record-names in "DATA RECORDS" clause.

3 -212 2
Reserved word "DIVISION" is missing.

3 -213 3
Level number hierarchy incorrect.

3 -214 3
The number of digit positions specified for this item exceeds the maximum allowed.

3 -215 2
The record clause specifies too large a record size.

3 -216 5
Inconsistent values in the "RECORD" clause.

3 -217 2
Inconsistent values in the "BLOCK CONTAINS" clause.

3 -218 3

A PICTURE character-string with a scaling factor is not compatible with the usage of this data item.

3 -219 3

Only usage DISPLAY is allowed when organization is H-2000 or ANSI.

3 -220 2

The "SIGN" clause cannot apply to any contained numeric usage DISPLAY elementary item (if any).

3 -221 3

The "SIGN" clause must be associated with usage DISPLAY and PICTURE character-string containing "S".

3 -222 3

The initial value specified is unrecognizable.

3 -223 3

The value is not in the range allowed for the item.

3 -224 2

Too many items subordinate to conditional variable: value size will not be checked.

3 -225 3

This LEVEL-62 feature is not implemented.

3 -226 2

This is a LEVEL-62 feature.

3 -227 6

This feature ("FILLER" at group level) is a NON STANDARD feature not included in the current compilation level.

3 -228 2

The "LABEL" clause is missing. Optional "LABEL" clause is a LEVEL-62 feature.

3 -229 3

Redundant "EXTERNAL" clause is detected, only one is allowed per data item.

3 -230 3

Only Working-Storage or Constant Section 01 or 77 non FILLER entries without "REDEFINES" can have the "EXTERNAL" clause.

3 -231 6

This feature (missing data-name) is a NON STANDARD feature not included in the current compilation level.

3 -232 2

This LEVEL-62 feature is not implemented. Subsequent entries are assumed to be Working-Storage.

3 -233 3

The "GLOBAL" clause may be specified for a data-name if its level is 01 only.

3 -234 3

Reserved word "WITHIN" is missing.

3 -235 3

DB-name is missing or in error.

3 -236 3

The specified code-set is not allowed with the file organization.

3 -237 1

The "CODE-SET IS IBCD" clause may only be used for a tape file when its organization is H-2000 sequential.

3 -238 3

A "VALUE" clause is missing for this 88 entry.

3 -239 1

Signed PICTURE character-string is assumed for this fixed binary item.

3 -240 2

A long fixed binary item is limited to 9 decimal digits.

3 -241 3

In the given "VALUE" clause, Boolean item has non Boolean value.

3 -242 3

In the given "VALUE" clause, non-Boolean item has Boolean value.

3 -243 3

The "THRU" phrase is not allowed with a Boolean data item.

3 -244 1

A ^-bit type 2 FILLER item was allocated before this item.

3 -245 1

A ^-bit type 3 FILLER item was allocated to align this synchronized item (see the *COBOL 85 Reference Manual*).

3 -246 3

An elementary item with usage BIT cannot be the object of a "REDEFINES" clause.

3 -247 3

The reserved word "EXTERNAL" or "GLOBAL" should appear here.

3 -248 3

A fixed binary data item is limited to 18 decimal digits.

3 -249 3

Too large item.

3 -250 3

Implementation restriction: integer in the "LINAGE" clause must not be greater than 2147483646.

3 -251 3

The number of digits specified for the exponent exceeds the maximum allowed.

3 -252 3

The object of "REDEFINES" is not byte aligned, first ^ bits cannot be overlayed.

3 -253 1

A ^-bit type 2 FILLER item was allocated after this item.

3 -254 7

This feature is temporarily kept in the COBOL Standard. It will disappear from a future version of the Standard.

3 -255 3

This clause may be specified in either the file control entry or the FD entry but it may not appear in both locations for a given file.

3 -256 3

The "SECONDARY KEY" clause is illegal in SD.

3 -257 7

This feature ("ALL literal" may be associated with a numeric or numeric-edited data item when the length of literal is greater than one) is temporarily kept in the COBOL Standard. It will disappear from a future version of the Standard.

3 -258 3

Non numeric literal or figurative constant expected here.

3 -259 3

"EXTERNAL" clause redundant in FD or illegal in SD.

3 -260 3

"GLOBAL" clause redundant in FD or SD.

3 -261 3

"FILE STATUS" clause redundant in FD or illegal in SD.

3 -262 3

"RECORD KEY" clause redundant in FD or illegal in SD.

3 -263 3

"ALTERNATE RECORD KEY" illegal in SD.

3 -264 3

"ACCESS MODE" clause redundant in FD or illegal in SD.

3 -265 3

Reserved word "STATUS" is missing.

3 -266 3

This clause may be used for indexed files only.

3 -267 3

The number of alternate keys is limited to a maximum of 15.

3 -268 3

Invalid access mode.

3 -269 3

This clause may be used for non sequential files only.

3 -270 3

This clause may be used for relative files only.

3 -271 3

This clause may be used for keyed files only.

3 -272 3

This clause may be used for extended indexed files only.

3 -273 3

The number of secondary keys is limited to a maximum of 8.

3 -274 3

Index file is incorrectly specified.

3 -275 1

"COMP-1" item is limited to 4 decimal digits. Usage "COMP-2" is assumed.

3 -276 3

"VALUE NULL" must be used for a usage POINTER elementary item only.

3 -277 5

When the group data item has initial value, the subordinate entry cannot contain a "SYNC", "JUST", "USAGE" (other than "DISPLAY") or initial "VALUE" clause.

3 -278 5

Another file is assigned to this internal-file-name.

3 -279 5
Ambiguous reference.

3 -280 3
Redundant "GLOBAL" clause is detected, only one is allowed per data item.

3 -281 3
The value in the "FALSE" phrase is not in the range allowed for the item.

3 -282 2
In the given "FALSE" phrase, value may be longer than length of data item.

3 -283 2
In the given "FALSE" phrase, unsigned item has signed value.

3 -284 3
In the given "FALSE" phrase, numeric data item has non-numeric value.

3 -285 3
In the given "FALSE" phrase, non-numeric data item has numeric value.

3 -286 5
In the present data entry, the following data property is inconsistent with numeric edited: "SIGN" clause exists.

3 -287 3
In the present data entry, the following data property is inconsistent with numeric edited: "SIGN" clause exists.

3 -288 3
The reserved word "FALSE" should appear here.

3 -289 3
This value must not equal the one specified in the "FALSE" phrase.

3 -290 3
This range of values must not include the one specified in the "FALSE" phrase.

3 -291 3

Period expected after the previous word.

3 -292 3

"VALUE IS NULL" is the only "VALUE" clause that can be specified for a usage POINTER elementary item.

3 -293 3

The "FALSE" phrase cannot be specified for a usage POINTER elementary item.

3 -295 3

Reserved word "USING" is missing.

3 -296 3

Sub-schema name is missing or in error.

3 -297 2

The clauses are not in the order stated in the CD entry format. The compiler may abort when the 'DEBUG' option is used.

3 -299 3

In the given "FALSE" phrase, Boolean item has non-Boolean value.

3 -300 3

In the given "FALSE" phrase, non-Boolean item has Boolean value.

3 -301 1

^.

3 -302 2

^.

3 -303 3

^.

3 -304 4

^.

3 -305 5

^.

3 -306 6
^.

3 -307 7
^.

4 -1 2
Expected word is "SECTION".

4 -2 2
Period is missing.

4 -3 3
RD entry is not given for a report specified in "REPORT" clause in FD.

4 -4 2
Level indicator "RD" or level number "01" should begin in Area A.

4 -5 2
This item should be written in Area B.

4 -6 2
Syntax checking discontinued from this item.

4 -7 2
Syntax checking is resumed.

4 -8 4
Compiler error. End of Data Division was detected.

4 -9 3
In the present entry, ^1 is inconsistent with the following property: ^2.

4 -10 2
No section can follow Report Section.

4 -11 3
Level indicator "RD" is missing.

4 -12 3
Level number "01" is missing.

4 -13 3
Level number or level indicator is expected after ".".

4 -14 3
Illegal level number.

4 -15 3
Level number unmatched.

4 -16 3
This clause is already specified.

4 -17 3
Illegal word in RD entry.

4 -18 3
Illegal word in 01 entry.

4 -19 3
Illegal word in report item description.

4 -20 3
Non signed integer is expected.

4 -22 3
Qualifier is missing after "IN" or "OF".

4 -23 3
")" is missing.

4 -24 3
Integer is missing in relative indexing.

4 -25 3
Subscripted reference is not allowed.

4 -26 3
Report-name cannot be qualified.

4 -27 3
Report group description should be within 3 levels.

4 -30 4
Implementation restriction. No space enough to accommodate this report description.

4 -31 4
Implementation restriction. No space enough to accommodate this report description.

4 -32 3
This feature is not implemented.

4 -33 5
Inconsistent use of singular and plural.

4 -34 3
Illegal use of plural.

4 -38 2
"FINAL" should be the first control level.

4 -39 3
Duplicate operand in "CONTROL" clause.

4 -40 3
Operand of "CONTROL" clause is missing.

4 -41 6
This feature is a ^1 feature, not included in the current compilation level.

4 -51 3
Syntax error in "PAGE" clause.

4 -52 2
The reserved word "DETAIL" is missing.

4 -53 3
Integer is missing in "PAGE LIMIT" clause.

4 -54 3
Integer is missing.

4 -55 3
Illegal integer.

4 -57 3
Relation between integers within "PAGE" clause is illegal.

4 -60 3
Report with "CODE" clause and report without "CODE" clause are mutually exclusive within a file.

4 -61 3
This report should have the "CODE" clause too.

4 -62 3
Code literal is missing.

4 -63 3
Literal of "CODE" clause should be of length 2.

4 -64 3
The reserved word "GLOBAL" should appear here.

4 -70 3
Syntax error in "USAGE" clause.

4 -71 2
No printable item subordinate to this item with the "USAGE" clause.

4 -73 2
Elementary report item with the "USAGE" clause should be printable item.

4 -76 6
This feature (^2) is a ^1 feature, not included in the current compilation level.

4 -80 3
Syntax error in "TYPE" clause.

4 -81 3
The reserved word "HEADING" or "FOOTING" is missing.

4 -82 3
PH or PF report group is not allowed for a report without "PAGE" clause.

4 -83 3
RH, PH, PF or RF report group should be defined at most once.

4 -84 3
Data-name or "FINAL" is missing within "TYPE" clause for type CH/CF report group.

4 -85 3
Type CH/CF report group should not appear in a report with no "CONTROL" clause.

4 -86 3
Control level cannot be defined for this group.

4 -87 3
CH or CF for a control level can be defined at most once.

4 -90 3
The reserved word "GROUP" is missing.

4 -91 3
Report group without line may have no "NEXT GROUP" clause.

4 -93 3
Syntax error in "NEXT GROUP" clause.

4 -94 3
Absolute "NEXT GROUP" clause may not appear in report without "PAGE" clause.

4 -95 3
Syntax error in next group integer.

4 -96 3
"NEXT GROUP" clause may not appear in PH or RF.

4 -97 3
"NEXT GROUP" clause may not appear in PF.

4 -100 3
Syntax error in "LINE" clause.

4 -101 3
Absolute "LINE" clause may not appear in report without "PAGE" clause.

4 -102 3
Illegal line integer.

4 -103 3
Absolute "LINE" clause should be in ascending order.

4 -104 3
Absolute "LINE" clause should precede relative "LINE" clause.

4 -105 3
"LINE" clause with "NEXT PAGE" should be the first "LINE" clause in a group.

4 -106 3
Line item should not be subordinate to line item.

4 -107 3
PF should begin with absolute line.

4 -108 3
"LINE" clause with "NEXT PAGE" may appear only within type, DE, CH, CF and RF report groups.

4 -109 3
A relative "LINE" clause must specify an integer greater than zero when it is the first "LINE" clause in a report group description.

4 -110 3
Syntax error in "PICTURE" clause.

4 -111 3
Illegal character in PICTURE character-string.

4 -113 3
The reserved word "ZERO" is missing after "BLANK".

4 -116 3
The number of digit positions specified for this item exceeds the maximum allowed.

4 -117 3

The number of digits specified for the exponent exceeds the maximum allowed.

4 -118 3

The "SIGN" clause may be specified only when the PICTURE character-string contains "S" but not "E".

4 -120 3

Column integer is missing.

4 -121 3

Column item without "LINE" clause should be subordinate to line item.

4 -122 3

This item overlaps previous item.

4 -123 3

Report record has insufficient size to print this item.

4 -124 3

Source operand is missing.

4 -125 3

"VALUE" clause operand is missing.

4 -126 3

Literal after "ALL" is missing.

4 -127 3

"VALUE" clause operand is inconsistent with item category.

4 -128 3

Reserved word "LEADING" or "TRAILING" expected.

4 -129 3

The "SEPARATE" phrase of the "SIGN" clause must be specified.

4 -130 3

"SUM" clause operand is missing.

4 -131 3
"SUM" clause should be specified within CF.

4 -132 3
"UPON" phrase operand is missing.

4 -133 3
"RESET" phrase operand is missing.

4 -134 3
Reset control level cannot be defined for this item.

4 -135 3
"SUM" clause operand should not be report item other than sum counter.

4 -136 3
Sum counter operand in "SUM" clause should be defined at lower or same control level.

4 -137 3
"UPON" phrase operand should be detail group within same report.

4 -138 3
"RESET" clause should specify higher or same control level.

4 -139 3
Multi-defined data-name within "SUM" clause or "UPON" phrase operand.

4 -140 3
Sum operand for "SUM" clause with "UPON" phrase should not be sum counter.

4 -141 3
The "COLUMN" clause should be specified when the "VALUE" clause is present.

4 -142 3
The "COLUMN" clause should be specified when the "SOURCE" clause is present.

4 -143 1
The "COLUMN" clause is not specified: this item is not printable.

4 -150 3
Report-name is missing.

4 -151 3
Report-name is not defined in any FD.

4 -152 3
Duplicate report description.

4 -160 3
"TYPE" clause is missing in report group description.

4 -161 3
No report group followed after RD entry.

4 -162 3
No body group appeared within this report.

4 -163 3
This report group violates upper limit rule for ^.

4 -164 3
This report group violates lower limit rule for ^.

4 -165 3
This ^ group cannot be presented on 1 page.

4 -166 3
This report group violates next group rule for ^.

4 -171 3
No subordinate item for format-2 item.

4 -172 3
No optional clause within format-2 item.

4 -173 3
Format-3 item may have only one of the "SOURCE", "SUM" or "VALUE" clauses.

4 -174 3
"PICTURE" clause is missing.

4 -175 3
No item can be subordinate to a format-3 item.

4 -176 3
"SOURCE" or "VALUE" clause is missing.

4 -177 3
"SOURCE", "SUM" or "VALUE" clause is missing.

4 -181 1
^.

4 -182 2
^.

4 -183 3
^.

4 -184 4
^.

4 -185 5
^.

4 -186 6
^.

4 -187 7
^.

5 -1 3
Twice the same file (cannot work at object time).

5 -2 3
Expected word was "DIVISION".

5 -3 3
Expected word was "." or "USING".

5 -4 3
"USING" not allowed with "INITIAL" clause in Data Division.

5 -5 3
Item is not 01 or 77 level data item defined in the Linkage Section

5 -6 3
A parameter should be defined in the Linkage Section of its own program.

5 -7 2
Period expected after this word (or identifier).

5 -8 3
Section header expected here.

5 -9 3
Expected word was "USE".

5 -10 3
Expected word was "BEFORE", "AFTER", "FOR" or "RANDOM".

5 -11 3
Expected word was "INPUT", "OUTPUT", "I-O", "EXTEND", or a file-name (except a sort or merge file-name)

5 -12 3
Item has "LABEL OMITTED" clause.

5 -13 3
This file-literal is invalid.

5 -14 3
Item is not report-group-name.

5 -15 3
Alphabet-name is unknown.

5 -16 3
Item is not numeric or numeric edited.

5 -17 3
"END COBOL" in wrong place.

5 -18 1
This option has no meaning in a virtual memory environment.

5 -19 3
Value out of range.

5 -20 3
Item is not paragraph or section declaration.

5 -21 3
Item is not identifier.

5 -22 3
Receiving item may not be alphabetic.

5 -23 3
Expected word was either the keywords "SYSIN, "CONSOLE", "DATE", ... or a mnemonic-name.

5 -24 3
Item is not elementary numeric.

5 -25 3
Item is not numeric.

5 -26 3
Item is not elementary.

5 -27 3
Item is not an alterable procedure-name.

5 -28 3
Item is not "TO".

5 -29 3
Item is neither a non-numeric literal nor an identifier.

5 -30 3
Item is not declared in File, Working-Storage, Communication or Linkage Sections.

5 -31 3
Item is not a non-sort file-name.

5 -32 3
Item is not "REWIND".

5 -33 3
Item is neither "=" nor the reserved words "FROM" or "EQUALS".

5 -34 3
Item is neither "INPUT" nor "OUTPUT".

5 -35 3
Item is not an output CD-name.

5 -36 3
Item is not an input CD-name.

5 -37 3
Item is not "KEY".

5 -38 3
Item is neither an alphanumeric identifier nor a literal.

5 -39 3
Item is neither an identifier nor a literal.

5 -40 3
Item is not a proper device.

5 -41 3
Expected word was "INTO" or "BY".

5 -42 3
"INVALID KEY" should not be used for that file.

5 -43 3
Item cannot be used in a "GENERATE" statement.

5 -44 3
Item is neither a procedure-name nor "DEPENDING".

5 -45 3
Item is not an elementary numeric integer.

5 -46 3
Error at character ^2 in assign-literal (^1).

5 -47 3
Item is not a report-name.

5 -48 3
Item is not an identifier of usage DISPLAY.

5 -49 3
Item is neither "TALLYING" nor "REPLACING".

5 -50 3
Item is not "FOR".

5 -51 3
Item is not "ALL", "LEADING" or "CHARACTERS".

5 -52 3
Item is neither a non-numeric literal nor an elementary data item with usage DISPLAY.

5 -53 3
Item is neither "ALL" nor "LEADING" nor "FIRST".

5 -54 3
Item is not "BY".

5 -55 3
Item is not "SEQUENCE".

5 -56 3
Item size is not equal to replaced item size.

5 -57 3
Item level is not "01" (or "77").

5 -58 3
Item is not "GIVING".

5 -59 3
Item is not "INPUT", "OUTPUT" or "I-O".

5 -60 3
File is not single reel/unit with sequential organization.

5 -61 3
"WRITE ADVANCING mnemonic-name" must not be used for a file described with the "LINAGE" clause.

5 -62 3
Item is not a procedure-name reference.

5 -63 3
Item is not "TIMES".

5 -64 3
Item is neither an elementary numeric item nor an index-name.

5 -65 3
Item is not "FROM".

5 -66 3
Item is not "UNTIL".

5 -67 3
Not accepted in declaratives.

5 -68 3
Item is not "CONVERSION".

5 -69 3
File description does not allow usage of "INTO".

5 -70 3
Item is same area as file-name.

5 -71 3
Item is not "MESSAGE" or "SEGMENT".

5 -72 3
Item is not "INTO".

5 -73 3
Item is not record-name in associated file.

5 -74 3
Item is not within sort input procedure range.

5 -75 3
Item is not within sort output procedure range.

5 -76 3
Item is not associated sort file.

5 -77 3
Item is not "ALL" or identifier.

5 -78 3
Item is not identifier or index-name.

5 -79 3
Item is not "WHEN".

5 -80 3
This reference should be a section reference.

5 -81 3
Item is not non-subscripted and non-indexed with "INDEXED BY" clause.

5 -82 3
Item does not have "KEY IS" clause.

5 -83 3
Expected word was "EOP".

5 -84 3
Item is not "ESI", "EMI", "EGI" or identifier.

5 -85 3
Misplaced report verb with regard to declaratives.

5 -86 2
Rules for transfer of control between procedures are violated.

5 -87 3
Item is neither an index-name identifier nor a positive integer.

5 -88 3
Item is neither an index-data-name nor a elementary integer.

5 -89 3
Item is not an integer.

5 -90 3
Item is not a sort file.

5 -91 3
Item is not "ASCENDING" or "DESCENDING".

5 -92 3
Item is not a fixed length scalar in associated file.

5 -93 3
Item is not "ON", "DESCENDING", "ASCENDING", "INPUT", "USING" or data-name.

5 -94 3
Item is not "OUTPUT" or "GIVING".

5 -95 3
Item is not of usage DISPLAY.

5 -96 3
Item is neither a non-numeric identifier nor a literal nor "DELIMITED".

5 -97 3
Item is neither a non-numeric identifier nor a literal nor "SIZE".

5 -98 3
Item is not fixed length with usage DISPLAY.

5 -99 3
Item is neither an identifier nor a non-numeric literal nor "INTO".

5 -100 3
Item should be a non-edited item with usage DISPLAY.

5 -101 3
Item is not an elementary numeric integer data item.

5 -102 3
Item is neither a numeric identifier nor a literal.

5 -103 3
Item is neither a numeric identifier nor a literal nor "FROM".

5 -104 3
The corresponding parameter of the called program is incompatible with this argument.

5 -105 3
Item is not alphanumeric.

5 -106 3
Item is neither "INTO" nor "DELIMITED".

5 -107 3
Item cannot be used without "DELIMITED".

5 -108 3
Item is not a record-name in a non sort file.

5 -109 3
Item is neither an identifier integer nor a mnemonic-name.

5 -110 3
"LINAGE" clause is missing in associated file.

5 -111 3
No "USE" applicable, so "INVALID" is mandatory.

5 -112 3
Item should be subscripted.

5 -113 3
Expected word was ")" (number of subscript levels exhausted).

5 -114 3
Item is not a proper subscript.

5 -115 3
Item should be index-name.

5 -116 3
Improper usage of an index.

5 -117 3
Item is not an unsigned non-zero integer-literal.

5 -118 3
The size of the composite of operands exceeds the allowed maximum in this arithmetic verb.

5 -119 3
Expected word was "ALL", "LEADING" or "UNTIL".

5 -120 3
Item is not single character literal or identifier with usage DISPLAY and class consistent with identifier.

5 -121 3
Expected word was "ALL", "LEADING", "UNTIL" or "FIRST".

5 -122 3
Expected word was "FIRST".

5 -123 2
Illegal comparison (non-numeric relation).

5 -124 3
Index data items may only be compared with indexes or index data items.

5 -125 2

This point can never be reached during execution.

5 -126 2

This statement may not be reached due to the previous "STOP RUN", "EXIT PROGRAM" or "GO".

5 -127 3

Item is neither "RUN" nor literal.

5 -128 3

Item is not an imperative verb.

5 -129 4

Compiler error. Expected word was "SECTION".

5 -130 3

Syntax error. Check along with the reference format.

5 -131 3

Paragraph (or section) name missing.

5 -132 3

Missing section header at beginning of Procedure Division.

5 -133 3

A new statement (or a period if appropriate) was expected here. Previous statement, if any, looked complete.

5 -134 3

Sentence must be imperative, this item makes it conditional.

5 -135 3

Expected word was "OVERFLOW".

5 -136 3

Expected word was "ERROR".

5 -137 3

Expected word was "DATA".

5 -138 3
The only allowed language-name is the reserved word "ESCAPE".

5 -139 3
Expected word was "END".

5 -140 3
Expected word was "COBOL".

5 -141 3
Item is not CD-name, identifier, procedure-name, file-name or "ALL".

5 -142 3
Item illegal in the scope of an "ENTER ESCAPE" statement.

5 -143 3
Expected word was "SIZE".

5 -144 3
Expected word was "NO" or "LOCK".

5 -145 3
Not allowed in imperative statement.

5 -146 3
This verb must be next to a procedure definition.

5 -147 2
Overlapping may occur between this receiving item and sending item.

5 -148 2
The receiving item may be truncated on the right.

5 -149 3
Numeric non-integer sending field not allowed with alphanumeric receiving field.

5 -150 2
Sign of sending item will not be moved to this item.

5 -151 3
Numeric sending field not allowed with alphabetic receiving field.

5 -152 3
Alphabetic sending field not allowed with numeric receiving field.

5 -153 3
Alphanumeric edited sending field not allowed with numeric receiving field.

5 -154 3
Numeric edited sending field not allowed with numeric receiving field.

5 -155 2
Possible right truncation.

5 -156 2
Possible left truncation.

5 -157 3
Expected word was "NO".

5 -158 3
Expected word was "REWIND".

5 -159 3
Item is not part of a condition.

5 -160 2
The result may be left truncated.

5 -161 3
Illegal relation between index and expression.

5 -162 6
This feature is a ^1 feature, not included in the current compilation level.

5 -163 3
Expected word was "INTO" or "END".

5 -164 1
Syntax checking discontinued.

5 -165 1
Syntax checking resumed.

5 -166 3
"USE" not permitted in non-declarative section.

5 -167 3
Data-names and indices not allowed together as subscripts.

5 -168 3
Imperative verb or "NEXT SENTENCE" expected here.

5 -169 3
File organization should be extended indexed.

5 -170 3
File is not an indexed file with random or dynamic access.

5 -171 4
Compiler error.

5 -172 3
Expected word was figurative constant or alphanumeric literal.

5 -173 3
Item is not alterable.

5 -174 3
Expected word was "POINTER".

5 -175 3
Item is not index-name, index data item, or elementary item described as an integer.

5 -176 3
Expected word was "TO", "UP", "DOWN" or an index-name.

5 -177 3
Expected word was "COMPL" or "COMPLEMENTARY".

5 -178 3
Item is not index-name, index data item, integer greater than zero or elementary item described as an integer.

5 -179 3
Item is not integer.

5 -180 3
Expected word was "WHEN", "AT" or "END".

5 -181 3
Item is not elementary alphabetic, alphanumeric, numeric edited or a group.

5 -182 3
Item is not figurative constant, non-numeric literal or identifier.

5 -183 4
Compiler error. Subroutine stack overflowed.

5 -184 1
This is a group move and operands do not have the same size.

5 -185 3
Item is not data-name.

5 -186 3
File is not sequential access or dynamic access.

5 -187 3
Item is not a one-character integer without an operational sign.

5 -188 3
Declarative portion cannot be referenced by non-declarative portion and vice versa.

5 -189 3
Expected word was "DUPLICATES".

5 -190 3
Expected word was "REMOVAL".

5 -191 3
Expected word was output CD-name.

5 -192 3
Item may not be Boolean.

5 -193 3
"ROUNDED" is not accepted in the "REMAINDER" phrase.

5 -194 3
Illegal subscript.

5 -195 3
Usage BIT arguments should be level 01.

5 -196 3
A reference modification is applicable only to usage DISPLAY items.

5 -197 3
Out of bounds leftmost character position.

5 -198 3
Colon is mandatory in reference modification.

5 -199 3
Too large length value (possibly due to too large leftmost character position).

5 -200 3
Expected word was "REFERENCE" or "CONTENT".

5 -201 3
Relation expected here.

5 -202 3
This operand should be numeric identifier.

5 -203 3
Relation or other condition operator expected here.

5 -204 3
")" is lacking.

5 -205 3
"(" matching this ")" is lacking.

5 -206 3

The previous operand should be neither numeric nor Boolean.

5 -207 3

CD-name expected here.

5 -208 3

The previous operand should be either an elementary non-alphabetic and non-Boolean usage DISPLAY data item or a group item whose subordinate elementary items are unsigned usage DISPLAY.

5 -209 3

Illegal relation (between two literals).

5 -210 3

This element is not valid beginning of condition.

5 -211 3

Item should be a key of the file.

5 -212 3

Verb or "NEXT SENTENCE" expected here.

5 -213 3

Expected word was "." or "ELSE".

5 -214 3

Operand missing.

5 -215 3

Subject of comparison missing.

5 -216 3

Expected word was "SENTENCE".

5 -217 3

This element is not valid condition.

5 -218 3

Expected word was data-name (or "TO" or "THAN" if appropriate).

5 -219 3
"DELETE" cannot be applied to a sequential file.

5 -220 3
Expected word was identifier or input CD-name

5 -221 3
Expected word was "COUNT".

5 -222 3
"WITH CONVERSION" is applicable only to elementary numeric data.

5 -223 3
Item does not reference an input device.

5 -224 3
Item does not reference an output device.

5 -225 2
This Constant Section item might be modified by the called procedure.

5 -226 3
Expected word was "LESS" or "<".

5 -227 3
File must be relative with a "RELATIVE KEY" phrase or indexed and must have sequential or dynamic access.

5 -228 3
Expected word was "EQUAL", "GREATER" or "NOT".

5 -229 3
Expected item was the data-name specified in the "RELATIVE KEY" phrase of the associated file control entry.

5 -230 3
Address of this item is not the same as the address of the record key.

5 -231 3
Expected item was "INITIAL" or a non-numeric literal or elementary data item whose usage is DISPLAY.

5 -232 3

This "MOVE" will abort object code (sending literal not digits).

5 -233 3

Expected item was "BY", "ALL", non-numeric literal, alphanumeric data item or any figurative constant except "ALL".

5 -234 3

Expected item was "OR" or "INTO".

5 -235 3

This identifier does not conform to the complex rules of the Language Standard.

5 -236 3

Integer out of range.

5 -237 3

The record-name in this statement must have an associated record prefix of "SSF".

5 -238 3

Expected word was non-numeric literal or alphanumeric identifier.

5 -239 3

Both procedure-names must be in the same declarative section.

5 -240 3

Expected word was an index-name, a positive integer or an elementary numeric integer data item.

5 -241 3

Expected word was a non-zero integer or an elementary numeric integer data item.

5 -242 3

Expected word was an index-name, literal or an elementary numeric data item.

5 -243 3

Expected word was an elementary numeric data item or a non-zero literal.

5 -244 3

The identifier following "VARYING" must be an elementary numeric integer data item.

5 -245 3

Sections in the declaratives must contain segment-numbers less than 50.

5 -246 3

This "PERFORM" statement does not conform to the complex rules of the Language Standard for segmentation.

5 -247 3

The segment-number must consist in one or two digits.

5 -248 3

This identifier may not be set.

5 -249 2

A use procedure already exists for this file.

5 -250 2

A use procedure has already been associated with this processing mode.

5 -251 3

Invalid argument in a "CALL".

5 -252 3

Expected word was "EXCEPTION".

5 -253 3

Expected word was "END-CALL".

5 -254 3

Expected word was "OVERFLOW".

5 -255 3

Record size of this file not compatible with record size of the "SD".

5 -256 1

Length over 31 characters .

5 -257 2
Embedded blanks have been skipped.

5 -258 3
File organization should be sequential.

5 -259 3
Forbidden usage of abbreviated relation.

5 -260 3
This feature is not implemented.

5 -261 3
Item is neither "PROCEDURES" nor an identifier.

5 -262 3
The file is described without subordinate 01 entry.

5 -263 3
Expected word was "."

5 -264 2
Sending and receiving fields overlap.

5 -265 4
Implementation restriction: too many nested "IF" statements and compound conditions.

5 -266 4
Implementation restriction: too many nested arithmetic expressions.

5 -267 3
The word "TO" or an integer numeric data item operand was expected.

5 -268 3
The word "TO", an index data item operand or an integer numeric data item operand was expected.

5 -269 3
The item should be either an index data item or an index.

5 -270 3
Item is neither "ON" nor "OFF".

5 -271 3
Item is not a switch-name.

5 -272 3
Item should be "WHEN", "." or "ELSE".

5 -273 3
Illegal assignment.

5 -274 3
"START" statement contradicts file organization and access.

5 -275 3
"ALTER" violates segmentation rules.

5 -276 3
This item should be a key of the searched table.

5 -277 3
This key has already been referenced in this "SEARCH ALL".

5 -278 3
At least one key reference is missing in the "WHEN" phrase of a "SEARCH ALL".

5 -279 2
This identifier does not comply to the rule on usage of key-item or of first index in a "SEARCH ALL" condition.

5 -280 3
H-2000 random files should not be open in output mode when the access is sequential.

5 -281 3
Item is not "OF".

5 -282 2
SSF format is implied for the corresponding file.

5 -283 1
Comparison between numeric and non-numeric items.

5 -284 1
Moving non-numeric to numeric.

5 -285 2
Neither "STOP RUN" nor "EXIT PROGRAM" was met.

5 -286 2
Previous calls to the same program had a different number of arguments.

5 -287 3
Abnormal arguments in a "CALL" to "H_CBL_UGETG4"
(2 mandatory usage COMP-1 arguments).

5 -288 2
Variable length should fit at object time.

5 -289 2
LEVEL-62 feature, the item is ignored.

5 -290 3
LEVEL-62 feature, not implemented.

5 -291 2
The result is unpredictable when a file that is not external is passed as argument.

5 -292 4
The use of "TERMINAL" is not available on your site, please contact supplier.

5 -293 3
Expected word was "CONSOLE".

5 -294 3
A float binary item should not appear in this context.

5 -295 3
Expected item was either a non sort file or "FILE".

5 -296 6

This feature (absence of the "DUPLICATES" phrase) is a NON STANDARD DSA feature, not included in the current compilation level.

5 -297 3

The reserved word "CONVERSION" is expected here.

5 -298 3

Illegal use of a Boolean item.

5 -299 3

Illegal use of a Boolean expression in a condition.

5 -300 3

Item is not a Boolean item.

5 -301 1

"END-ADD" implied here.

5 -302 1

"END-CALL" implied here.

5 -303 1

"END-DIVIDE" implied here.

5 -304 1

"END-MULTIPLY" implied here.

5 -305 1

"END-SUBTRACT" implied here.

5 -306 3

A verb terminated on an explicit end of scope should not contain any reference to "NEXT SENTENCE".

5 -307 1

"END-COMPUTE" implied here.

5 -308 1

"END-IF" implied here.

5 -309 3
Illegal use of reference modification.

5 -310 3
The literal contains several times the same character.

5 -311 1
"END-DELETE" implied here.

5 -312 3
Expected word was "INVALID".

5 -313 1
"END-READ" implied here.

5 -314 1
"END-RECEIVE" implied here.

5 -315 1
"END-RETURN" implied here.

5 -316 1
"END-REWRITE" implied here.

5 -317 1
"END-START" implied here.

5 -318 1
"END-STRING" implied here.

5 -319 1
"END-UNSTRING" implied here.

5 -320 1
"END-WRITE" implied here.

5 -321 3
The file does not support complementary records.

5 -322 3
Abnormal argument in a "CALL" to "H_CBL_USETST"
(1 mandatory integer not over 32767).

5 -323 2
LEVEL-62 feature (paragraph definition missing).

5 -324 2
LEVEL-62 feature (Procedure Division should begin with a section).

5 -325 3
The data-name must be byte aligned for "LENGTH OF" purpose.

5 -326 1
Value of subscript is out of range (accepted when "NSUBCK").

5 -327 3
"NEXT" not accepted for a file whose organization is sequential.

5 -328 3
Expected item was "TO".

5 -329 3
The "SAME SORT (or SORT-MERGE) AREA" clause of this file is incompatible with that of a previous file in this statement.

5 -330 3
Expected item was "TEST".

5 -331 3
Expected item was "BEFORE" or "AFTER".

5 -332 2
Sending item contains non-alphabetical characters.

5 -333 2
Literal is not within normal range of compared item.

5 -334 2
Literal contains characters that normally do not exist in compared item.

5 -335 1
"END-SEARCH" implied here.

5 -336 3

The "SAME AREA" clause of this file is incompatible with that of a previous file in the statement.

5 -337 1

This file already exists in the "USING" series.

5 -338 2

LEVEL-62 feature.

5 -339 4

Implementation restriction (variable length).

5 -340 6

A "COBOL C" rule is violated here.

5 -341 6

A "COBOL C" program cannot be both segmented and a called program.

5 -342 3

"ALL" is not allowed with floating item.

5 -343 3

Only one "USE BEFORE REPORTING" is allowed for a given report group.

5 -344 1

Integer out of range (accepted when "NSUBCK").

5 -345 3

Access to the file should be sequential.

5 -346 3

Incompatible with "ADVANCING PAGE".

5 -347 5

New syntax for COBOL-85.

5 -348 3

Only input CD or input-output CD allowed with the "TERMINAL" phrase.

5 -349 3

Only output CD or input-output CD allowed here.

5 -350 3

Only input CD or input-output CD allowed here.

5 -351 3

Item is not index-name, index data item, elementary item described as an integer, "SORT", "MERGE", "SORT-MERGE", "CODE-SET" or condition-name.

5 -352 3

A condition-name or "TO" was expected here.

5 -353 3

The keyword "TRUE" is mandatory.

5 -354 3

Arithmetic expression not accepted in this context.

5 -355 3

"EXIT PROGRAM" statement cannot be followed by another verb.

5 -356 3

Arguments must be byte aligned.

5 -357 3

Arguments must not be reference modified.

5 -358 3

The number of arguments is different from the number of parameters of the called program.

5 -359 2

Size of argument not equal to size of parameter.

5 -360 3

A "SORT" or "MERGE" procedure must be such that the beginning section is before the ending section in the source program.

5 -361 3

A "SORT" or "MERGE" input (output) procedure cannot be part of any output (input) procedure.

5 -362 3

The associated input procedure contains no "RELEASE" statement.

5 -363 3

The associated output procedure contains no "RETURN" statement.

5 -364 5

The first key (it must be an ascending key) should have the same relative address in the sort file record as the prime record key in the records of this file.

5 -365 3

Item is not identifier, literal or expression.

5 -366 3

Expected item was "END-ADD".

5 -367 3

Expected item was "END-CALL".

5 -368 3

Expected item was "END-COMPUTE".

5 -369 3

Expected item was "END-DELETE".

5 -370 3

Expected item was "END-DIVIDE".

5 -371 3

Expected item was "END-MULTIPLY".

5 -372 3

Expected item was "END-READ".

5 -373 3

Expected item was "END-RECEIVE".

5 -374 3
Expected item was "END-RETURN".

5 -375 3
Expected item was "END-REWRITE".

5 -376 3
Expected item was "END-START".

5 -377 3
Expected item was "END-STRING".

5 -378 3
Expected item was "END-SUBTRACT".

5 -379 3
Expected item was "END-UNSTRING".

5 -380 3
Expected item was "END-WRITE".

5 -381 3
Implementation restriction: this item must be a reference to a section.

5 -382 3
Expected item was identifier, paragraph-name, section-name or "PROGRAM".

5 -383 3
Expected item was alphanumeric literal or non-Boolean and non-numeric identifier.

5 -384 3
Invalid "BY CONTENT" argument to externally compiled program.

5 -385 6
Some phrase or option is missing, though mandatory when compiling at "DSA" level.

5 -386 5
A "SORT" statement may not exist in a declarative.

5 -387 5

An "INPUT PROCEDURE" or an "OUTPUT PROCEDURE" may not be part of the declaratives portion of the Procedure Division.

5 -388 6

This item is too long for "DSA" compiler (5 characters).

5 -389 6

This feature (over 32767 cycles) is a NON STANDARD DSA feature, not included in the current compilation level.

5 -390 3

Item should be a selection subject.

5 -391 3

"ZERO" not accepted in this context.

5 -392 3

Item is not a valid selection object.

5 -393 3

Previous selection objects should be just as many as the selection subjects above.

5 -394 4

Implementation restriction. No room enough to process this statement.

5 -395 2

This "WHEN" clause will never be exercised either because condition(s) will never be true or because of previous "WHEN" clause(s).

5 -396 3

Imperative verb or "END-PERFORM" expected here.

5 -397 2

This rule is partially redundant or inconsistent with the previous rule(s).

5 -398 1

"END-EVALUATE" implied here.

5 -399 2

In this relation, the literal is not in normal range of the compared data or is unnecessarily precise.

5 -400 4

Compiler error. "EVALUATE" statement cannot be processed.

5 -401 3

This feature is only accepted at main program level.

5 -402 2

The hierarchy of the logical operators "AND" and "OR" also applies across abbreviated conditions. Check the use of the previous "AND".

5 -403 3

Expected item was "CURRENCY", "LINES-PER-PAGE", "MINIMUM-DB-KEY" or "NUMBER-OF-PAGES".

5 -404 5

Item should be a usage DB-KEY data reference.

5 -405 3

Expected item was "WITHIN" or a record-name reference.

5 -406 3

Item should be an elementary integer data reference.
(usage COMP-1 suggested).

5 -407 3

Item should be an elementary integer data reference.
(usage COMP-2 suggested).

5 -408 3

Item should be a record-name reference.

5 -409 3

Item should be an elementary alphanumeric data reference.
("PIC X(30)" suggested).

5 -410 3

Item should be "REALM-NAME".

5 -411 3
Item should be a set-name reference.

5 -412 3
Item should be either "SET" or a set-name reference.

5 -413 3
Expected item was either "DB-KEY" or "WITHIN".

5 -414 3
Expected item was "USING".

5 -415 3
Expected item was "WITHIN".

5 -416 3
Item should be either a set-name or a realm-name reference.

5 -417 3
Only one occurrence of each option is allowed.

5 -418 3
Item should be either "OWNER", "MEMBER" or "TENANT".

5 -419 3
Item should be "EMPTY".

5 -420 3
Item should be either "ALL" or a set-name reference.

5 -421 3
Item should be either "USAGE-MODE" or a realm-name reference.

5 -422 3
Item should be either "RETRIEVAL" or "UPDATE".

5 -423 3
Item should be "RETRIEVAL".

5 -424 1
Too big "EVALUATE" statement. Consistency and redundancy checking may not be fulfilled.

5 -425 1
Usage DB-KEY item handled as usage COMP-2.

5 -426 3
Item is not a "MANUAL" member of the set-name.

5 -427 3
Item should have a "CALC" location mode.

5 -428 3
Duplicates are not allowed in "LOCATION MODE CALC" declaration.

5 -429 3
Only one "USE FOR DB-EXCEPTION" accepted in declaratives.

5 -430 3
This item is not a field-name declared within a record defined as a member of the set.

5 -431 3
This item does not belong to the specified record.

5 -432 3
The record was not defined as member of the realm.

5 -433 3
The record was not defined as a tenant of the set.

5 -434 3
No schema-name can be attached to this statement.

5 -435 3
Expected item was "TERMINAL".

5 -436 3
Item is not an input-output CD-name.

5 -437 3
The set has no manual members.

5 -438 3
All items should belong to the same record.

5 -439 3

This item was previously found in the series, this will fail at run time.

5 -440 1

The description of this item has led to inefficient code in arithmetic context.

5 -441 6

This feature (^2) is a ^1 feature, not included in the current compilation level.

5 -442 3

When segmentation is used, the entire Procedure Division must be in sections.

5 -443 2

Binary floating data items are invoked in a relation which tests for equality.

5 -444 2

At run time, actual program-name should not be larger than 31 characters.

5 -445 2

Reference modification should not be applied to a non-display group item.

5 -446 3

Item should be "EQUAL".

5 -447 3

Item should be either "FILE" or "MEMBER".

5 -448 3

Item should be either an alphanumeric identifier, a non-numeric literal or "ACTUAL".

5 -449 3

Item should be either an alphanumeric identifier, a non-numeric literal or a non sort file-name.

5 -450 3

This file should be of "QUEUED" organization.

5 -451 3

Implementation restriction. An "ASSIGN" statement is accepted only when the corresponding "SELECT" clause contains a file literal.

5 -452 3

"B-EXOR" is ambiguous in this context. Use parenthesis.

5 -453 5

This item has to be subscripted although it has one occurrence only.

5 -454 7

This clause is temporarily kept in the COBOL Standard. It will disappear from a future version of the Standard.

5 -455 2

Period missing before this word.

5 -456 2

The field is not large enough to hold the longest record.

5 -457 1

This "USE FOR DEBUGGING" statement identifies a "GLOBAL" item.

5 -458 2

"USE FOR DEBUGGING" is restricted to data names dealing with at most 3 "OCCURS" levels.

5 -459 3

A file with the "LINAGE" clause may not be opened in the "EXTEND" mode.

5 -460 3

Expected item was "ALSO".

5 -461 3

"WITH CONVERSION" phrase is ignored when such a device is used.

5 -462 1

The structure of the "INPUT PROCEDURE" does not permit to check that it contains at least one "RELEASE" statement.

5 -463 1

The structure of the "OUTPUT PROCEDURE" does not permit to check that it contains at least one "RETURN" statement.

5 -464 3

Unexpected left parenthesis.

5 -465 3

AFTER phrase is not allowed in an in-line perform.

5 -466 4

Abnormal end of conditional phrase.

5 -467 2

Comparison between index-name and alphanumeric item. The result will be unpredictable.

5 -468 3

All the elements of this verb must belong to the same subschema.

5 -469 3

Expected word was subschema-name.

5 -470 3

The record was not defined as a tenant of this key.

5 -471 3

No explicit subschema.

5 -472 3

Expected word was ADVANCING.

5 -473 4

Too many errors; compiler stops processing.

5 -474 3

No possibilities for this Record-name in this verb.

5 -475 2

Out of bounds leftmost character position.

5 -476 2

Too large length value (possibly due to too large leftmost character position).

5 -477 3

Key-name expected here.

5 -478 3

Item should be of usage POINTER.

5 -479 3

Item is not a valid based data.

5 -480 3

Illegal pointer comparison.

5 -481 3

Only test for equality is allowed.

5 -482 3

Not allowed in this context.

5 -483 3

Not allowed in COBOL 74 level.

5 -484 3

Expected word was VALUE.

5 -485 3

Item may not be of usage pointer.

5 -486 3

Item may not be an Index data item.

5 -487 1

The size of this item is not equal to the record size.

5 -488 3

Ambiguous or erroneous condition. Please use parenthesis.

5 -489 3

Not enough subscripts.

5 -490 3
Unexpected Reserved Word.

5 -491 3
This Condition-name has no FALSE value.

5 -492 3
This set may not be modified.

5 -493 3
The record was not defined as a tenant of this key.

5 -494 3
This item is a field-name declared within this record defined as the owner of the set.

5 -495 3
The record was not defined as member of this set.

5 -496 2
Impossible retaining clause.

5 -497 3
This sets have no common members.

5 -498 3
Two operands connected by a THROUGH phrase must be of the same class.

5 -499 3
The previous operand should be neither alphabetic nor numeric.

5 -500 3
A file-literal may not contain an IFN.

5 -501 3
ERROR-501. RFU.

5 -502 3
No valid key for this SORT.

5 -503 3
Invalid Sort Key.

5 -504 3
Invalid data-name for a SORT statement.

5 -505 2
This feature is not allowed because its implementation is not completed.

5 -506 3
END DECLARATIVES is missing.

5 -507 3
Illegal use of Intrinsic Function.

5 -508 3
Wrong number of argument.

5 -509 3
Wrong type of argument.

5 -510 3
Unknown name of Intrinsic Function.

5 -511 3
Erroneous parameter.

5 -512 3
This Function should be alphanumeric.

5 -513 3
This Function should be numeric.

5 -514 3
Invalid GIVING option in a CALL statement.

5 -515 3
Invalid GIVING option in an EXIT statement.

5 -516 3
Invalid operand in a GIVING phrase for a CALL statement.

5 -517 3

Invalid operand in a GIVING phrase for an EXIT statement.

5 -518 3

ERROR-518. RFU

5 -519 2

ERROR-519. RFU

5 -520 3

ERROR-520. RFU

5 -521 3

ERROR-521. RFU

5 -522 3

"PREVIOUS" not accepted for a file whose organization is sequential or relative.

5 -523 3

"LESS" or "NOT GREATER" not accepted for a file whose organization is relative.

5 -524 3

Expected word was "REFERENCE".

5 -525 3

Abnormal argument in a "CALL" to "H_CBL_UGETPN" (1 mandatory 30 - character alphanumeric).

6 -1 3
Ambiguous reference.

6 -2 3
Item not declared.

6 -3 3
Ambiguous qualified item.

6 -4 3
Paragraph-name not found in the current section.

6 -5 3
Paragraph-name multiply declared within its containing section.

6 -6 3
Qualified name multiply declared within its containing group item.

6 -7 3
Bad component in subscript.

6 -8 3
Numeric literal, data-name or index-name expected here.

6 -9 3
Numeric literal expected here.

6 -10 3
Incomplete qualification. Data-name expected here.

6 -11 3
Too many qualifiers in this reference.

6 -12 3
When the "THRU" option is present, the first referenced item must begin on a byte boundary.

6 -13 3
When the "THRU" option is present, the second referenced item must end on a byte boundary.

6 -14 3
A control item must not have variable length.

6 -15 3
Identifier must have an "OCCURS" clause in its description.

6 -16 3
Identifier must have an "INDEXED BY" clause in its description.

6 -17 3
Identifier must have a "KEY IS" clause in its description.

6 -18 3
Identifier expected here.

6 -19 4
Compiler error. Name table buffer contains no new entry for two consecutive loads.

6 -20 3
This item must be an unsigned integer.

6 -21 2
A relative key cannot belong to a record of this file.

6 -22 3
This item must be alphanumeric and not variable length.

6 -23 3
This item must belong to a record of this file.

6 -24 3
Status can only be 2 alphanumeric characters and not in File, Constant or Linkage Sections.

6 -25 5
This item must be an integer.

6 -26 3
Rename object cannot have an "OCCURS" clause in its data description nor can it be subordinate to one.

6 -27 3

A 66 level entry cannot rename another 66 level entry nor can it rename a 88 or a 01 level entry nor an index data item.

6 -28 3

Rename object1 and object2 area range conflict.

6 -29 3

Key field is too small.

6 -30 3

Key location value too large.

6 -31 3

Key location is outside of the record area.

6 -32 3

Class is not alphanumeric.

6 -33 2

Item is not elementary.

6 -34 3

Field is too short for a relative key.

6 -35 2

Field is too long for a relative key.

6 -36 3

The data-name referenced in the "DEPENDING ON" clause of "OCCURS" must not be specified in the range of this "OCCURS" clause.

6 -37 3

Illegal reference.

6 -38 3

Alternate key cannot have the same offset as that of the record key or any other alternate key.

6 -39 3

The size of this key must not be greater than 255 characters.

6 -40 3
Control item must be a data-name.

6 -41 6
This feature is a ^1 feature, not included in the current compilation level.

6 -42 3
This item cannot have an "OCCURS" clause nor be subordinate to a group which contains an "OCCURS" clause.

6 -43 3
Report item can only be used for sum counter reference qualification.

6 -44 6
This feature ^ feature not included in the current compilation level.

6 -45 3
Secondary key exceeds 255 characters.

6 -46 3
Secondary key cannot have the same offset as that of record key or any other secondary key.

6 -47 3
Invalid control item.

6 -48 3
"USE FOR DEBUGGING" must not reference a use for debugging procedure-name.

6 -49 3
Named more than once in "USE FOR DEBUGGING".

6 -50 3
Illegal key reference.

6 -51 3
Only data-name is allowed as a parameter.

6 -52 3
Too long literal.

6 -53 3
Renames object cannot have usage BIT attribute.

6 -54 3
Item must be an alphabet-name.

6 -55 3
Secondary key in Working-Storage Section or in Linkage Section must have a two characters internal-file-name.

6 -56 3
This item must be an external data item described in the current Data Division.

6 -57 6
This feature ("^1" is not unique) is a NBS HIGH INTERMEDIATE feature, not included in the current compilation level.

6 -58 3
Key-name duplicate.

6 -59 3
Key cannot be Boolean.

6 -60 2
This data item should be unsigned.

6 -61 1
Exceeds 5 levels of qualification.

6 -62 4
Implementation restriction. No room enough to ^.

6 -63 3
Only data-name is allowed here.

6 -64 3
Illegal reference of a Linkage Section item.

6 -65 3
User defined "TALLY" conflicts with the special register implied by the "EXAMINE" statement.

6 -66 3
This item must be a global data item described in the current Data Division.

6 -67 5
This item must be an unsigned integer.

6 -68 4
Compiler error while processing a "RENAMES" clause.

6 -69 5
Procedure-name not allowed when "ALL PROCEDURES" option is also used.

6 -70 2
The value of this item should be non-negative.

6 -71 3
This item must be an unsigned integer whose description does not contain the PICTURE symbol "P".

6 -72 1
This name is assumed to be a system name.

6 -73 1
This alphabet name is assumed declared as "EBCDIC".

6 -74 3
This item must be an external data item.

6 -75 3
Key cannot be a pointer data item.

6 -76 6
This feature (^2) is a ^1 feature, not included in the current compilation level.

6 -77 2
This item must be defined in File, Working-Storage or Linkage section.

6 -78 3
This item must be defined in Working-Storage or Linkage section.

6 -79 3
The PADDING data item must be 1-character alphanumeric.

6 -80 3
The PADDING data item must not be defined in the Communication Section, the File Section or the Report Section.

6 -81 1
^.

6 -82 2
^.

6 -83 3
^.

6 -84 4
^.

6 -85 5
^.

6 -86 6
^.

6 -87 7
^.

6 -90 3
This item must be a symbolic character.

6 -91 3
Only a data name or a symbolic character is allowed here.

6 -92 3
The description of this item must not contain the PICTURE symbol "P".

7 -1 2
"CORRESPONDING" option results a null match. Items which are 66, 88 or which contain or subordinate "REDEFINES", "OCCURS" or "USAGE IS INDEX" are not considered.

7 -2 3
Item is not identifier.

7 -3 3
Item is not identifier or literal.

7 -4 3
Expected word was "BY".

7 -5 3
Expected word was "DATA" or "BY".

7 -6 3
Expected word was "ERROR".

7 -7 1
When executing in debugging mode, the subscript value in debug item will be that after the statement is executed.

7 -8 6
This feature is a ^ feature, not included in the current compilation level.

7 -9 3
Identifier may not be defined with level 66 or with "USAGE IS INDEX" clause.

7 -10 3
Identifier may not be an index data item.

7 -11 4
Compiler error. Unexpected token in "CORRESPONDING" operand.

7 -12 3
Name multiply declared within a structure.

7 -13 3
Hierarchy error in "CORRESPONDING" operand.

7 -14 3
Category of identifier different from the one specified.

7 -15 4
Compiler error. Premature end of file during "CORRESPONDING" option or "INITIALIZE" statement.

7 -16 3
Identifier must be group item.

7 -17 3
Expected word was "FROM".

7 -18 3
Expected word was "(".

7 -19 3
Expected word was "SIZE".

7 -20 3
Expected word was "TO".

7 -21 3
Identifier in "CORRESPONDING" option cannot be reference modified.

7 -22 3
Operand of "INITIALIZE" may not have "OCCURS DEPENDING ON".

7 -23 3
Illegal operand in the "REPLACING" clause of "INITIALIZE".

7 -24 6
This feature (level 66 item) is a NON STANDARD feature, not included in the current compilation level.

7 -25 2
"INITIALIZE" statement results in no match.

7 -26 3

"INITIALIZE" sending operand not legal category.

7 -27 4

Implementation restriction. No room enough to accommodate "CORRESPONDING" or "INITIALIZE" processing.

7 -28 4

Implementation restriction. Too many operands for this statement.

7 -29 4

Implementation restriction. This structure has too many data descriptions subordinate to it.

7 -30 7

This clause is temporarily kept in the COBOL Standard. It will disappear from a future version of the Standard.

/* PRINT phase errors */

8 -1 2

Implementation restriction. No room enough to hold diagnostic messages, part of them will not be printed embedded in the source, but before the listing and/or will not be sorted in line number order.

8 -2 2

Implementation restriction. No room enough to hold line number of erroneous lines in summary listing. Summary report will be incomplete, refer to SOURCE and/or ERROR LISTING.

8 -3 2

Implementation restriction. No room enough to hold diagnostics processed interactively. Identical diagnostics will be entirely printed.

/* CROSS REFERENCE phase errors */

8 -11 2

Compiler error. Internal line number ^ has no corresponding external line number, 0 is assumed.

8 -12 2

Implementation restriction. No room enough to hold CROSS REFERENCE; some references may be missing.

8 -13 4

Implementation restriction. No room enough to hold CROSS REFERENCE.

8 -14 2

Implementation restriction. No room enough to hold computed table size, only part of table will be used.

/* MAP phase errors */

8 -20 2

Implementation restriction. No room enough to hold PERFORM/ALTER BUCKETS.

/* GENERATOR phase errors */

8 -21 4

Implementation restriction. Feature not yet implemented.

8 -22 4
Compiler error.

8 -23 4
Implementation restriction. No room enough to hold "ALTER" table.

8 -24 4
Implementation restriction. No room enough to hold "PERFORM" table.

8 -25 4
Implementation restriction. No room enough to hold intermediate data item.

8 -26 4
H_CBL_USPACE error.

8 -27 4
Implementation restriction. No room enough to hold a complete verb.

8 -28 4
Implementation restriction. More than 32767 tags.

8 -38 4
Compiler error. Base register allocation error.

8 -39 4
Implementation restriction. No more base registers available, please contact supplier.

8 -42 4
Implementation restriction. No more general registers available, please contact supplier.

8 -43 2
Compiler error. General register ^1 allocation error.

8 -44 2
Compiler error. General register ^1 lock error.

8 -45 4
Compiler error. Invalid internal segment number.
/* GARBAGE COLLECTOR (ESCAPE) errors */

8 -51 4

Compiler error. The address ^1 is unknown to the garbage collector.

8 -52 4

Compiler error. Incompatible new asked area size ^1 for area ^2 whose reserved maximum size is ^3.

8 -53 4

Unrecoverable difficulty due to system error while expanding segment ^1 to ^2 bytes (^3).

8 -54 4

Unrecoverable difficulty due to system error while asking for maximum size of segment ^1 (^2).

8 -55 2

Impossible to create a segment with a maximum size of ^1 (^2).

8 -56 2

Segment ^1 could not be deleted (^2).

/* DEBUG phase errors */

8 -61 4

Implementation restriction. No room enough to hold ^1 procedure-names to build PCF tables.

8 -62 4

Compiler error. Unknown data-name defined on line ^1.

8 -63 4

Compiler error. Unknown compiler generated data-name: ^1.

8 -64 4

Implementation restriction. No room enough to hold ^1 contained programs to build PCF tables.

8 -65 4

Implementation restriction. Only ^1 areas were reserved to hold data-names to build PCF tables.

8 -66 4

Implementation restriction. No room enough to hold ^1 areas to hold data-names to build PCF tables.

8 -67 2

Implementation restriction. Too many items in the statement, part of them will not be included in the PCF tables.

8 -68 4

COBOL-85 features not implemented on this system release.

8 -69 4

Implementation restriction. Signed COMP-5 items not implemented in PCF.

/* USE FOR DEBUGGING phase errors */

8 -81 4

Implementation restriction. No room enough to hold "USE FOR DEBUGGING" tables.

8 -82 4

Implementation restriction. Too many data-names or procedure-names to build "USE FOR DEBUGGING" tables.

/* FIXUP phase errors */

8 -91 4

^ is not a CU library.

8 -92 4

CULIB is full.

8 -93 4

I/O error on CULIB.

8 -94 4

Segment number limit of 1024 ISNs has been exceeded. Increase segment size.

8 -95 4

Implementation restriction. No room enough to hold ^ tags.

8 -96 4

Implementation restriction. No room enough to hold sort table. Increase data segment size.

8 -97 4

Compiler error. Invalid tag number ^1 defined at address ^2.

8 -98 4

Compiler error. Invalid tag number ^1 equated to tag number ^2.

8 -99 4

Compiler error. Invalid tag number ^1 for symref.

8 -100 4

Compiler error. Tag number ^1 at address ^2 not defined.

8 -101 4

Impossible to process CU. Temporary name ^1 already exists and is an alias of another CU.

8 -102 4

Impossible to process CU. Cu name ^1 already exists and is an alias of another CU.

8 -103 2

The execution of paragraph "^" would lead to an infinite loop; it will abort instead.

8 -104 4

Impossible to open H_CULIB.

8 -105 4

Implementation restriction. No room enough to hold alias table. Delete old CU.

8 -106 4

Compiler error. Duplicate definition for tag number ^1 defined at address ^2.

8 -107 4

Compiler error. Invalid equivalence of tag number ^1 and tag number ^2.

8 -108 4

Compiler error. No ISN was computed for EXTERNAL item "^1".

8 -109 4

Compiler error. EXTERNAL item with internal number ^1 not found.

8 -110 4

Segment number limit of 256 has been exceeded for code segments. Gather sections.

8 -111 4

Impossible to create CU. Linkage Section exceeds 32K bytes (currently ^1 bytes).

8 -112 4

Library assigned to H_CULIB is not known.

/* DATA DICTIONARY phase errors */

8 -121 4

Compiler restriction. No room enough to sort the references for theDATA DICTIONARY.

8 -122 4

Compiler restriction. No room enough to hold the name table for the DATA DICTIONARY.

8 -123 4

Compiler error. Attempt to open a protected string before the current one is closed.

8 -124 4

Compiler error. Attempt to close a protected string while none is opened.

8 -125 4

Compiler error. Name table contains no new entry for two consecutive loads.

8 -126 3

Compiler error. Invalid internal value (^1) for ^2.

8 -127 4

Impossible to open DICLIB.

8 -128 4
^1 is not an SL library.

8 -129 4
Impossible to open temporary subfile in DICLIB.

8 -130 4
Impossible to close temporary subfile created in DICLIB.

8 -131 4
Impossible to close ^1.

8 -132 4
I/O error on DICLIB.

8 -133 4
DICLIB is full.

8 -134 4
Impossible to open the command file.

8 -135 4
Impossible to get next command from the command file.

8 -136 4
Impossible to close the command file.

8 -137 4
Library assigned to DICLIB is not known.

8 -138 2
DATA DICTIONARY direct input does not support sub schemas.

8 -139 2
W A R N I N G ! . More than 256 ISNs (^1). This CU may be incompatible with some GCOS-7 processors.

8 -140 4
Limit of 256 CU Segments has been exceeded. (^1)

9 -1 4
Unrecoverable difficulty.

9 -2 4
Unrecoverable difficulty.

9 -3 4
Unrecoverable difficulty.

9 -4 4
Unrecoverable difficulty.

9 -5 4
Unrecoverable difficulty.

9 -6 4
Unrecoverable difficulty.

9 -7 4
Unrecoverable difficulty.

9 -8 4
Unrecoverable difficulty.

9 -9 4
Unrecoverable difficulty.

9 -10 4
Unrecoverable difficulty.

9 -11 4
Unrecoverable difficulty.

9 -12 4
Unrecoverable difficulty.

9 -13 4
Unrecoverable difficulty.

9 -14 4
Unrecoverable difficulty.

9 -15 4
Unrecoverable difficulty.

9 -16 4
Unrecoverable difficulty.

9 -17 4
Unrecoverable difficulty.

9 -18 4
Unrecoverable difficulty.

9 -19 4
Unrecoverable difficulty.

9 -20 4
Unrecoverable difficulty.

9 -21 4
Unrecoverable difficulty.

9 -22 4
Unrecoverable difficulty.

9 -23 4
Unrecoverable difficulty.

9 -24 4
Unrecoverable difficulty.

9 -25 4
Unrecoverable difficulty.

9 -26 4
Unrecoverable difficulty.

9 -27 4
Unrecoverable difficulty.

9 -28 4
Unrecoverable difficulty.

9 -29 4
Unrecoverable difficulty.

9 -30 4
Unrecoverable difficulty.

9 -31 4
Unrecoverable difficulty.

9 -32 4
Unrecoverable difficulty.

9 -33 4
Unrecoverable difficulty.

9 -34 4
Unrecoverable difficulty.

9 -35 4
Unrecoverable difficulty.

9 -36 4
Unrecoverable difficulty.

9 -37 4
Unrecoverable difficulty.

9 -38 4
Unrecoverable difficulty.

9 -39 4
Unrecoverable difficulty.

9 -40 4
Unrecoverable difficulty.

9 -41 4
Unrecoverable difficulty.

9 -42 4
Unrecoverable difficulty.

9 -43 4
Unrecoverable difficulty.

9 -44 4
Unrecoverable difficulty.

9 -45 4
Unrecoverable difficulty.

9 -46 4
Unrecoverable difficulty.

9 -47 4
Unrecoverable difficulty.

9 -49 4
Compiler error during ^1 on sequential workfile "^2". File is opened input.

9 -50 4
Compiler error during ^1 on sequential workfile "^2". File is opened output.

9 -51 4
Compiler error during ^1 on sequential workfile "^2". File is closed.

9 -52 4
Compiler error during ^1 on sequential workfile "^2". File is exhausted.

9 -53 4
Compiler error during ^1 on sequential workfile. Invalid file pointer.

9 -54 4
Compiler error during ^1 on sequential workfile "^2". Invalid length (^3).

9 -55 4
^ is full.

9 -56 4
Backing store is full. Use WORK files for large programs.

9 -57 4
I/O error on sequential workfile.

9 -58 4
Compiler error during ^1 on direct workfile. File is opened.

9 -59 4
Compiler error during ^1 on direct workfile. File is closed.

9 -60 4
Compiler error during ^1 on direct workfile. File is exhausted.

9 -61 4
Compiler error during ^1 on direct workfile. Invalid length (^2).

9 -62 4
Compiler error on common file. Invalid key number (^1).

9 -63 4
Implementation restriction. Too many names in an 01.

9 -64 4
Compiler error on direct workfile. Unable to perform I/O before first block.

9 -65 4
Common file overflow.

9 -66 4
Unrecoverable difficulty due to system error.

9 -67 4
Compiler error on direct workfile. Block number ^ already blocked.

9 -68 4
Compiler error during ^1 on direct workfile. File is not blocked.

9 -69 4
Backing store is full. Too many jobs running concurrently.

9 -70 4
I/O error on direct workfile.

9 -71 4
Compiler error during ^1 on direct workfile. Invalid file pointer.

9 -72 4
File or library assigned to ^1 is not known.

9 -73 4
^1 not defined.

9 -74 4
Impossible to open ^1.

9 -75 4
Impossible to close ^1.

9 -76 4
Invalid ^1 file.

9 -77 3
Impossible to delete subfile on ^.

9 -78 4
Too short block size for ^1. Block size should be at least ^2 bytes.

9 -79 4
Compiler error during open on a sequential workfile "^1". Wrong pmd "^2".

9 -80 4
"^1" is a NON STANDARD feature, not included in the current compilation level.

9 -81 4
Compilation level not allowed or missing.

9 -82 2
Illegal "DSEGMAX" option is ignored (^1).

9 -83 2
Illegal "PSEGMAX" option is ignored (^1).

9 -84 2
Specified "DSEGMAX" option exceeds 4M bytes; 4M bytes assumed.

9 -85 2
Specified "PSEGMAX" option exceeds 32K bytes; 32K bytes assumed.

9 -86 2
Illegal "LEVEL" option is ignored (^1).

9 -87 4
Unrecoverable difficulty due to system error.

9 -88 4
Unrecoverable difficulty due to system error.

9 -89 2
Illegal "ISEGMAX" option is ignored (^1).

9 -90 2
Illegal "TEMP" option is ignored (^1).

9 -91 2
Specified "TEMP" option must range between 13 and 30 inclusively; default value assumed.

9 -92 2
Specified "ISEGMAX" option exceeds 4M bytes; 4M bytes assumed.

9 -93 2
Specified "PSEGMAX" option exceeds 64K bytes; 64K bytes assumed.

9 -94 2
Illegal "CODE" option is ignored (^1).
/* JOR messages */
/* CBL01 messages */

9 -99
ERROR

9 -100

ERROR WHILE COMPILING ^1

9 -101

ERROR WHILE COMPILING LINE ^2 OF ^1

/* CBL02 message */

9 -102

SUMMARY FOR ^1: ^2.

/* CBL03 message */

9 -103

ILLEGAL USE OF COMPILER.

/* CBL04 message */

9 -104

NOT ENOUGH MEMORY TO RUN COMPILER.

/* CBL05 message */

9 -105

^1K EXTRA MEMORY USED FOR ^2 CPU MINUTES, ^3 ELAPSED MINUTES.

/* CBL06 message */

9 -106

^1

/* CBL07 message */

9 -107

SPECIFIED WORKING SET TOO SMALL, ^1K USED.

/* CBL08 message */

9 -108

LISTING CREATED WITH THE NAME ^1 (PROGRAM-NAME LARGER THAN 29 CHARACTERS).

/* CBL09 message */

9 -109

INTERACTIVE COMPILATION IS NOT AVAILABLE ON YOUR SITE, PLEASE CONTACT SUPPLIER.

/* CBL10 message */

9 -110

THIS COBOL COMPILER IS NOT AVAILABLE ON YOUR SITE, PLEASE CONTACT SUPPLIER.

/* CBL11 message */

9 -111

THE REPORT WRITER IS NOT AVAILABLE ON YOUR SITE, PLEASE CONTACT SUPPLIER.

/* CBL12 message */

9 -112

THE DATA MANIPULATION LANGUAGE (DML) IS NOT AVAILABLE ON YOUR SITE, PLEASE CONTACT SUPPLIER.

/* CBL13 message */

9 -113

THE DATA DICTIONARY DIRECT INPUT IS NOT AVAILABLE ON YOUR SITE, PLEASE CONTACT SUPPLIER.

/* CBL14 message */

9 -114

THE TEST COVERAGE FACILITY IS NOT AVAILABLE ON YOUR SITE, PLEASE CONTACT SUPPLIER.

9 -115

THE SUBSCHEMA PROCESSOR IS NOT AVAILABLE ON YOUR SITE, PLEASE CONTACT SUPPLIER.

/* additional information to cbl01 messages */

9 -120

IMPOSSIBLE TO OPEN H_PR

9 -121

INVALID TYPE FOR SYSOUT

9 -122

INVALID TYPE FOR PRTFILE

9 -123

INVALID TYPE FOR PRTLIB

9 -124
LISTING FILE EXHAUSTED

9 -125 4
FILE OR LIBRARY ASSIGNED TO H_PR IS NOT KNOWN.
/* end of JOR messages */
/* H_CBL_EFDCNS procedure errors */

9 -150 4
^1 not defined.

9 -151 4
Impossible to assign ^1.

9 -152 4
Impossible to open ^1.

9 -153 4
Impossible to close ^1.

9 -154 4
I/O error on ^1.

9 -155 4
Impossible to prompt ^1.

9 -156 4
Impossible to force write on ^1.

9 -157 4
File assigned to ^1 is not known.

10-1 4
No input "DDLIB" specified for sub-schema processing.

10-2 4
Impossible to open ^1.

10-3 4
^1 is not a BIN library.

10-4 4
Impossible to open ^1.

10-5 3
^1 not found in assigned "DDLIB" libraries.

10-6 3
Impossible to close ^1.

10-7 3
Impossible to close ^1.

10-8 3
^1 is not a data description object.

10-9 4
Impossible to access ^1.

10-10 4
Implementation restriction. No room enough to hold object directory of ^1.

10-11 4
Impossible to access directory of ^1.

10-12 4
Implementation restriction. No room enough to hold object districts of ^1.

10-13 4
Impossible to access districts of ^1.

10-14 4
Not enough memory to process ^1. Use "SIZEOPT = ^2".

10-15 4
Too many source lines.

10-16 4
Impossible to init fetch of ^1.

10-17 4
Compiler error. Wrong item returned from fetch.

10-18 2
^1 does not contain realms though specified.

10-19 4
Impossible to fetch in ^1.

10-20 3
This realm does not exist in ^1.

10-21 3
This record does not exist in ^1.

10-22 2
^1 does not contain records though specified.

10-23 2
No record was selected in ^1.

10-24 2
Unknown data type, replaced by alphanumeric.

10-25 4
Implementation restriction. No room enough to hold occurs for the following item.

10-26 6
This feature is a ^1 feature, not included in the current compilation level.

10-27 3
The dimension of occurs cannot exceed 3.

10-28 4
Implementation restriction. No room enough to hold records table for ^1.

10-29 4
Library assigned to ^1 is not known.

10-30 4
Not enough memory to process ^1. Use "SIZEOPT".

10-31 4
Not enough memory to hold large record description of ^1.

10-32 4
^1 not found within the specified schema.

Index

A

B

C

D

E

F

G

H

I

J

K

L

M

N

O

P

Q

R

S

T

U

V

W

X

Z

Technical publication remarks form

Title :	DPS7000/XTA NOVASCALE 7000 COBOL85 User's Guide Languages: COBOL

Reference N° :	47 A2 06UL 06	**Date:**	**June 2002**

ERRORS IN PUBLICATION

SUGGESTIONS FOR IMPROVEMENT TO PUBLICATION

Your comments will be promptly investigated by qualified technical personnel and action will be taken as required.
If you require a written reply, please include your complete mailing address below.

NAME : _____ Date : _____

COMPANY : _____

ADDRESS : _____

_____

Please give this technical publication remarks form to your BULL representative or mail to:

Bull - Documentation Dept.
1 Rue de Provence
BP 208
38432 ECHIROLLES CEDEX
FRANCE
info@frec.bull.fr

Technical publications ordering form

To order additional publications, please fill in a copy of this form and send it via mail to:

BULL CEDOC
357 AVENUE PATTON
B.P.20845
49008 ANGERS CEDEX 01
FRANCE

Phone: +33 (0) 2 41 73 72 66
FAX: +33 (0) 2 41 73 70 66
E-Mail: srv.Duplicopy@bull.net

CEDOC Reference #	Designation	Qty
_ _ _ _ _ _ _ _ _ [_ _]		
_ _ _ _ _ _ _ _ _ [_ _]		
_ _ _ _ _ _ _ _ _ [_ _]		
_ _ _ _ _ _ _ _ _ [_ _]		
_ _ _ _ _ _ _ _ _ [_ _]		
_ _ _ _ _ _ _ _ _ [_ _]		
_ _ _ _ _ _ _ _ _ [_ _]		
_ _ _ _ _ _ _ _ _ [_ _]		
_ _ _ _ _ _ _ _ _ [_ _]		
_ _ _ _ _ _ _ _ _ [_ _]		
_ _ _ _ _ _ _ _ _ [_ _]		
_ _ _ _ _ _ _ _ _ [_ _]		

[_ _] : The latest revision will be provided if no revision number is given.

NAME: ______________________________ Date: ____________

COMPANY: ______________________________

ADDRESS: ______________________________

____

PHONE: ______________________ FAX: ______________________

E-MAIL: ______________________________

For Bull Subsidiaries:

Identification: ______________________________

For Bull Affiliated Customers:

Customer Code: ______________________________

For Bull Internal Customers:

Budgetary Section: ______________________________

For Others: Please ask your Bull representative.

BULL CEDOC
357 AVENUE PATTON
B.P.20845
49008 ANGERS CEDEX 01
FRANCE

REFERENCE
47 A2 06UL 06