SERIES 60 (LEVEL 64)
COBOL USER GUIDE

SUBJECT

Description of the Compilation and Execution of COBOL Programs under GCOS Level 64

SPECIAL INSTRUCTIONS

For users of Release 0400 this manual replaces Revision 0 dated July 1977 which remains valid for Release 0300 users. Because of extensive revision, change bars have not been used

SOFTWARE SUPPORTED

Level 64 GCOS Release 0400

ORDER NUMBER

AQ63, Rev. 1

September 1978

Honeywell

PREFACE

This manual describes how the user can compile and execute COBOL programs under Level 64 GCOS.

This manual complements the Level 64 COBOL Reference Manual. The Reference Manual contains formal specifications of COBOL. The User Gui(on the other hand, discusses those aspects of COBOL whose implementati(in Level 64 needs further explanation. It also discusses certain operational aspects such as the JCL and utilities needed to input, compile, link, execute and debug COBOL programs. In discussing these aspects there are some areas of overlap with the Job Control Language Referenc(Manual, Job Control Language User Guide, Library Maintenance Reference Manual and Library Maintenance User Guide. The programmer is cross referred to these manuals for details when necessary.

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

- I/O using the UFAS, BFAS and HFAS file access systems.

- Data base processing using IDS/II.

- Communications and transaction driven programming using TDS/64 and VCAM.

This manual is divided into two parts. Part 1 describes the development of a coded COBOL source program into a working load module. Section I describes methods of introducing source programs into a Level 64 system and maintaining source programs in a disk library. Sections II and III describe the compilation and linkage of COBOL programs. Section IV describes debugging techniques and gives hints on how to deal with abnormal program terminations.

Part 2 of this manual discusses COBOL program interfaces with the system; programming techniques are described which lead to efficient use of the system. Section V discusses the representation of data in memory. Section VI discusses calling and called programs. Section VII discusses segmentation for Level 64 Virtual Memory Management. Section VIII describes various programming techniques for reducing the size and increasing the execution speed of COBOL programs. Section IX discusses various general aspects of file usage. Section X describes the standard record formats accepted by Level 64 Data Management. Section XI describes the use of unit record files. Section XII contains a number of miscellaneous programming topics.

The reader of this User Guide is assumed to be familiar with the Level 64 COBOL language and with the basic functions of the Level 6 GCOS Operating System and JCL.

PREFACE

This manual provides the information about Level 64 COBOL and the Level 64 system needed by a programmer to develop working COBOL programs which will execute efficiently in a Level 64 system.

This manual complements the COBOL Language Reference Manual. The reference manual contains a formal specification of the COBOL programming language. The COBOL User Guide, on the other hand, discusses those aspects of COBOL whose implementation in Level 64 needs further explanation.

The COBOL User Guide also discusses certain operational aspects such as the JCL and utilities needed to input, compile, link, execute and debug COBOL programs. In discussing these aspects there are some areas of overlap with the Job Control Language (JCL) Reference Manual, Job Control Language (JCL) User Guide, Library Maintenance Reference Manual, and Library Maintenance User Guide. The programmer is cross referenced to these manuals for details when necessary.

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

- I/O using the UFAS, BFAS and HFAS file access systems.

- Data base processing using IDS/II.

- Communications and transaction driven programming using TDS/64 and VCAM.

This manual is divided into two parts. Part 1 describes the development of a coded COBOL source program into a working load module. Section I describes methods of introducing source programs into a Level 64 system and maintaining source programs in a disk library. Sections II and III describe the compilation and linkage of COBOL programs. Section IV describes debugging techniques and gives hints on how to deal with abnormal program terminations.

Part 2 of this manual discusses some of the COBOL program's interfaces with the system; programming techniques are described which lead to efficient use of the system. Section V discusses the representation of data in memory. Section VI discusses calling and called programs. Section VII discusses segmentation for Level 64 Virtual Memory Management. Section VIII describes various programming techniques for reducing the size and increasing the execution speed of COBOL programs. Section IX discusses various general aspects of file usage. Section X describes the standard record formats accepted by Level 64 Data Management. Section XI describes the use of unit record files. Section XII contains a number of miscellaneous programming topics.

The reader of this User Guide is assumed to be familiar with the Level 64 COBOL language and with the basic functions of the Level 64 GCOS Operating System and JCL. The following manual may be used as background material:

- System Overview Manual, Order No. AQ98.

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

- COBOL Language Reference Manual, Order No. AQ64
- Job Control Language (JCL) Reference Manual, Order No. AQ10
- Job Control Language (JCL) User Guide, Order No. AQ11
- Library Maintenance Manual, Order No. AQ28
- Library Maintenance User Guide, Order No. AQ87
- Error Messages and Return Codes Manual, Order No. CQ31

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

UPPERCASE	The keyword item is coded exactly as shown.
lowercase	Indicates a user-supplied parameter value.
[item]	An item within square brackets is optional.
{ item1 / item2 / item3 }	A column of items within braces means that one value must be selected if the associated parameter is specified.
	Note the way in which underlining is used in COBOL and JCL syntax descriptions. An underlined word in COBOL syntax descriptions is a word which must be used. An underlined parameter in JCL syntax descriptions is the parameter assumed if none is specified.
( )	Parentheses must be coded if they enclose more than one item.
...	An ellipsis indicates that the preceding item may be repeated one or more times.

Each section of this document is structured according to the heading hierarchy shown below. Each heading indicates the relative level of the text that follows it.

Level	Heading Format
1 (highest)	ALL CAPITAL LETTERS, UNDERLINED
2	Initial Capital Letters, Underlined
3	ALL CAPITAL LETTERS, NOT UNDERLINED
4	Initial Capital Letters, Not Underlined
5 (lowest)	ALL CAPITAL LETTERS FOLLOWED BY COLON: Text begins on the same line.

The Level 64 Document Set follows. Many of the manuals may be referenced in the text.

LEVEL 64 DOCUMENT LIST

Order Number	Title
AQ02	*Series 100 Program Mode Operator Guide*
AQ03	*Series 100 Conversion Guide*
AQ04	*Series 200/2000 Conversion Guide*
AQ05	*System 360/370 Conversion Guide*
AQ09	*System Management Guide*
AQ10	*Job Control Language (JCL) Reference Manual*
AQ11	*Job Control Language (JCL) User Guide*
AQ13	*System Operation Operator Guide*
AQ14	*System Operation Console Messages*
AQ18	*Operator Reference Manual*
AQ20	*Data Management Utilities Manual*
AQ21	*Series 200/2000 Program Mode User Guide*
AQ22	*Series 200/2000 Program Mode Operator Guide*
AQ26	*Series 100 File Translator*
AQ27	*Series 200/2000 File Translator*
AQ28	*Library Management Manual*
AQ40	*System 3 Conversion Guide*
AQ49	*Network Control Terminal Operation Manual*
AQ50	*Terminal Operations Manual*
AQ52	*Program Checkout Facility Manual*
AQ53	*Communications Processing Facility Manual*
AQ55	*TDS/64 Standard Processor Site Manual*
AQ56	*TDS/64 User Guide*
AQ57	*TDS/64 Processor Programmer Reference Manual*
AQ59	*Unit Record Devices User Guide*
AQ63	*COBOL User Guide*
AQ60	*Interactive Operation Facility*
AQ64	*COBOL Language Reference Manual*
AQ65	*FORTRAN Language Reference Manual*
AQ66	*FORTRAN User Guide*
AQ67	*FORTRAN Mathematical Library*
AQ68	*RPG Language Reference Manual*
AQ69	*RPG User Guide*
AQ72	*Series 200/2000 COBOL to Level 64 COBOL Translator*
AQ73	*IBM COBOL Translator*
AQ77	*File Translation Manual*
AQ82	*BFAS User Guide*
AQ83	*HFAS User Guide*
AQ84	*UFAS User Guide*
AQ85	*Sort/Merge Manual*
AQ86	*Catalog Management Manual*
AQ87	*Library Maintenance User Guide*
AQ88	*I-D-S/II User Guide, Volume 1*
AQ89	*I-D-S/II User Guide, Volume 2*
AQ90	*COBOL Reference Card*
AQ92	*Operator's Reference Card*
AQ93	*RPG Reference Card*
AQ94	*FORTRAN Reference Card*
AQ98	*System Overview Manual*
CQ31	*Error Messages and Return Codes Manual*
CQ35	*Remote Batch Facility*

ACKNOWLEDGMENT

This acknowledgment has been reproduced from the CODASYL COBOL Journal of Development, 1973 as requested in that publication, prepared and published by the CODASYL Programming Language Committee.

"Any organization interested in reproducing the COBOL standard and specifications in whole or in part, using ideas from this document 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 acknowledgment 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 acknowledgment of the source, but need not quote the acknowledgment.

> 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, copyrignted 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

PART I

Section I Input and Maintenance of Source Programs 1-01
Input Enclosures 1-01
Source Libraries 1-02
Creating a Library from Cards 1-02
Creating a Member Interactively............ 1-03
Updating the Source Member 1-04
COBOL Reference Format 1-04
Punched Card Format 1-05
Library Member Text Format 1-07
Interactive Terminal Line Format 1-10
Source Files 1-10

Section II Compilation 2-01
Job Control Language 2-01
SOURCE,INFILE,COMFILE,INLIB and INLIBn Parameters 2-04
CARDID,NCARDID, and DCARDID Parameters 2-06
CASEQ and NCASEQ Parameters 2-08
CKSEQ and NCKSEQ Parameters 2-08
CODAPND and NCODAPND Parameters 2-08
CULIB Parameter 2-08
DCLXREF and NDCLXREF Parameters 2-09
DEBUG and NDEBUG Parameters 2-09
DEBUGMD,NDEBUGMD and DDEBUGMD Parameters .. 2-09
DIAGIN and NDIAGIN Parameters 2-10
DSEGMAX and PSEGMAX Parameters 2-10
EXPLIST and NEXPLIST Parameters 2-10
LEVEL Parameter 2-10
LIST,NLIST and NCLIST Parameters 2-11
MAP and NMAP Parameters 2-11
OBJ and NOBJ Parameters 2-11
OBSERV and NOBSERV Parameters 2-11
PRTFILE Parameter 2-12
PRTLIB Parameter 2-12
STEPOPT Parameter 2-13
SUBOPT and NOPT Parameters 2-13
WARN and NWARN Parameters 2-13
WORK1, WORK2 and WORK3 Parameters 2-13
XREF and NXREF Parameters 2-15
JCL Status 2-15
Libraries Referred to in the COPY Statement 2-16

The Alter Facility 2-16.2
Serial Compilation 2-19
Compiler Limits 2-21
Compiling Level 62 Programs 2-22
Object Code 2-24
Printed Output 2-24
Banner Page 2-25
Program 2-25
User and Project 2-25
Date and Time 2-25
Compiler Version 2-27
User Options and Active Options 2-27
Compilation Level 2-27
Compiler Input 2-28
Program Listing 2-28
Headings 2-33
Source Lines 2-33
Diagnostic Error Messages 2-34
Map Listings and Cross-Reference Listings . 2-37
Data Map and Proc. Definition Listing . 2-37
Cross-Reference Listing (Decl.Order) .. 2-39
Cross-Reference Listing (Alpha.Order) . 2-39
Procedure Map Listing 2-43
PERFORM/ALTER Bucket Listing 2-43
Summary Page 2-45
Summary of Errors 2-45
CU Produced 2-46
Segment List 2-46
Run-Time Package Procedures 2-46
Job Occurrence Report Summary 2-46
Abnormal Compiler Termination 2-47

Section III Linking 3-01
Job Control Language 3-01
Load-module-name Parameter 3-02
ENTRY Parameter 3-03
OUTLIB Parameter 3-03
COMMAND and COMFILE Parameters 3-04
ENTRY Command 3-05
INCLUDE Command 3-05
VACSEG Command 3-05
STEPOPT Parameter 3-06
Library Search Path........................ 3-06
Serial Linkage.............................. 3-06
Operation of $LINKER 3-07
Printed Output 3-08
Banner Page and $LINKER Commands Listing .. 3-08
Included Compile Units 3-08.1
Task Listing 3-08.1
Group Information 3-11
Linkage Report and End Page 3-12
Error Messages 3-14

Section IV Execution 4-01
Program Debugging 4-01
Debugging Code 4-01
Program Checkout Facility 4-03
Dump Analysis 4-04
Structure of the Dump Listing 4-05
The Stack 4-06
Data Division Variables 4-12
General Information 4-13
Job Execution Messages 4-15
Messages Output by the System 4-15
Messages Output by COBOL 4-16
Exception Messages 4-16
Format of Exception Messages 4-17
Exception 09-01 Illegal Decimal Data .. 4-18
Exception 17-02 Out of Array Range 4-19
Exception 06-00 Out of Segment Bounds.. 4-19
Unexpected Return Code 4-19

PART 2

Section V Representation of Data 5-01
Format of Data in Memory.......... 5-01
DISPLAY Data Items 5-02
Packed Decimal Numbers 5-03
Fixed-Point Binary Numbers.......... 5-05
Floating-Point Binary Numbers.......... 5-05
INDEX Data Item 5-06

Section VI Calling and Called Programs 6-01
Transfer of Control 6-02
LINKAGE SECTION and USING Phrase 6-03
The EXTERNAL Phrase 6-04
CALL Identifier.......... 6-05
The CANCEL Statement 6-05
Interface With FORTRAN Programs 6-06
Constraints 6-07
Using Files 6-08
Report Writer 6-08
Guidelines 6-08

Section VII Segmentation 7-01
Methods of Segmentation 7-02
Control of Segmentation by the Programmer 7-02
PROCEDURE DIVISION Segmentation 7-02
DATA DIVISION Segmentation 7-05
Preferred Segment Size 7-06
Automatic Segmentation 7-07
Data Segments 7-08
Procedure Segments 7-10
Internal Segment Numbers 7-10
Declared Working Set.......... 7-11

Section VIII Efficiency 8-01
Data Manipulation Techniques 8-01
Data Description Techniques 8-03

Section IX Files 9-01
Files Names 9-01
Data Management Overriding Rules 9-02
Optional Files 9-04
Close With Lock 9-08
The $POOL Statement 9-08
Multivolume Files 9-09
Multi Logical Unit Files 9-09
Multiple File Tape Volumes 9-11
File Concatenation 9-12
UFAS, BFAS and HFAS 9-13
ORGANIZATION 9-14
APPLY NO-SORTED-INDEX 9-14
APPLY NO-RESIDENT-INDEX 9-14
Error Handling 9-15
FILE STATUS 9-15
Return Code 9-16
Restrictions on Certain File Organizations 9-17
Record Size..................................... 9-17
The ACTUAL KEY Phrase........................... 9-18

Section X Standard Record Formats 10-01
System Standard Format (SSF) 10-02
The Stream Reader, $LIBMAINT and the
COBOL Compiler 10-02
Reading SSF Files in COBOL Programs 10-04
Writing SSF Files in COBOL Programs 10-05
Standard Access Record Format (SARF) 10-05
The Stream Reader, $LIBMAINT and the
COBOL Compiler 10-06
Reading SARF Files in COBOL Programs 10-06
Writing SARF Files in COBOL Programs 10-07
General Points concerning SSF and SARF 10-07
The Output Writer 10-07
Summary of Rules for the SELECT Clause 10-07

Section XI Using Unit Record Files 11-01
Printing 11-01
Using SYSOUT Files for Printing.............. 11-01
Printing Directly 11-05
Form Control................................. 11-05
The LINAGE Clause............................ 11-06
Reading Cards 11-07
Using Standard SYSIN Subfiles 11-07
Reading Cards Directly 11-08
Punching Cards 11-09
Using SYSOUT Files for Cards 11-09
Punching Cards Directly 11-10
ACCEPT,DISPLAY and STOP Literal 11-11
The ACCEPT Statement 11-11
The DISPLAY Statement 11-13

Selection of I/O Device.................... 11-15
The STOP Literal Statement 11-16

*

Using Cassettes................................ 11-17
Types of Cassette File..................... 11-17
GCOS 64 Standard Cassette File............. 11-18
GCOS 62 Standard Cassette File............. 11-18
Foreign Cassette Files..................... 11-19

Section XII Miscellaneous 12-01
Sorting and Merging 12-01
Comparison of COBOL SORT/MERGE and
$SORT/$MERGE 12-01
JCL for COBOL SORT 12-02
User JCL Status 12-03
Switches 12-04
Checkpoint, Restart and Journalization 12-05
Alphabets 12-06
PROGRAM COLLATING SEQUENCE 12-08
SORT and MERGE COLLATING SEQUENCE 12-08
CODE-SET 12-08
HIGH-VALUE LOW-VALUE 12-09
$STEP OPTIONS 12-10
The Report Writer.............................. 12-11
General Concepts........................... 12-11
The DATA DIVISION.......................... 12-12
The PROCEDURE DIVISION..................... 12-12
REPORT Clause in FD........................ 12-13
Summimg Techniques......................... 12-13
The Use of SUM............................. 12-14
SUM Routines............................... 12-15
Page Breaks................................ 12-17
WITH CODE Clause........................... 12-17
Control Footings and Page Format........... 12-18
Floating First Detail Rule................. 12-19
Report Writer Routines..................... 12-20
Table Handling................................. 12-20
Subscripts................................. 12_20
The SET Statement.......................... 12-20
The SEARCH Statement....................... 12-23
Building Tables............................ 12-25
Intermediate Results........................... 12-26
Length of Intermediate Result Fields....... 12-27
Fixed Binary Data Items.................... 12-29
COBOL Run-time Package..................... 12-29
The ON SIZE ERROR Phrase................... 12-29
Communications Programs........................ 12-29
INSPECT and EXAMINE............................ 12-30

Appendix A Sample COBOL Program A-01
Appendix B $COBOL Error Messages B-01
Appendix C $LINKER Error Messages.................................... C-01

Index i-01

ILLUSTRATIONS

Figure 2-1. $COBOL Statement Format 2-02
Figure 2-2. Sample Banner Page 2-26
Figure 2-3. Sample Alter Listing 2-30
Figure 2-4. Sample Source Listing 2-31
Figure 2-5. Sample Expanded Source Listing 2-32
Figure 2-6. Sample Data Map and Procedure Definition Listing . 2-40
Figure 2-7. Sample Cross-Reference Listing (Declaration Order) 2-41
Figure 2-8. Sample Cross-Reference Listing (Alphabetic Order). 2-42
Figure 2-9. Sample Procedure Map Listing and Perform/Alter Bucket Listing 2-44
Figure 2-10. Sample Summary Page 2-45
Figure 3-1. $LINKER Statement Format 3-02
Figure 3-2. Structure of a Linked Program 3-07
Figure 3-3. Sample Banner Page and $LINKER Commands Listing .. 3-09
Figure 3-4. Sample Task Listing 3-10
Figure 3-5. Sample Group Information Listing 3-13
Figure 3-6. Sample Linkage Report and End Page 3-15
Figure 4-1. First Page of Dump 4-07
Figure 4-2. Start of PCS Dump 4-08
Figure 4-3. Ring 3 User Stack 4-09
Figure 4-4. Sample Stack Frame 001 Dump 4-10
Figure 4-5. Sample $LINKER Segment List 4-11
Figure 4-6. Segment Dump 4-14
Figure 7-1. PROCEDURE DIVISION Segmentation................... 7-05
Figure 7-2. DATA DIVISION Segmentation........................ 7-06
Figure 9-1. The Use of Optional Files 9-07
Figure 12-1. Sample GROUP INDICATE Clause...................... 12-17
Figure 12-2. Sample Table Layout in Memory..................... 12-22
Figure 12-3. Rules for the SET Statement....................... 12-22

TABLES

Table 1-1.	COBOL Reference Format	1-05
Table 1-2.	Punched Card Formats	1-06
Table 1-3.	Format of SYSIN Records	1-06
Table 1-4.	Library Member Record Formats	1-08
Table 1-5.	Language Types of COBOL Programs	1-09
Table 2-1.	The Effects of Using CARDID, NCARDID and DCARDID..	2-07
Table 2-2.	Severity Values Set by the Compiler	2-15
Table 2-3.	Compiler Limits	2-21
Table 5-1.	Data Representation in Level 64 System	5-04
Table 6-1.	Data Formats in FORTRAN Called Programs	6-06
Table 9-1.	Specification and Applicability of File Characteristics	9-03
Table 9-2.	Permitted File Organizations	9-05
Table 9-3.	Features Not Available with Certain File Organizations	9-17
Table 10-1.	Summary of Rules for the SELECT Clause	10-08
Table 11-1.	Methods of Producing SYSOUT Print Files	11-03
Table 11-2.	Methods of Producing SYSOUT Punch Files	11-10
Table 11-3.	Variables Governing the Selection of I/O Devices..	11-15
Table 12-1.	High Values and Low Values	12-09
Table 12-2.	Length of Intermediate Result Fields..............	12-28
Table 12-3.	Comparison of INSPECT and EXAMINE..................	12-30

SECTION I

INPUT AND MAINTENANCE OF 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 job run via the Interactive Operation Facility. If a library member is created it may be updated later using Library Maintenance facilities.

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

INPUT ENCLOSURES

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

```
$JOB ...
COBOL SOURCE = *PROG1, CULIB = RES.CULIB;
$INPUT PROG1;
000100 INDENTIFICATION 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 statement). SARF format is described in Section X.

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 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 even if the input enclosure name is different.

SOURCE LIBRARIES

Using an input enclosure in the above manner means that the source program must be re-read from cards 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 cards. The use of libraries is discussed in the following paragraphs (for more details refer to the Library Maintenance User Guide).

Creating a Library Member from Cards

The following example shows the use of the utilities $LIBALLOC and $LIBMAINT to create a library named SL.LIB containing source language members PROG1 and PROG2. An input enclosure is used containing a $LIBMAINT MOVE command and the source program. This input enclosure is read by $LIBMAINT.

```
$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 will set up a library, SL.LIB, with a size of 5 cylinders (this utility need not be used if the library already exists). The MOVE command of the $LIBMAINT utility will then create two library members, PROG1 and PROG2, each containing one of the programs in the input enclosure.

The TYPE = COBOLX option in the MOVE command 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 command of $LIBMAINT under the control of the Interactive Operation Facility. 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 Library Maintenance User Guide for details.

Creating a Library Member Interactively

The Interactive Operation Facility (IOF) may be used to create a source language library member during an interactive job. IOF does not use input enclosures. Instead the source language is entered under the control of the $LIBMAINT command EDIT.

The following example illustrates the use of the EDIT command at an interactive terminal. Program PROG1 is entered under the control of the EDIT command, 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: $JOB...
S: LIBMAINT SL, LIB = SL.LIB;
>>> 09:28 LIBMAINT 20.04 21
C: EDIT;
R: A
I:  IDENTIFICATION DIVISION.
I:  PROGRAM-ID. PROG1.
      .
I:  F
R: W (CBX) PROG1
R: Q
C: RENUMBER PROG1;
C: PRINT PROG1;
000010 IDENTIFICATION DIVISION.
000020 PROGRAM-ID. PROG1.
      .
C: QUIT
<<< 09:30
S:
```

The prompt S: is output by the system. Following this prompt the user can enter any JCL statement at the job enclosure level. When the $LIBMAINT statement is entered, $LIBMAINT ouputs a heading (>>> etc.), containing the time, followed by the C: prompt which invites

the user to enter a $LIBMAINT command. The user then enters the EDIT command after which $LIBMAINT outputs the R: prompt. This invites the user to enter an EDIT request. The user enters an append data request (A) after which $LIBMAINT outputs the I: prompt. This invites the user to enter input data until the escape sequence is encountered. The user then enters the source program. When the user enters the F sequence $LIBMAINT again ouputs the R: prompt and waits for a request. The response "W (CBX) PROG1" requests that the source program just entered be written to a library member named PROG1. The Q request then terminates the EDIT session. $LIBMAINT outputs the C: prompt and waits for 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 Section X) 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 QUIT command. $LIBMAINT outputs the time (e.g. 09:30) immediately before terminating. The system then outputs an S: prompt and the user may enter further JCL statements at the job enclosure level.

UPDATING THE SOURCE MEMBER

A source program can be updated in a batch job using the $LIBMAINT command UPDATE. This command allows the user to insert, replace or delete specific lines which he identifies by giving the 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) the EDIT command should be used.
The UPDATE command is normally used with an input enclosure containing program updates. Since input enclosures are not used by the Interactive Operation Facility, the EDIT command should be used when updating source programs interactively (it may also be used in batch mode).

A full description of the use of the UPDATE and EDIT commands is given in the Library Management Manual and the Library Maintenance User Guide.

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 Table 1-1.

Table 1-1. COBOL Reference Format

Characters	Used for
1-6	Sequence number area
7	Indicator area
8-11	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 cards 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 $COBOL 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 the following paragraphs.

NOTE: The meaning of the term "record format" will be limited to that used in Section X, Standard Record Formats. The term "text format" will be used in the following 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.

Punched Card Format

A program may be punched onto cards in one of the formats shown in Table 1-2 (unless the CONTCHAR option is used in the $INPUT statement - see below).

Table 1-2. Punched Card Formats

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

On punched cards 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 on punched cards is processed from an input enclosure is controlled by the TYPE parameter of the $INPUT statement :

$$\text{TYPE} = \left\{\begin{array}{l}\underline{\text{DATA}}\\ \text{COBOL}\\ \text{DATASSF}\end{array}\right\}$$

The Stream Reader will copy each card in the input enclosure to a temporary subfile of the system file SYS.IN according to the rules shown in Table 1-3. Subfiles in the system file SYS.IN are known as "SYSIN subfiles".

Table 1-3. Format of SYSIN Records

TYPE parameter in $INPUT statement	Format of record in SYSIN (see Section X)	Card columns copied to SYSIN
DATA	SARF	1-80
COBOL	SSF	7-80
DATASSF	SSF	1-80

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 Section II, Compilation). If however, the source program is moved to a library by $LIBMAINT, 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 statement should not specify COBOL. It is recommended that the TYPE parameter be omitted from the $INPUT statement, in which case a TYPE of DATA will be assumed.

The text format of library member records is disscussed in the following paragraphs.

The card formats shown above do not apply when the CONTCHAR option of the $INPUT 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 Job Control Language Reference Manual for more details.

Library Member Text Format

COBOL text in a library member record is held in one of the formats shown in Table 1-4. (Library member records are variable length, therefore the last character position given for area B or the card identifier area is a maximum value.)

Table 1-4. 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
1-6	sequence number area	-	-	-	-
7	Indi-cator area	1	Indi-cator area	1	Indi-cator area
8-11	Area A	2-5	Area A	2-5	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 charac-ter position 255)	
Last 8 Characters	Card iden-tifier	Last 8 Characters	Card iden-tifier	-	-

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 Table 1-5.

Table 1-5. Language Types of COBOL Programs

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

The effect of using these parameters is shown in Table 1-4.

It is recommended that the same language type be used for all COBOL library members in the user's installation. This avoids any confusion that might arise concerning the text format of programs which the user intends to update. The recommended language type is COBOLX. It has the following advantages :

- The sequence number area is removed from the COBOL text (it is stored in the SSF header). This means that the user does 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 $LIBMAINT 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 with DATASSF). This fact, together with the suppression of the sequence number area and card-identifier area results in a compact record which occupies a minimimum of disk space.

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

It is recommended that a sequence number be present if the source program is input from cards. This number will be included in the SSF header when the cards are 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.

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 Table 1-4 except that the size of area B is determined by the number of characters entered by the user on one line (including continuation lines if the continuation character (-) is used).

The COBOL 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.

SOURCE FILES

In addition to the library members mentioned above, the user 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 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 the user's installation. These files can normally be read directly into the compiler using the INFILE parameter.

SECTION II

COMPILATION

This section describes the use of the COBOL compiler. The necessary JCL and the output of the compiler are described in detail.

JOB CONTROL LANGUAGE

The extended JCL statement $COBOL 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 file 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.

Figure 2-1 shows the format of the $COBOL statement. Note that the parameters which are underlined in Figure 2-1 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 $COBOL statement the CKSEQ parameter will be assumed by the compiler. Default parameters are therefore redundant in the $COBOL 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 $COBOL statement. See the Alter Facility, below.

```
COBOL
{ SOURCE = *input-enclosure-name
     [,COMFILE =  {*input-enclosure-name        }]
                  {(sequential-file-description)}

  SOURCE = member-name
    [ {INLIB  =   (library-file-description)} ]
    [,{INLIB1                               } ]
    [ {INLIB2                               } ]
    [ {INLIB3                               } ]
    [,COMFILE =  {*input-enclosure-name        }]
                 {(sequential-file-description)}

  SOURCE = (member-name[,member-name]...)
   [ {INLIB = (library-file-description)} ]
   [,{INLIB1                            } ]
   [ {INLIB2                            } ]
   [ {INLIB3                            } ]

  SOURCE = ('asterisk-name'[,'asterisk-name']...)
     {INLIB = (library-file-description)}
    ,{INLIB1                            }
     {INLIB2                            }
     {INLIB3                            }

           {*input-enclosure-name        }
  INFILE = {(sequential-file-description)}
     [,COMFILE =  {*input-enclosure-name        }]
                  {(sequential-file-description)}

            {*input-enclosure-name        }
  COMFILE = {(sequential-file-description)}
     [,INLIB = (library-file-description)]                  }

[ {CARDID }]      [,{CASEQ }]      [,{CKSEQ }]
[,{NCARDID}]        {NCASEQ}         {NCKSEQ}
[ {DCARDID}]

[,{CODAPND }]     [,CULIB  =  {(library-file-description)}]
  {NCODAPND}                  {TEMP                      }

[, {DCLXREF }]    [,{DEBUG }]      [ {DEBUGMD }]
   {NDCLXREF}       {NDEBUG}       [,{NDEBUGMD}]
                                   [ {DDEBUGMD}]
```

Figure 2-1. $COBOL Statement Format

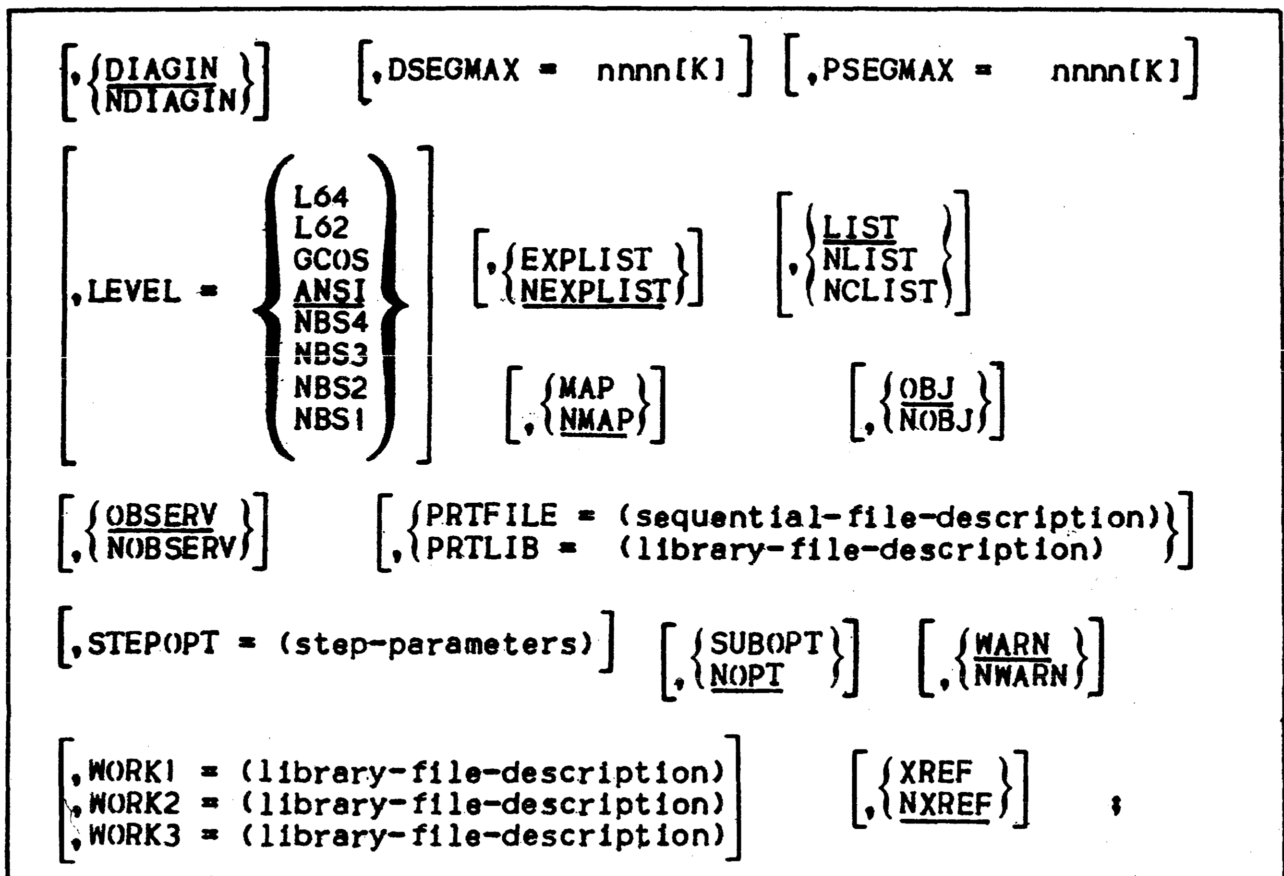


Figure 2-1. $COBOL Statement Format (cont.)

As the $COBOL 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;
COBOL 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 which may be used in the $COBOL statement.

SOURCE, INFILE, COMFILE, 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 $COBOL statement. All of the remaining parameters are optional. SOURCE may appear in the same $COBOL 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:

```
COBOL 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 $COBOL statement as follows:

```
COBOL SOURCE = member-name, INLIB = (library-file-description);
```

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

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

COBOL SOURCE = member-name;
```

The $LIB 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 statements shown in this section 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 statement, the library to be used can

be specified by the INLIBn parameter of the $COBOL statement. In this case the normal search path is overriden by the INLIBn parameter. The statement format is as follows:

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

                                  (INLIB1)
COBOL 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. For example:

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

The chosen search path (given by INLIB or $LIB) 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 which 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:

```
COBOL SOURCE = member-name,
                INLIB = (library-file-description),
          (*input-enclosure-name            )
COMFILE = {(sequential-file-description)};

  [$INPUT input-enclosure-name;   ]
  [.                              ]
  [.                              ]
  [COMFILE commands               ]
  [.                              ]
  [.                              ]
  [$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 $LIBMAINT using the OUTFILE option) The INFILE parameter is used for this purpose as follows:

```
COBOL INFILE = (sequential-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 "asterisk convention" (same as the star convention used by $LIBMAINT). The following statement format is used:

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

```
                                        ( INLIB = (library-file-  )
COBOL SOURCE = (asterisk-name           (         description)    )
          [,asterisk-name]...),         { INLIB1                  };
                                        ( INLIB2                  )
                                        ( INLIB3                  )
```

Note that the library to be used must be specified in the $COBOL statement i.e. no library search is carried out. The $LIB statement can be omitted if a library-file-description is included in the INLIB parameter. Using the asterisk 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 asterisk convention works in exactly the same way as the $LIBMAINT star convention. For a description of the star convention, see the Library Management Manual. The COMFILE parameter cannot be used if the asterisk convention is used.

Both the SOURCE and INFILE parameters can be excluded from the $COBOL 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.

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 Table 2-1.

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	The compiler ignores the rightmost eight character positions	The compiler treats the rightmost eight character positions as COBOL text.	Same as CARDID
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 $LIBMAINT	The compiler treats the rightmost eight character positions as COBOL text. The card-identifier area has already been removed by $LIBMAINT.	Same as NCARDID

The CARDID, NCARDID and DCARDID parameters should normally be omitted from the $COBOL 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.

CASEQ and NCASEQ Parameters

CASEQ requests that all small letters which are not included in a non-numeric literal are processed as if they were capital letters (default parameter).

NCASEQ requests that small letters are different from capital 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 small or capital letters, or both). Thus the user-word "abC" is different from the user-word "aBC", whereas "move" is the same reserved word as "MOVE".

The letters a,b,c,d,e,p,r,s,v,x and z in PICTURE character strings are always taken to be their corresponding capital 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.

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). 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.

CODAPND and NCODAPND Parameters

CODAPND requests that, if there is only one generated code segment, an attempt is made to put it 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 (TDS) as this reduced the amount of disk activity required for program loading.

CULIB Parameter

The CULIB parameter specifies the library in which the resulting compile unit is to be stored. A library-file-description or the word TEMP may be used in the CULIB parameter.

If a library is specified, it must have been allocated previously by, for example, the $LIBALLOC utility, unless the library-file-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 temporary member of a system 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 temporary compile units produced with no CULIB parameter, or with CULIB = TEMP, the compile unit library TEMP should normally be present in the $LIB 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.

DCLXREF and NDCLXREF Parameters

The DCLXREF parameter produces a cross reference listing in declaration order. 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).

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 Section IV.

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.

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 Reference Manual (default parameter).

DIAGIN and NDIAGIN Parameters

DIAGIN specifies that all errors are embedded in the alter, source and/or expanded listings (default parameter).

NDIAGIN specifies that alter errors are embedded in the alter listing, but that other (purely COBOL) errors are listed after the source and/or expanded listing.

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 SOURCE-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 Section VII, Segmentation.

EXPLIST and NEXPLIST Parameters

EXPLIST specifies that an expanded source listing is to be produced. The source listing, if produced, includes COPY and REPLACE statements. In the expanded source listing COPY and REPLACE statements are not printed and replaced and/or deleted words are actually replaced or deleted according to the REPLACE statement or the REPLACING clause. See Program Listing, below.

NEXPLIST specifies that an expanded listing is not to be produced (default parameter).

LEVEL Parameter

The LEVEL parameter specifies that the compilation level is full Level 64, Level 62, GCOS level, full ANSI 74 standard, high NBS level, high intermediate NBS level, low intermediate NBS level, or low NBS level. All features beyond the specified level are flagged as fatal errors (****). No object code is then produced. This parameter is not accepted if the level specified is above the maximum level specified for the installation. Unless modified by the Field Engineering, the default level is ANSI 74, and the maximum level for the installation is L64.

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

Computer-names other than LEVEL-64, but starting with LEVEL-6 cause a warning message (**). When SOURCE-COMPUTER is LEVEL-62 the default device for ACCEPT and DISPLAY is CONSOLE, the default meaning of COMPUTATIONAL is DISPLAY, so called hexadecimal embedded literals are accepted according to Level 62 COBOL syntax, and compilation level is restricted to GCOS.

LIST, NLIST and NCLIST Parameters

NLIST specifies that the source program listing is not to be produced. However, unless NDIAGIN 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).

MAP and NMAP Parameters

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.

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 default parameter assumed if NOBJ is not specified.

OBSERV and NOBSERV Parameters

The NOBSERV parameter suppresses all observation diagnostic messages in the program listing. However, the number of observation messages is printed in the compilation summary page and in the Job Occurrence Report. 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 is the default assumed if NOBSERV is not specified.

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, $SYSOUT, $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. For example, the user could specify output to tape:

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

In this case, only the Job Occurrence Report will be printed at the end of job execution. (Note that the Job Occurrence Report 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, all listings are stored in a single file.

If the SYSOUT file is full, the compilation terminates with the following message in the Job Occurrence Report:

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

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

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, 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 Job Occurrence Report. See PRTFILE Parameter, above.

STEPOPT Parameter

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

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

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 Section VIII, Efficiency, for more information on the use of SUBOPT.

WARN 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 is printed in the compilation summary page and in the Job Occurrence Report, 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 assumed if NWARN is not specified.

WORK1, WORK2 and WORK3 Parameters

The compiler does not normally use files for its work areas ; instead it works directly in backing store. When the backing store cannot accomodate the required work areas, a fatal error message is printed:

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

Using the WORKn parameters 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. 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 WORK1, WORK2, and WORK3 parameters.

The first two work files each require a capacity of upto 150 bytes per source line ; the third requires up to 300 bytes per source line.

Example of the use of temporary work files:

```
COBOL
WORK1 = (MYFILE1, FILESTAT = TEMPRY, SIZE = 20),
WORK2 = (MYFILE2, FILESTAT = TEMPRY, SIZE = 20),
WORK3 = (MYFILE3, FILESTAT = TEMPRY, SIZE = 40),
SOURCE = MYPROGRAM;
```

In this example, the compiler will use MYFILE1, MYFILE2 and MYFILE3 which will be allocated temporarily to the job with a size of 20, 20 and 40 cylinders on the RESIDENT disk pack. If this size is not large enough for the compilation, for any of the files, it will be increased automaticaly 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 which 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 3 work files. If SIZE is not specified, the compilation will start with 4 cylinders for each file, and it could happen that enough space is not available for the compilation. 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 WORK3 if desired. They should be preallocated in the following way.

```
PREALLOC        external-file-name,

                BFAS = (LINKQD = (TYPE = NONE,
                       BLKSIZE = nnnnn,
                       RECSIZE = nnnnn,
                       RECFORM = FB,
                       NODEL)),

                FILESTAT = CAT,
                DEVCLASS = device-class,
                GLOBAL   = (MEDIA = media-list),
                SIZE     = nnnnn;
```

The values of BLKSIZE and RECSIZE must both be 4240 for WORK1 and WORK2. They must both be 2046 for WORK3. The DEVCLASS, MEDIA and SIZE parameters are explained in the Data Management Utilities Manual under the PREALLOC utility. The above example preallocates a catalogued file, but uncatalogued files may also be used.

The use of permanent work files has two advantages: the cost of dynamic formatting is avoided, and only one compilation using a given set of work files can be active at a given moment (useful for queuing compilations submitted from an IOF terminal).

Compilation time will increase by about 3% if the WORKn parameters are used.

XREF and NXREF Parameters

The XREF parameter produces a cross reference listing in alphabetic order. 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).

JCL Status

At the end of the compilation, subsequent job processing may be determined by testing with the $JUMP statement a severity value set by the compiler or by the system. Severity values are printed in the Job Occurrence Report. The possible values are shown in Table 2-2.

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 which has just been compiled:

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

See User JCL Status, Section XII, for more details.

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 $COBOL statement or in an earlier $LIB statement. For example:

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

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

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

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

If the $LIB statement is used, and the INLIB = (library-file-description) parameter is not included in the $COBOL statement, the COPY statement may optionally contain OF/IN INLIB1/INLIB2/INLIB3. In this case the text to be copied is in the specified library in the $LIB 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 will be ignored.

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

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

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

JCL:
    LIB  SL, INLIB1 = MY_LIBRARY_A,
             INLIB2 = MY_LIBRARY_B,
             INLIB3 = MY_LIBRARY_C;
    COBOL SOURCE = MY_MEMBER, INLIB = 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 INLIB parameter of $COBOL or in the INLIBn parameters of $LIB. The libraries whose names would be INLIB, INLIB1, INLIB2 or INLIB3 may not be referenced by their actual name.

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

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,...);
COBOL COMFILE = *MY_ALTER;
$INPUT MY_ALTER;
COMPILE;
R MY_PROGRAM
12D
16D
Q
$ENDINPUT;
```

The command file is normally specified in the COMFILE parameter of the $COBOL 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 COMPILE command consists of the word "COMPILE", optional $COBOL 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	
CODAPND	NCODAPND	
DCLXREF	NDCLXREF	
DEBUG	NDEBUG	
DEBUGMD	NDEBUGMD	DDEBUGMD
DIAGIN	NDIAGIN	
EXPLIST	NEXPLIST	
LIST	NLIST	NCLIST
MAP	NMAP	
OBJ	NOBJ	
OBSERV	NOBSERV	
SUBOPT	NOPT	
WARN	NWARN	
XREF	NXREF	

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 $LIBMAINT EDIT requests, namely:

A	append
C	change
D	delete
I	insert

N	no request
Q	quit
R	read member
S	substitute
"	comment

Apart from Q and R, 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 Q request is not mandatory for the COBOL compiler. It can be ommitted in all cases, but the programmer may wish to include the Q 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:

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

This request is different from the R request of $LIBMAINT EDIT. A series of R requests input to the EDIT command of $LIBMAINT will cause the specified set of members to be concatenated. This does not happen when a series of R requests is input to the compiler. Each R request will result in a separate compilation.

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 statement under the INLIB1, 2 or 3 parameters respectively. INLIB means that the member is to be searched for in the library specified in the INLIB = (library-file-description) parameter of the $COBOL statement. In the absence of these keywords, member-name is searched for first in the libraries specified in the $LIB statement, according to the implied searching rules, then in the INLIB library. 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.

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.

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 all 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). Each program is compiled in its turn.

The upper level may be at the level of the SOURCE parameter of the $COBOL 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:

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

could also be written:

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

Note that SOURCE or INFILE can be used in the same $COBOL 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 $COBOL statement:

```
COBOL 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:

```
COBOL 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 $COBOL 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 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.

COMPILER LIMITS

The COBOL compiler has the limits shown in Table 2-3.

Table 2-3. Compiler Limits

Variable	Limit
Number of user-names (data, filler, paragraphs)	28000
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 (ANS standard is 120)	256 digits
Maximum size of a code segment	32000 bytes
Maximum size of an edited item	256 char.

Limits which are in excess of the ANS standard are available only if the LEVEL = L64 parameter is included in the $COBOL statement; otherwise the ANS standard limits apply.

COMPILING LEVEL 62 PROGRAMS

Level 62 COBOL programs may be submitted to the Level 64 COBOL compiler for compilation and execution on the Level 64 computer. Programs are compiled as Level 62 programs by the compiler when the two following conditions are met:

- The SOURCE-COMPUTER paragraph specifies LEVEL-62 as computer-name, and
- The compilation is requested with the LEVEL = L62 parameter in the $COBOL statement.

The program will not compile if any of the following conditions exists:

- The Level 62 communications feature is used.
- The Level 62 extended indexed organization is used.
- A file is referenced in the PROCEDURE DIVISION USING...header or as a CALL argument.
- OPEN REVERSED is used.
- A non-numeric comparison references operands of different usages.

The program will compile but may not execute correctly if any of the following conditions exists:

- The commercial at sign (@) is used in column 7 (it is processed as an asterisk).
- The JOB-LINKAGE SECTION header is used (the compiler processes items described in that section as WORKING-STORAGE SECTION items).
- RERUN... EVERY condition-name and its related SET condition-name is used (the compiler ignores this clause and statement).
- USE AFTER... ON DATA or PROGRAM ERROR is used (the compiler ignores these sections).
- STATUS KEYs are used that do not have the same meaning in both implementations.
- An attempt is made to read a record of an indexed file that has been written in the same run (it will be retrieved in Level 64, whereas it would not in Level 62).

There may also be cases where the precision of intermediate results differs from one machine to the other.

If none of the above conditions exists the program will compile and execute correctly in spite of options or defaults that are Level-62 specific. In some circumstances, the option is ignored, and will have to be replaced by appropriate JCL statements. The special actions taken by the compiler when compiling in the Level-62 mode are:

- DEBUG-ITEM-ERR is accepted as a group item subordinate to the DEBUG-ITEM special register.
- The format of so-called "embedded hexadecimal values within nonnumeric literals" is that of Level 62 (and therefore, the Level 64 format is not allowed).
- The LITERAL-WITHIN clause is ignored (but the Level 64 compiler accepts the apostrophe as well as the quote as a nonnumeric literal delimiter).
- The OPTIONAL phrase of the SELECT clause is allowed for any file organization.
- The BLOCK FORMAT and BLOCK LENGTH phrases of the SELECT clause are ignored.
- NEW INDEXED is accepted as equivalent to INDEXED.
- The default file organization is LEVEL-62 instead of UFF
- RERUN-FILE is accepted as the checkpoint file name.
- The APPLY MARK SENSE clause is ignored.
- USAGE IS COMP is equivalent to USAGE IS DISPLAY.
- Unsigned COMP-3 items are allocated with a sign position (i.e. behave as Level 64 COMP-8 items).
- The REDEFINES clause need not reference the first redefined item.
- The default device for ACCEPT and DISPLAY is CONSOLE.
- The internal-file-name suffixes are accepted : those whose first two characters are "PR" are processed as PRINTER : when the suffix is REPORT, it is processed as SYSOUT.

When the above actions are based on syntax (i.e. are not the result of default application) they are shown by appropriate diagnostics. The fact that the program is processed as a Level 62 program is shown in the banner page of the compilation listing.

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 FI.
23          LEC.
24              READ FI AT END GO TO FIN.
            .
            .
29          FIN.
30              CLOSE FI.
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 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 or if one statement is used which is equivalent to several simple verbs (e.g. MOVE CORRESPONDING) 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).

OPEN,CLOSE,OCCURS DEPENDING,ON etc. use subroutines which are generated at the end of the main object code in what is called the "Epilogue".

PRINTED OUTPUT

The following paragraphs describe the printed 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

Banner Page

A sample banner page is shown in Figure 2-2. 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

where: yy is year
ddd is day
hh is hour
mm are minutes
ss are seconds.

This program-name is used in forming the name of the listing file when a permanent SYSOUT file is being used.

USER AND PROJECT

These items show the USER and PROJECT specified in the $JOB statement.

DATE AND TIME

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

```
********************************************************************************************************************
********************************************************************************************************************
**** GCOS L64                                                                                                   ****
****                                       C  O  B  O  L                                                        ****
****                                                                           VERSION: 50     DATED: MAR 10, 1978 ****
*****************************************************************************************2     ***********************
********************************************************************************************************************

PROGRAM: FIND-DAY

USER: BOURGAIN

PROJECT: BOURGAIN

DATE: 03/31/78

TIME: 13:42:36

COMPILER VERSION: L64 COBOL V-50.2

USER OPTIONS: COMFILE LIB=1 LEVEL=L64 DCLXREF XREF EXPLIST

ACTIVE OPTIONS: OBJ, NDEBUG, WARN, OBSERV, NMAP, DCLXREF, XREF, LIST, EXPLIST, CKSEQ, CARDID, CASEQ, DIAGIN,
         NCODAPND, NOPT, DDEBUGMD, PSEGMAX=4096(BYTES), DSEGMAX=4096(BYTES).

COMPILATION LEVEL: L64

COMPILER INPUT:
  ALTER FILE
    RSTR (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 RSTR (H_INLIB1)
    CD=01/23/78 CT=10:35:24 MD=03/07/78 MT=13:16:12 SL=DAT MN=11 NM=FIND-DAY
  COPY FILE (COPIED TEXT ON LINES 38 THROUGH 49)
    DAYS IN RSTR (H_INLIB1)
    CD=01/23/78 CT=10:35:24 MD=01/23/78 MT=10:35:24 SL=DAT MN=00 NM=DAYS
```

Figure 2-2. Sample Banner Page

COMPILER VERSION

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

```
COMPILER VERSION : L64 COBOL V-50
```

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 : L64 COBOL V-50.6
```

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 : L64 COBOL V-50.6-2
```

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

```
COMPILER VERSION : L64 COBOL V-50.6-2
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 $COBOL statement. ACTIVE OPTIONS lists all of the compiler default parameters in addition to those specified by the user.

COMPILATION LEVEL

This shows the level of COBOL which is expected by the compiler and is derived from the LEVEL parameter in the $COBOL statement and the SOURCE-COMPUTER paragraph in the source program. The possible values are: L64, L62, GCOS, ANSI, NBS4, NBS3, NBS2 or NBS1.

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 verb, the file from which text is copied is also identified.

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

COPY FILE (COPIED TEXT ON LINES 21 THROUGH 26)

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 appear only if the file is an SSF file. 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)
MN = modification number. This is zero for a new file or subfile and is augmented by one for each update.
NM = name (normally the same as the subfile name.)

Program Listing

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

- Alter listing (Figure 2-3).
- Source listing (Figure 2-4).
- Expanded source listing (Figure 2-5).
- 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 and expanded source listing.

The source listing is printed if there is no NLIST parameter specified in the $COBOL statement. The listing incorporates any alterations specified in a command file and (if there is no NCLIST parameter in the $COBOL statement) any text referred to in COPY statements. However, the effects of a REPLACE statement in the CONTROL DIVISION or a REPLACING clause in a COPY statement are not shown. That is, the specified words are listed in their original form, with an indication of the first and last word replaced or deleted.

The expanded source listing is only produced if the EXPLIST parameter is specified in the $COBOL statement. It is printed after the source listing. The expanded listing has the following differences from the source listing.

- COPY and REPLACE statements are not printed.

- The effects of a REPLACE statement or REPLACING clause are shown; that is, the specified words are listed in their new form.

If both the source and the expanded listings are requested but there is neither a COPY statement nor a REPLACE statement in the program, only one listing is output.

The internal line numbers of alter listings, source listings and expanded listings are distinguished in the following way.

- 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. If there is an expanded listing as well as a source listing, the internal line numbers in the source listing are prefixed by "S." . However, if there is no expanded listing this prefix is not printed.

- Expanded source listing. The internal line numbers of expanded source listings are not prefixed.

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

```
         COBOL     V-50.2        X93.1      LISTING  BOURGAIN BOURGAIN              13:42:36  MAR 31, 1978   PAGE      2
FIND-DAY            ALTER LISTING


     A.1 ----->          COMPILE;
     A.2
     A.3            R: R FIND-DAY
     A.4
     A.5            R: /DATA/S/DIVISION/&./
     A.6
     A.7            R: /01  DITWEEK-TAB/,/SUNDAY/C                COMMENT
                                              1
   * 1 1-44  TEXT FOLLOWS THE 'A', 'C', 'I' CR 'Q' COMMAND ON THE LINE. TEXT IS IGNORED.

     A.8            I:               COPY DAYS
     A.9            I:                 REPLACIN( == PIC X(8) == BY == PIC X(10) ==.
    A.10            I: ¢F
```

Figure 2-3. Sample Alter Listing

```
           COBOL     V-50.2        X93.1      LISTING  BOURGAIN BOURGAIN              13:42:36  MAR 31, 1978   PAGE       3
FIND-DAY              SOURCE LISTING
    ILN          XLN  TEXT  7-10--------20--------30--------40--------50--------60--------70--........<-

    S.1            1         IDENTIFICATION DIVISION.
    S.2            2         *                        THIS ROUTINE, STARTING FROM A DATE, GIVES        <-
    S.3            3         *                        THE DAY IN THE WEEK CORRESPONDING TO THE
    S.4            4         *                        DATE
    S.5            5         PROGRAM-ID. FIND-IAY.
    S.6            6         *
    S.7            7         ENVIRONMENT DIVISIJON.
    S.8            8         CONFIGURATION SECTION.
    S.9            9         SOURCE-COMPUTER. LEVEL-64
    S.10          10         OBJECT-COMPUTER. LEVEL-64.
    S.11          11         *
    S.12         *12         DATA DIVISION.                                                              <-
    S.13          13         *
    S.14          14         WORKING-STORAGE SICTION.
    S.15          15         *                        TEMPORARIES
    S.16          16         01  X PICTURE 9(1().
    S.17          17         01  Y PICTURE 9(5).
    S.18          18         *                        TOTAL NUMBER OF DAYS PRECEDING THE MONTH
    S.19          19         *                        (SHOWN BY ITS ORDINAL NUMBER IN THE LIST)
    S.20          20         *                        IN THE YEAR
    S.21          21         01  PREC-D-TAB.
    S.22          22             02  FILLER PI( 999 VALUE   0.
    S.23          23             02  FILLER PI( 999 VALUE  31.
    S.24          24             02  FILLER PI( 999 VALUE  59.
    S.25          25             02  FILLER PI( 999 VALUE  90.
    S.26          26             02  FILLER PIC 999 VALUE 120.
    S.27          27             02  FILLER PIC 999 VALUE 151.
    S.28          28             02  FILLER PIC 999 VALUE 181.
    S.29          29             02  FILLER PIC 999 VALUE 212.
    S.30          30             02  FILLER PI( 999 VALUE 243.
    S.31          31             02  FILLER PI( 999 VALUE 273.
    S.32          32             02  FILLER PI( 999 VALUE 304.
    S.33          33             02  FILLER PIC 999 VALUE 334.
    S.34          34         01  PREC-D-TAB-REI REDEFINES PREC-D-TAB.
    S.35          35             02  PRECEDING-DAYS PIC 999 OCCURS 12.
    S.36          36         *                        TABLE GIVING THE NAME OF THE DAYS IN THE
    S.37          37         *                        WEEK
    S.38           .             COPY DAYS
    S.39           .               REPLACING == PIC X(8) == BY == PIC X(10) ==.
    S.40         ..1         01    THER-UNUSED.
    S.41         ..2                 FILLER PIC X.
    S.42         ..3                 FILLER COMP-1 SYNC.
    S.43         ..4         01     TWEEK-TAB.
    S.44         ..5                   FILLER PI( X(8)  VALUE "LUNDI    ".

                                          1   2
  1 1-32  FIRST WORD OF TEX  EPLACED (OR DELETED).
  2 1-33  LAST WORD OF TEXT  PLACED (OR DELETED).

    S.45         ..6               ?  FILLER PI(  (10) VALUE "MARDI    ".
    S.46         ..7               ?  FILLER PIC  (10) VALUE "MERCREDI".
    S.47         ..8               ?  FILLER PIC  (10) VALUE "JEUDI    ".
    S.48         ..9               ?  FILLER PI( .(10) VALUE "VENDREDI".
    S.49        ..10              02  FILLER PIC X(10) VALUE "SAMEDI   ".
```

Figure 2-4. Sample Source Listing

```
          COBOL    V-50.2        X93.1     LISTING  BOURGAIN BOURGAIN              13:42:36  MAR 31, 1978   PAGE      6
FIND-DAY                EXPANDED SOURCE LISTING
   ILN        XLN  TEXT  7-10--------20--------30--------40--------50--------60--------70--........<-

     1          1          IDENTIFICATION DIVISION.
     2          2         *                        THIS ROUTINE, STARTING FROM A DATE, GIVES        <-
     3          3         *                        THE DAY IN THE WEEK CORRESPONDING TO THE
     4          4         *                        DATE
     5          5          PROGRAM-ID. FIND-DAY.
     6          6         *
     7          7          ENVIRONMENT DIVISION.
     8          8          CONFIGURATION SECTION.
     9          9          SOURCE-COMPUTER. LEVEL-64
    10         10          OBJECT-COMPUTER. LEVEL-64.
    11         11         *
    12        *12          DATA DIVISION.                                                              <-
    13         13         *
    14         14          WORKING-STORAGE SECTION.
    15         15         *                        TEMPORARIES
    16         16          01  X PICTURE 9(10).
    17         17          01  Y PICTURE 9(5).
    18         18         *                        TOTAL NUMBER OF DAYS PRECEDING THE MONTH
    19         19         *                        (SHOWN BY ITS ORDINAL NUMBER IN THE LIST)
    20         20         *                        IN THE YEAR
    21         21          01  PREC-D-TAB.
    22         22              02  FILLER PIC 999 VALUE   0.
    23         23              02  FILLER PIC 999 VALUE  31.
    24         24              02  FILLER PIC 999 VALUE  59.
    25         25              02  FILLER PIC 999 VALUE  90.
    26         26              02  FILLER PIC 999 VALUE 120.
    27         27              02  FILLER PIC 999 VALUE 151.
    28         28              02  FILLER PIC 999 VALUE 181.
    29         29              02  FILLER PIC 999 VALUE 212.
    30         30              02  FILLER PIC 999 VALUE 243.
    31         31              02  FILLER PIC 999 VALUE 273.
    32         32              02  FILLER PIC 999 VALUE 304.
    33         33              02  FILLER PIC 999 VALUE 334.
    34         34          01  PREC-D-TAB-RED REDEFINES PREC-D-TAB.
    35         35              02  PRECEDING-DAYS PIC 999 OCCURS 12.
    36         36         *                        TABLE GIVING THE NAME OF THE DAYS IN THE
    37         37         *                        WEEK
    38        ..1          01  OTHER-UNUSED.
    39        ..2              02 FILLER PIC X.
    40        ..3              02 FILLER COMP-1 SYNC.

                                   1
 * 1 2-199 A 1 BYTE TYPE 2 FILLER WAS ALLOCATED TO ALIGN THIS SYNCHRONIZED ITEM (SEE REFERENCE MANUAL).

    41        ..4          01  DITWEEK-TAB.
    42       *..5              02  FILLER                     PIC X(10)         <-
    43       *..5                                 VALUE "LUNDI    ".
    44        ..6              02  FILLER PIC X(10) VALUE "MARDI    ".
    45        ..7              02  FILLER PIC X(10) VALUE "MERCREDI".
    46        ..8              02  FILLER PIC X(10) VALUE "JEUDI    ".
    47        ..9              02  FILLER PIC X(10) VALUE "VENDREDI".
    48        ..10             02  FILLER PIC X(10) VALUE "SAMEDI   ".
    49        ..11             02  FILLER PIC X(10) VALUE "DIMANCHE".
    50        -46          01  DITWEEK-TAB-RED REDEFINES DITWEEK-TAB.
```

Figure 2-5. Sample Expanded Source Listing

No source listing will be produced when the NLIST parameter is specified in the $COBOL statement and no COPY text will be printed when the NCLIST parameter is specified. An error listing is printed whenever the NLIST and/or NDIAGIN parameters are specified. When the NLIST parameter is specified and NDIAGIN is not specified the error listing contains any diagnostic error messages generated by the compiler, together with the relevant lines of source program. When the NDIAGIN parameter is specified (with or without NLIST) the error listing contains any diagnostic error messages generated by the compiler, but does not contain any lines of source program. Instead the line number and column number of the spurious source code is printed with each error message.

The layout of the source listing and expanded 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).

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.

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, for lines following COPY statements, when the part of the line following a COPY statement is repeated. 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). In the expanded source listing, periods, asterisks and minus signs have the same meaning as above; in addition, the asterisk means that the line has been modified as the result of the application of the REPLACE statement of the CONTROL DIVISION, or of the REPLACING clause of a COPY statement.

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 both the source and expanded source listings, whenever a line is not in 80 column format (with source from columns 8 to 72) an arrow <- is printed at the right of the line. The arrow appears in the two columns following the card identifier area.

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 and expanded source listings. However, if the NDIAGIN parameter is specified in the $COBOL statement, all errors will be listed after the source and/or expanded 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 $COBOL statement. The following example contains an observation message:

```
14                          FD F1
15                              LABEL RECORD STANDARD
16                              DATA RECORD A1 A2.
17                          01 A1 PIC X(80).
18                          01 A2 PIC X(80).
19                          01 A0 PIC X(80).

                                |
      * 1 3-191 RECORD DESCRIPTION ASSUMED TO BE DATA RECORD
                FOR 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 $COBOL 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 5-148 THIS RECEIVING ITEM MAY BE TRUNCATED ON 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 6-2   ITEM NOT DECLARED
     * 1 5-164 SYNTAX CHECK DISCONTINUED
```

The "SYNTAX CHECK DISCONTINUED" observation message indicates that the compiler has interrupted analysis from this point, until it encounters a recognizable sequence (point, paragraph, section, etc.). The compiler then resumes its analysis indicating "SYNTAX CHECK 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.unrecoverable 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 $COBOL statement).

The following example contains a serious 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 LEVEL-64 SPECIFIC FEATURE, NOT
                INCLUDED IN THE CURRENT COMPILATION LEVEL.
```

In this example the compilation has been requested without the LEVEL = L64 parameter in the $COBOL 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 UNTIL A9 EQUAL TO B9.<-

                             1          2         3
   *  1  5-165  SYNTAX CHECK RESUMED.
****  2  5-162  THIS FEATURE IS A NBS HIGH INTERMEDIATE FEATURE, NOT
                INCLUDED IN THE CURRENT COMPILATION LEVEL.
 ***  3  6-2    ITEM NOT DECLARED.
   *  3  5-164  SYNTAX CHECK DISCONTINUED.
```

The error under PERFORM results from an error on a previous line. The error under UNTIL results from LEVEL = NBS2 used in the $COBOL statement. The arrow at the end of the line indicates that there is no blank character after the terminating period. Therefore, the last eight characters are ignored and the compiler recognized EQUA instead of EQUAL. Hence the error messages.

Map Listings and Cross-Reference Listings

Depending on the parameters specified by the user in the $COBOL 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 (declaration order) 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 $COBOL 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.
- DCLXREF produces a cross-reference listing (declaration order).
- XREF 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 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)
- usage
- picture string (if applicable)
- internal line number of the line in which the identifier is defined.

A sample data map and procedure definition listing is shown in Figure 2-6. 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 the 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. 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 Section III, Linking).

Under USAGE there is a description 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.

- DISP (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.

- INX-DATA indicates an index data item.

- ALPH-NM indicates an alphabet-name.

- MNEM-NM indicates a mnemonic-name.

- REPORT indicates a report-name

- INX indicates an INDEXED file.

- REL indicates a RELATIVE file.

- SEQ indicates a sequential file.

- ...-SEQ where the ACCESS MODE IS SEQUENTIAL clause is used.

- ...-DYN where ACCESS MODE IS DYNAMIC.

- ...-RAN where ACCESS MODE IS RANDOM.

- SORT indicates a sort-file (SD appeared under LN).

- INDX-NM indicates an index-name declared by use of the INDEXED BY clause in a table description.

- UNDEFINED indicates a data-name or paragraph-name which is referenced, but never declared.

Under the PIC-STRING heading there is a simplified version of the explicit or implicit picture clause. When the picture string includes editing symbols, only the word EDITED is printed.

Under the heading DEF the internal line number at which the identifier is defined is shown.

The data map and procedure definition listing also contains a list of all paragraph-names and SECTION names (NAME) together with the internal line numbers at which these names are defined (DEF). 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 Figure 2-6 for an example. The type of name is indicated under USAGE by PARA-NM or SECT-NM. This list is used in conjunction with the procedure map listing. The procedure map listing contains memory adresses and internal line numbers. These internal line numbers can be related to the containing paragraph or SECTION using this listing.

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 internal line numbers for those lines which refer to the identifier. See Figure 2-7 for an example.

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.LINES. More than one reference to the same identifier on a single line is shown by a plus sign following the internal line number. An ellipsis (...) indicates that some referencing lines are missing.

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. See Figure 2-8 for an example.

An additional piece of information in the cross-reference listing in alphabetical order is the 01 level data-name which appears in parenthesis after each non-01 level data-name. This shows record-name to which each data-name belongs.

```
             COBOL     V-50.2        X93.4      LISTING  BOURGAIN BOURGAIN                13:49:48  MAR 31, 1978   PAGE      10
CALENDAR              DATA MAP AND PROCEDURE DEFINITION LISTING
LN NAME                                       PN  ADDRESS   USAGE    PIC-STRING     DEF.

02 DIGIT-EIGHT                                  1 :00254    DISP     X(27)          78
02 DIGIT-NINE                                   1 :0026F    DISP     X(27)          79

01 HEADER-DIGITS-R                              1 :0017C    GROUP    X(270)         80
02 DIGITS                                       1 :0017C    GROUP    X(27)          81
03 DIGIT                                        1 :0017C    DISP     X(3)           82

01 MONTH-STATUS                                 1 :00294    GROUP    X(96)          84

01 MONTH-STATUS-R                               1 :00294    GROUP    X(96)          99
   MONTH                                        1 :002F8    INDX-NM                 100
   FIRST-MONTH                                  1 :00300    INDX-NM                 100
02 MONTH-ST                                     1 :00294    GROUP    X(8)           100
03 DAY-OF                                       1 :00294    DISP     9(1)           101
03 MONTH-DAY                                    1 :00295    DISP     9(2)           103
03 MAX-MONTH-DAY                                1 :00297    DISP     9(2)           105
03 YEAR-DAY                                     1 :00299    DISP     9(3)           107

01 WEEK-DAYS-I                                  1 :00308    GROUP    X(70)          110

01 WEEK-DAYS-R                                  1 :00308    GROUP    X(70)          119
02 WEEK-DAY                                     1 :00308    DISP     X(10)          120

01 L                                          1 0 :00000    COMP-2   FIX BIN (31)   125

01 O                                          2 0 :00000    DISP     VARIABLE       126

   INIT                                                     PARA-NM                 129
   EXIT-P                                                   PARA-NM                 143
   STOP-R                                                   PARA-NM                 145
   YEAR-CALENDAR                                            PARA-NM                 150
   INIT-MONTHS                                              PARA-NM                 161
   TITLES                                                   PARA-NM                 178
   YEAR-TITLE-LINES                                         PARA-NM                 197
   NORMAL-LINES                                             PARA-NM                 213
   NORMAL-LINE                                              PARA-NM                 219
```

Figure 2-6. Sample Data Map and Procedure Definition Listing

```
           COBOL     V-50.2       X93.1     LISTING  BOURGAIN BOURGAIN           13:42:36  MAR 31, 1978    PAGE      9
FIND-DAY            CROSS-REFERENCE LISTING (DECLARATION ORDER)
LN NAME                                        PN   ADDRESS    USAGE    PIC-STRING   DEF.   REF.  LINES


77 TALLY                                           1:00010    DISP     9(5)                 NOREF

01 X                                               1:00064    DISP     9(10)        16     95 96 97 98 99 100 105

01 Y                                               1:00070    DISP     9(5)         17     105 106 107 109 110 111

01 PREC-D-TAB                                      1:00078    GROUP    X(36)        21     NOREF

01 PREC-D-TAB-RED                                  1:00078    GROUP    X(36)        34     NOREF
02 PRECEDING-DAYS                                  1:00078    DISP     9(3)         35     89

01 OTHER-UNUSED                                    1:000A0    GROUP    X(4)         38     NOREF

01 DITWEEK-TAB                                     1:000A4    GROUP    X(70)        41     NOREF

01 DITWEEK-TAB-RED                                 1:000A4    GROUP    X(70)        50     NOREF
02 DAY-IN-THE-WEEK                                 1:000A4    DISP     X(10)        51     111

01 SPLIT-DATE                                      1:000F0    GROUP    X(8)         54     81 83
02 CENTURY                                         1:000F0    DISP     9(2)         55     84
02 SHORT-DATE                                      1:000F2    GROUP    X(6)         56     83
03 MONTHR                                          1:000F4    DISP     9(2)         58     89 94
03 DAY-OF-MONTH                                    1:000F6    DISP     9(2)         59     88
03 DAY-OF-MONTH-X                                  1:000F6    DISP     X(2)         60     82

01 YEAR                                            1:000F0    DISP     9(4)         61     90 94+ 95 97 99

01 DAYS-IN-THE-ERA                                 1:000F8    DISP     9(10)        65     87 96 98 100 105

01 FULL-DATE                                   1     00000    DISP     X(8)         71     78 81

01 DAY-OF-THE-WEEK                             2     00000    DISP     9(1)         74     78 110

01 DAY-ITSELF                                  3     00000    DISP     X(10)        76     78 111

   BEGIN                                                      PARA-NM               80     NOREF
   THE-END                                                    PARA-NM               112    NOREF
```

Figure 2-7. Sample Cross-Reference Listing (Declaration Order)

```
          COBOL     V-50.2        X93.1      LISTING  BOURGAIN BOURGAIN             13:42:36  MAR 31, 1978   PAGE      10
FIND-DAY            CROSS-REFERENCE LISTING (ALPHABETIC ORDER)
LN NAME                                        PN   ADDRESS    USAGE     PIC-STRING     DEF.   REF.  LINES

   BEGIN                                                       PARA-NM                  80     NOREF
02 CENTURY (SPLIT-DATE)                             1:000F0    DISP      9(2)           55     84
02 DAY-IN-THE-WEEK (DITWEEK-TAB-RED)                1:000A4    DISP      X(10)          51     111
01 DAY-ITSELF                                   3     00000    DISP      X(10)          76     78 111
03 DAY-OF-MONTH (SPLIT-DATE)                        1:000F6    DISP      9(2)           59     88
03 DAY-OF-MONTH-X (SPLIT-DATE)                      1:000F6    DISP      X(2)           60     82
01 DAY-OF-THE-WEEK                              2     00000    DISP      9(1)           74     78 110
01 DAYS-IN-THE-ERA                                  1:000F8    DISP      9(10)          65     87 96 98 100 105
01 DITWEEK-TAB                                      1:000A4    GROUP     X(70)          41     NOREF
01 DITWEEK-TAB-RED                                  1:000A4    GROUP     X(70)          50     NOREF
01 FULL-DATE                                    1     00000    DISP      X(8)           71     78 81
03 MONTHR(SPLIT-DATE)                               1:000F4    DISP      9(2)           58     89 94
01 OTHER-UNUSED                                     1:000A0    GROUP     X(4)           38     NOREF
01 PREC-D-TAB                                       1:00078    GROUP     X(36)          21     NOREF
01 PREC-D-TAB-RED                                   1:00078    GROUP     X(36)          34     NOREF
02 PRECEDING-DAYS (PREC-D-TAB-RED)                  1:00078    DISP      9(3)           35     89
02 SHORT-DATE (SPLIT-DATE)                          1:000F2    GROUP     X(6)           56     83
01 SPLIT-DATE                                       1:000F0    GROUP     X(8)           54     81 83
77 TALLY                                            1:00010    DISP      9(5)                  NOREF
   THE-END                                                     PARA-NM                  112    NOREF
01 X                                                1:00064    DISP      9(10)          16     95 96 97 98 99 100 105
01 Y                                                1:00070    DISP      9(5)           17     105 106 107 109 110 111
01 YEAR                                             1:000F0    DISP      9(4)           61     90 94+ 95 97 99
```

Figure 2-8. Sample Cross-Reference Listing (Alphabetic Order)

PROCEDURE MAP LISTING

The procedure map listing consists of a table of PROCEDURE DIVISION internal line numbers and the corresponding starting memory addresses for the generated object code. See Figure 2-9 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 an internal line number from a corresponding memory address. This is necessary when a user program terminates abnormally and a memory address is printed in the Job Occurrence Report.

A memory address is printed for each statement in the PROCEDURE DIVISION. Therefore there will be several memory address for the same internal line number if there is more than one statement on a source line or if the statement implies several simpler statements (e.g. MOVE CORRESPONDING) .

Internal line numbers will normally be in ascending 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, internal line numbers in the procedure map listing will not be in ascending order. In this case the complete procedure map listing is repeated in internal line number order. That is, two listings are produced, one in memory address order and one in internal line number 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.

```
         COBOL     V-50.2        X93.4      LISTING  BOURGAIN BOURGAIN              13:49:48  MAR 31, 1978    PAGE      11
CALENDAR                   PROCEDURE MAP LISTING
```

BEGIN	LINE	BEGIN	LINE	BEGIN	LINE	BEGIN	LINE	BEGIN	LINE	BEGIN	LINE
2:00024	130	2:0004E	131	2:00056	132	2:00160	135	2:0016C	135	2:00174	136
2:00194	137	2:00198	138	2:0020C	139	2:00240	142	2:0027C	144	2:00298	146
2:002A4	151	2:002B4	152	2:002E8	154	2:002F6	155	2:002FE	156	2:00324	157
2:00332	158	2:0033A	159	2:00364	162	2:00376	163	2:0037E	164	2:0038A	165
2:003B2	167	2:003BE	168	2:003CA	169	2:003D6	171	2:003DE	172	2:003EA	173
2:0041E	176	2:0042E	179	2:0043A	180	2:00470	181	2:004A4	183	2:004AE	184
2:004C2	189	2:004D2	193	2:00508	194	2:00514	195	2:00552	198	2:0055E	199
2:00568	200	2:00578	201	2:00596	202	2:005A0	203	2:005B0	204	2:005CE	205
2:005D8	206	2:005E8	207	2:00606	208	2:00610	209	2:00620	210	2:0063E	211
2:00678	214	2:00688	215	2:00694	216	2:006DA	218	2:00714	220	2:00724	221
2:00730	222	2:00746	223	2:0076E	224	2:00796	225	2:007AA	226	2:007BA	228
2:007C6	229	2:007D2	230	2:007E2	231	2:007F2	232	2:007FE	233	2:00808	233
2:00824	126	2:0083A	126	2:00840	126						

```
         COBOL     V-50.2        X93.4      LISTING  BOURGAIN BOURGAIN              13:49:48  MAR 31, 1978    PAGE      12
CALENDAR                   PERFORM/ALTER BUCKET LISTING
```

LINE	BEGIN	LINE	BEGIN	LINE	BEGIN	LINE	BEGIN	LINE	BEGIN	LINE	BEGIN
150	1:00362	161	1:00366	178	1:0036A	197	1:0036E	213	1:00372	219	1:00376

Figure 2-9. Sample Procedure Map Listing and Perform/Alter Bucket Listing.

For each such address, the internal line number of the relevant paragraph or SECTION is given before the address in ascending internal line number order.

See Figure 2-9 for an example.

Summary Page

A sample summary page is shown in Figure 2-10. The information contained on this page is discussed in the following paragraphs.

```
          COBOL     V-50.2        X93.1      LISTING  BOURGAIN BOURGAIN   13:42:36  MAR 31, 1978    PAGE
FIND-DAY            COMPILATION SUMMARY


SUMMARY OF ERRORS

      *              5         ON LINES A.7 40 S.44 83
    * *              3         ON LINES 83 110
  * * *              0
* * * *              0


CU PRODUCED ON LIBRARY :000093.TEMP.CULID


SEGMENT NAME         TYPE   SIZE (IN BYTES)

FIND-DAY.0           ..L          99
FIND-DAY.1           .D.         278
FIND-DAY.2           C..         434

STACK                             68


RUN TIME PACKAGE PROCEDURES INVOKED

NONE
```

Figure 2-10. Sample Summary Page

SUMMARY OF ERRORS

This shows the number of observation (*), warning(**), serious error (***) and fatal error (****) conditions detected by the compiler. The internal line number of the first 10 lines for which the compiler has output a message is shown for each type of message. When necessary the internal line number is prefixed by "A." or "S." to differentiate between the alter listing, source listing and expanded 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. Extra segments are generated when the DEBUG parameter is included in the $COBOL 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 Section III, Linking.

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 Section VII) 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.

Job Occurrence Report Summary

A summary of the compilation (message CBL02) is printed in the Job Occurrence Report. 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 *:2 **:4 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 Job Occurrence Report 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 out in the compilation listing, and an error message is written in the Job Occurrence Report with the following format:

```
CBL01. ERROR [AT address] WHILE COMPILING [LINE xxx OF] program-id
       [RETURN CODE IS rc FROM siuic (G4 = xxxxxxxx)] [ON file]
```

The diagnostic in the compilation listing specifies which kind of error was encountered. For example:

8-92 CULIB IS FULL.

8-93 I/O ERROR ON CULIB.

9-55 WORKn IS FULL.

9-45 UNRECOVERABLE DIFFICULTY DUE TO SYSTEM ERROR.

Another error message can be written in the Job Occurrence Report 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, or a PRTFILE sequential file or a PRTLIB library) is full.

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

SECTION III

LINKING

$LINKER is a Level 64 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.

Note: The system recognizes three forms of segment number during compilation, linking, program loading and execution. To avoid confusion in this and later sections 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 $COBOL data map, cross-reference and procedure map listings. Internal segment numbers are also included in the segment lists produced by the COBOL 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.

JOB CONTROL LANGUAGE

The extended JCL statement $LINKER is used to execute the $LINKER utility.

$LINKER generates a load module and a listing. The load module may, optionally, be stored in a temporary or a permanent library.

Figure 3-1 shows the format of the $LINKER statement.

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

[,ENTRY = compile-unit-name]

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

[,{COMMAND = 'linker-command [,linker-command]...'                     }]
[ {COMFILE = {*input-enclosure-name                                   }}]
[ {          {(library-file-description,SUBFILE=member-name)          }}]

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

Figure 3-1. $LINKER 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...
  LIBALLOC  LM, (LM.LIB, SIZE = 5), MEMBERS = 100;
  LIB CU    INLIB1 = CU.LIB;
  LINKER    PROG4,
            ENTRY = PROG1,
            OUTLIB = LM.LIB;
$ENDJOB;
```

The $LIBALLOC LM statement is used to create a library, LM.LIB, with a size of 5 cylinders (this utility need not be used if the library already exists). The $LIB 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 PROG1 (specified in ENTRY = PROG1). 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 PROG4.

The following paragraphs describe the parameters which may be used in the $LINKER statement.

Load-module-name Parameter

This parameter is used to specify the name of the load module created by $LINKER.

If there is no ENTRY parameter 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 from the $LINKER statement.

It is not possible to use the same name in this way if calling and called programs are used (see Section VI) because there will be several source programs and compile units for 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 load-module-name. This indicates that a series of load modules is to be linked during a single execution of $LINKER. See Serial Linkage, below.

ENTRY Parameter

This parameter specifies the (main) compile unit to be used as the starting point when building the load module. It can be omitted if the name of the main compile unit is the same as load-module-name.

When the ENTRY parameter is used the COMMAND parameter must be omitted.

OUTLIB Parameter

The OUTLIB parameter specifies the library in which the resulting load module is to be stored. A library-file-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 library-file-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 $LIBMAINT LM. Details of $LIBMAINT LM are given in the Library Management Manual.

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.

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. Several different commands may be specified, but the only ones of interest to the COBOL programmer are the ENTRY, INCLUDE and VACSEG commands. These commands are described in the following paragraphs. The COMMAND and COMFILE parameters may also be used to specify a series of load modules to be linked during a single execution of $LINKER. See Serial Linkage, below.

ENTRY COMMAND

The format of the ENTRY command is:

```
ENTRY = member-name
```

This command has exactly the same function as the ENTRY parameter. When the COMMAND parameter is used in the $LINKER statement the ENTRY parameter cannot be used. The ENTRY command should be used instead.

INCLUDE COMMAND

The format of the INCLUDE command is as follows:

```
            { member-name[,member-name]... }
INCLUDE =   {           INLIBn             }
```

This command is used to specify a list of compile units which are to be included in the load module as if they had been referred to by another compile unit. If the INLIBn option is used then all compile units of the library specified by INLIBn are included in the load module. INLIBn refers to the library specified in the corresponding INLIBn parameter in the $LIB statement preceding the $LINKER statement. "n" may have a value from 1 to 4.

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.

VACSEG COMMAND

The format of the VACSEG command is as follows:

```
VACSEG = (SHARE = +a)
```

This command must be used if "multi logical unit files" are to be used in the COBOL program. See Section IX for an explanation of how to calculate the value of "a".

STEPOPT Parameter

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

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

Library Search Path

A $LIB CU statement may precede the $LINKER statement to define a search path for the compile units to be linked. Up to four compile unit libraries can be specified in the search path. If no $LIB CU statement is active, $LINKER will search the TEMP compile unit library.

User compile units may be created in the same job as the $LINKER execution. If so, compile units may be in the system compile unit library specified by TEMP in the $COBOL CULIB parameter. In such a case TEMP must be included in the search path of $LINKER (unless all compile units involved in the linking are in TEMP). For example:

```
LIB CU, INLIB1 = USER.LIB,
        INLIB2 = TEMP;
```

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. The COMMAND or COMFILE parameter is used to specify a set of parameters for each load module to be linked.

For 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 there must be a comma after each parameter including the final parameter for each load module (i.e. immediately before the semi-colon). Also, when the COMMAND parameter is used, the parameters for each load module except the last must be followed by a hyphen (i.e. the semi-colon must be followed by -).

OPERATION OF $LINKER

The way in which $LINKER builds a load module will be shown by discussing a particular example.

Suppose that a COBOL program comprises a main program, MAINPAY, which calls a program EDITION, which in turn calls a program GETDATE (see Section VI, Calling and Called Programs). The relationships between these programs and the run-time package and the system routines might be as shown in Figure 3-2.

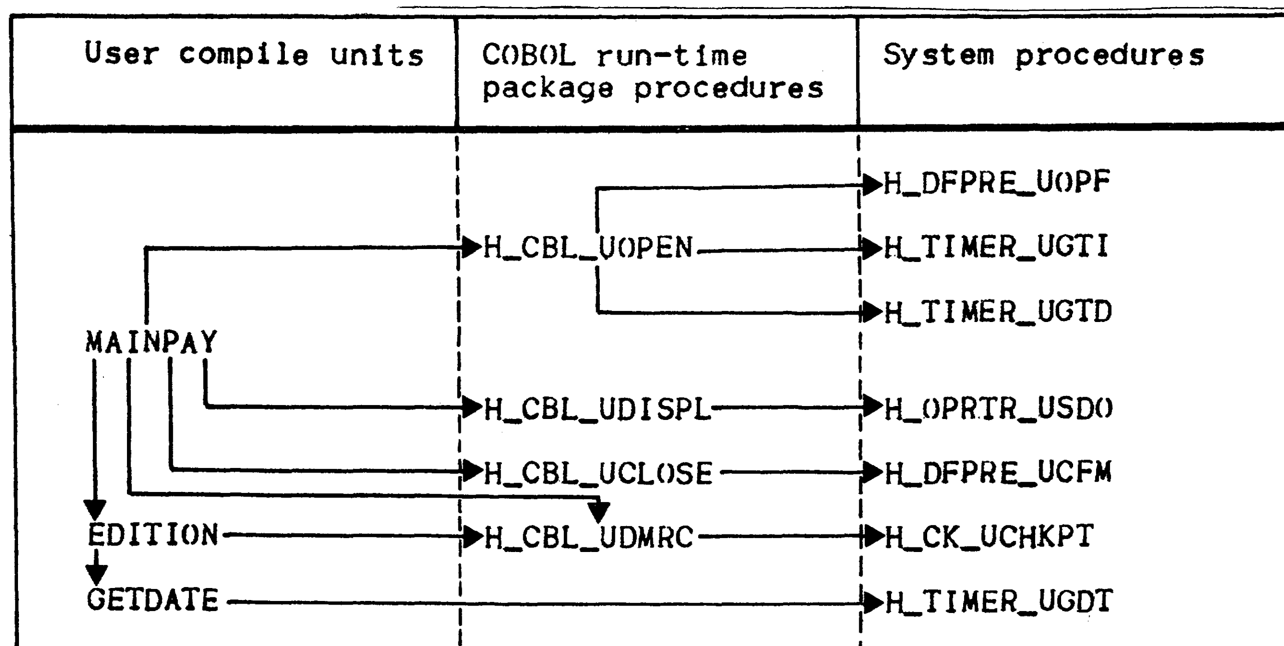


Figure 3-2. Structure of a linked program

The procedures named in this linkage are of three origins:

- User compile units resulting from COBOL compilations;
- Run-Time Package: a group of procedures used by COBOL programs: they are available in memory in a single copy and are not built into the load module;
- System procedures: the parts of the system handling user I/O requests etc: they are not built into the load module.

$LINKER, starting with the main compile unit (MAINPAY) scans the object code for all unresolved external references. After incorporating each compile unit that resolves such a reference $LINKER descends to the next level in the hierarchy and resolves the references made in the incorporated compile units. This procedure continues until all external references which can be resolved are resolved. References to the COBOL run-time package and some system procedures are not resolved until run-time.

PRINTED OUTPUT

The following paragraphs describe the printed output produced by $LINKER. The output is described in the order in which it is produced under the following headings:

- Banner page and $LINKER commands listing.
- Included compile units (if any).
- Task listing.
- Group information.
- Linkage report and end page.

Banner Page and $LINKER Commands Listing

An example banner page and an example $LINKER commands listing are shown together in Figure 3-3. All commands included in the COMMAND parameter of the $LINKER statement are listed in the $LINKER commands listing.

Included Compile Units

Details are printed for each compile unit included in the load module as a result of using the INCLUDE command. The format of this listing is similar to the Task Listing (see Figure 3-4.)

The heading for each compile unit shows the compile unit name, the library from which the compile unit was taken, the date and time at which the compile unit was created and the name (i.e. language) and version of the compiler which generated the compile unit.

For each compile unit a list of symbolic references (SYMREFS) is printed. A symbolic reference is a reference to an entry point or data item in another compile unit. Such a reference remains in symbolic form (i.e. in the form of a label) because it cannot be converted into an address at compile time. The following information is printed for each symbolic reference:

- SYMREFS — The labels which are being referenced.
- TYPE — The type of reference. DATA indicates a reference to a data item. PROC indicates a reference to a compile unit entry point.
- LOCATION — The location at which the symbolic reference was made. This is in the form stn.ste.sra. where stn is the segment table number, ste is the segment table entry and sra is the address relative to the start of the segment.
- MATCH.DEF.IN — The compile unit in which the referenced label was found by $LINKER and the address of this label in the form stn.ste.sra. If the referenced label could not be found in any compile unit in the library search path (defined by $LIB CU), the comment **NOLINK** is printed instead of the compile unit and address.

Symbolic references which begin H_ are not listed unless there is an error report for that reference.

Task Listing

There is a task listing for each task in the load module. An example task listing is shown in Figure 3-4.

A load module will contain more than one task only if it contains two or more sequences of program code which may be executed asynchronously. This type of processing is not possible in load modules written entirely in COBOL (unless the Message Control System is being used). For this reason there will normally only be one task listing for a COBOL program. This listing starts with the heading TASK = MAIN.

The first item in the task listing concerns the task entry-point (the point at which execution begins). The name of the entry point and the name of the containing compile unit are printed. For COBOL programs these names are identical. The location of the entry-point is shown in the form stn.ste.sra.

The remainder of the task listing consists of a list of all the compile units in the task together with details of the symbolic references in each compile unit. This part of the task listing has a format similar to the list of included compile units described above. Compile units which are listed in the list of included compile units are not repeated in the task listing. Symbolic references which begin H_ are not listed unless there is an error report for that reference.

```
           LINKER      50.10       X121.2     NC214     MAVIER    COBL                   15:07:10  MAY 03, 1978    PAGE      20



*******************************************************************************************************************************
*******************************************************************************************************************************
**** GCOS L64                                                                                                              ****
****                                 L I N K E R                                                                           ****
****                                                                            VERSION: 50.10  DATED: DEC 01,1977         ****
*****NC214_K**************************************************************************** 10************************************
*******************************************************************************************************************************



           LINKER      50.10       X121.2     NC214     MAVIER    COBL                   15:07:10  MAY 03, 1978    PAGE      21
LM : NC214

* * * * * * * * * * * * * * * * * * * * * * LINKER CONTROL STATEMENTS * * * * * * * * * * * * * * * * * * * * * * * * *

    ENTRY=NC214,
```

Figure 3-3. Sample Banner Page and $LINKER Commands Listing

```
         LINKER      50.10      X93.5     LISTING  BOURGAIN BOURGAIN                13:50:17  MAR 31, 1978   PAGE      2
LM : CALENDAR

* * * * * * * * * * * * * * * * * * * * * * * *TASK=MAIN* * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * *

    ENTRY POINT = CALENDAR         LOCATION: 08.0C.C00008     IN CU: CALENDAR

   0.CU= CALENDAR           FROM:INLIB1 CREATED  ON  3/31/78 AT 13:49:48 BY:  COBOL 50.2

    -SYMREFS                    TYPE   LOCATION     MATCH. DEF IN
    1.PRT                       DATA   08.0C.00C044 **__BLANK**/08.10.000000
    1.H_CBL_UOPEN               PROC   08.0C.00C070 **NOLINK**
**** WARNING 2604               UNRESOLVED REFERENCE
    1.H_CBL_UCLOSE              PROC   08.0C.00C078 **NOLINK**
**** WARNING 2604               UNRESOLVED REFERENCE
    1.H_CBL_USTOP               PROC   08.0C.00C080 **NOLINK**
**** WARNING 2604               UNRESOLVED REFERENCE
    1.FIND-DAY                  PROC   08.0C.000092 FIND-DAY/08.11.000008

       1.CU= FIND-DAY             FROM:INLIB1 CREATED  ON  3/31/78 AT 13:47:24 BY:  COBOL 50.2

    (CALENDAR - CONT'D)
    -SYMREFS                    TYPE   LOCATION     MATCH. DEF IN
    1.H_CBL_UDMRC               PROC   08.0C.00C0C2 **NOLINK**
**** WARNING 2604               UNRESOLVED REFERENCE
```

Figure 3-4. Sample Task Listing

Group Information

A sample group information listing is shown in Figure 3-5. This listing contains information concerning the entire process group (i.e. the entire load module). The listing is in two parts:

- Global segment list.

- Segment list.

Global segments are data segments which can be referenced from more than one procedure segment. They contain records which have been declared EXTERNAL in the COBOL source program (see Section VI). The global segments listing contains the segment name and the $LINKER segment number (stn.ste) for each global segment. Also listed are the data-names and internal-file-names within each global segment and their corresponding segment relative addresses (sra).

The segment list, on the other hand, contains an entry for each segment in the load module (including global segments but excluding segments with a name beginning H_). 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 Section IV).

- The size of each segment in bytes is shown. This may be useful when segmenting a COBOL program (see Section VII) and when estimating working set requirements for program execution.

The headings and information in the segment list 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 $COBOL).

SH This indicates the shareability of the segment. It can have the following value:

1 The segment can be shared between certain process groups in the system.

2 The segment can be shared between all the processes of the process group.

3 The segment is private to a process.

RF This indicates the residence factor of the segment.

RD,WR,EX These indicate the minimum protection ring values which other segments must have in order to read from, write to or execute the current segment.

WP,EP W or E under these headings indicates that the segment may be written to (modified) or executed.

G,S G or S under these headings indicates that the segment is a gate or semaphore segment .

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. The calculation of working sets is described in the System Management Guide.

CONT.P. Names processes which "contain" the segment. An asterisk (*) under this heading indicates that all processes in the process group have access to the segment. Since most COBOL programs consist of only one process, an asterisk will normally be found under CONT.P.

Linkage Report and End Page

An example linkage report and an example end page are shown together in Figure 3-6.

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.

```
          LINKER      50.10      X93.5      LISTING  BOURGAIN BOURGAIN              13:50:17  MAR 31, 1978   PAGE      3
LM : CALENDAR

* * * * * * * * * * * * * * * * * * GROUP  INFORMATION * * * * * * * * * * * * * * * * * * * * * * * * * * * * * *

                                          GLOBAL  SEGMENTS

    SEGNAME               SEG NUM  CONTAINS:
    H_U_BIFN              09.0B                                LOCATION                                    LOCATION
                                   H_CBL_DRTP(2                  000000   H_CBL_D_CSP                        000034
    H_CBL_DRTP            08.0F                                LOCATION                                    LOCATION
                                   H_CBL_DRTP(1                  000000
    __BLANK               08.10                                LOCATION                                    LOCATION
                                   PRT                           000000
```

SEGMENT LIST

SEG.#	IN CU:	TYPE	SH	RF	RD	WR	EX	WP	EP	G	S	SIZE	MAXSIZE	CONT.P.
08.0C	CALENDAR.0	..L	3	3	3	3	3					208		*
08.0D	CALENDAR.1	.D.	3	3	3	3	3	W				1408		*
08.0E	CALENDAR.2	C..	3	3	3	3	3		E			2256		*
08.0F	H_CBL_DRTP	.D.	3	3	3	3	3	W				608	8192	*
08.10	__BLANK	.D.	3	3	3	3	3	W				16		*
08.11	FIND-DAY.0	..L	3	3	3	3	3					112		*
08.12	FIND-DAY.1	.D.	3	3	3	3	3	W				288		*
08.13	FIND-DAY.2	C..	3	3	3	3	3		E			480		*
09.00	PGCS	CD.	2	3	3	0	3	W	E			1232		
09.04	TERMINATION	.D.	2	3	3	0	0	W			S	96		
09.0B	H_U_BIFN	.D.	2	3	3	3	3	W				320		
09.0E	SEMPH. POOL	.D.	2	3	3	1	1	W			S	464		

Figure 3-5. Sample Group Information Listing

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.

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. A complete list of $LINKER error messages is given in Appendix C.

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.

NOTE: When building a load module from COBOL compile units $LINKER will almost always output several error messages with a message number 2604. If these messages refer to symbolic references beginning H_ they can be ignored by the user. These messages simply mean that references to COBOL run-time package procedures and certain system procedures have not been resolved. These references will be resolved at execution-time.

```
        LINKER      50.10       X93.5      LISTING  BOURGAIN BOURGAIN                  13:50:17  MAR 31, 1978   PAGE      4
LM : CALENDAR

* * * * * * * * * * * * * * * * * * * * * * * * LINKAGE  REPORT * * * * * * * * * * * * * * * * * * * * * * * * * * *

                  ERRORS DETECTED (*)
                  -------------------------
                    - WARNINGS (SEV.1):    4

-OUTPUT MODULE PRODUCED ON LIBRARY     :00003.TEMP.LMLIB



(*) WARNINGS AND ERRORS ARE INTERSPERSED WITH LISTING,AT THE PLACE WHERE THEY OCCUR.
                  THEY ARE MARKED WITH "****" IN THE LEFTHAND MARGIN.
    CONFLICTING ITEMS ARE MARKED WITH "--->" IN THE LEFTHAND MARGIN.

* * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * *
* * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * *
* * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * *
```

```
        LINKER      50.10       X93.5      LISTING  BOURGAIN BOURGAIN                  13:50:17  MAR 31, 1978   PAGE      5

* * * * * * * * * * * * * * * * * * * * * * * * *  END OF SESSION * * * * * * * * * * * * * * * * * * * * * * * * * LAST
PERCENTAGE OF SPACE USED

* * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * *
* * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * *
* * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * *
* * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * *
* * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * *
* * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * *
```

Figure 3-6. Sample Linkage Report and End Page

SECTION IV

EXECUTION

This section introduces the Level 64 debugging facilities and 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 Section III, Linking.

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.

Debugging Code

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

- One or more debugging SECTIONs in the PROCEDURE DIVISION DECLARATIVE. 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 $COBOL 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 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" column 7).

When a load module consists of more than one COBOL program and the DEBUG parameter in used in the $STEP 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) command CHANGE (C):

```
C,stn.ste.30 = "0000a"X;
```

A full explanation of the CHANGE command can be found in the Program Checkout Facility Manual. "stn.ste" gives the segment table number and the segment table entry corresponding to 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 Section III 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 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 statement then, unless commands specify otherwise, all USE FOR DEBUGGING SECTIONs are activated in all programs when execution starts.

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 Manual 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 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 $COBOL 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 (symbolic addressing). However, the size of these tables should be born in mind (about 60 bytes per source line). A segment containing these tables is generated for each 200 lines of source code (approximately).

The DEBUG parameter in the $STEP 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.

DUMP ANALYSIS

A memory dump of the user program can be obtained if the DUMP parameter is included in the $STEP 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.

Structure of the Dump Listing

The dump listing is divided into two parts. The first part contains the segments shared by all processes of the process group. The first segment in this part is the Process Group Control Structure for the step. It is preceded by the following heading:

```
************
****PGCS****
************
```

See Figure 4-1 for an example of the first page of a dump. 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). The segments in first part of 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.

The second part of the dump listing contains the segments which are private to the process. The first segment in this part is the Process Control Structure for the process. It is preceded by the following heading:

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

See Figure 4-2 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******

The remainder of the second part of the dump listing contains the data segments (and code and linkage segments if DUMP = ALL was specified) which make up the COBOL program proper.

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. 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=03 STE=00 SEGDESCR 9C00B935 FE0C0C7F
```

The ring 3 stack is the one relevant to the user program. See Figure 4-3 for an example of such a stack.

```
*****************************************************************************************
*****************************************************************************************
**** GCOS L64                                                                        ****
****                                 D U M P                                         ****
****                                          VERSION: JB_30  DATED: MAY 11, 1978    ****
*****************************************************************************************
*****************************************************************************************

(C) 1976, CII-HONEYWELL BULL

DUMP  J=64   P=06     TERM.MSG 20820405

************
****PGCS****
************

/J=64/P=00/  /STN=69/ STF=00/  SEGMENT  DESCRPT:  9C002478 CF0000F1

SEGM.HEADR:  00024310 000256A0 09000400 00024950  00006A09 00000000 000000F1 00000000

0000 024760  0CF10F20 01002478 0054005A 00690190  07200740 00000100 0000001A 00000000
0020 0247A0  0900901C0 FFFF0FF0 00000800 00026670  09040008 FFFFFFFF 09020000 84308000
0040 0247C0  00000000 7FFFFFFF FF040102 EF320150  80002489 69912900 03000000 00000000
0060 0247E0  00000200 182F0334 00000000 00009022  00000900 0EF441F0 00000900 00000000
0080 024800  C0011E05 000E0080 005C0090 0063007A  00101000 82A04005 83004005 00104000
00A0 024820  09040010 03000000 0902002C C0090003  1A000000 00000000 7FFED0D5 FFFFFFFF
00C0 024840  FFFFFFFF FFFFFFFF C410FF00 01F00130  FF000210 3580FF00 02200000 00000000
00E0 024860  00000000 00000000 0E0051FC 7A090000  00000010 07A20000 00000079 500C0000
0100 024880  0000064A ED308000 00000000 0A739000  30F06000 00000000 49040040 07000000
0120 0248A0  07002493 03002425 0C6E0010 01490028  0A210486 00000000 01400000 01800000
0140 0248C0  01000000 01400074 01490030 0013002A  001F0000 02000000 0A490000 0201109C
0160 0248E0  0A0D0578 000000F0 00070000 00000020  1226J0A8 00000000 0000A000 0000000C
0180 024900  00000039 00000073 0000002C 00000000  00000000 122600A8 00000000 00000001
01A0 024920  12250004 00000000 00000000 00010000  01000239 04002427 01000280 01002499
01C0 024940  00000000 00000000 00000000 00010000  35002A9A 69002485 FF000298 FF00031B
01E0 024960  FF000398 86000418 00000000 00010000  03000000 00000000 9C00A548 020000BF
0200 024980  24-03A622 55000075 10000000 11001075  9C0000FD FE00007F 00000000 00000000
0220 0249A0  9C002475 C2000014 9C00242C 62010001  03000000 00000000 10178601 42040000
```

Figure 4-1. First Page of Dump

```
0060 024300  00000000 00000000 00000030 00000000                                    @@@@@@@@@@@@@@@@

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

/J=04/P=00/  /STN=08/ STE=00/  SEGMENT  DESCRPT:  9C00247E C2000014

        DUMP       JG_30       X7.4      CREACTDB MIGHILL  COBL              10:53:07  MAY 11, 1978   PAGE     29

0000 0247E0  00000200 182F0334 00000000 00000022  00000900 0EF441F0 00000000 00000000  @@@@@@@@@@@@@@@@@@@@@@4@0@@@@@@@@
0020 024800  0001F4CE 00880080 005C0000 0063007A  00101000 82A04005 83004005 00104000  @@4@@@@@@*@@@@@:@@@@@@ @@@ @@@ @
0040 024820  09040010 03000000 0202002C C0000003  1A000000 00000000 7FFEB6D5 FFFFFFFF  @@@@@@@@@@@@@@@@@@@@@@@@@"@@N@@@
0060 024840  FFFFFFFF FFFFFFFF 0410FF00 01F00130  FF000210 3580FF00 02200000 00000000  @@@@@@@@@@@@0@@@@@@@@@@@@@@@@@@
0080 024860  00000000 00000000 00009204 00930000  00000011 86270000 0000007A 484F0000  @@@@@@@@@@@R@@@@@@@@@@@@@@@:@|@@
00A0 024880  00000051 3D7E8000 00000000 165A0000  80E06000 D0000000 490E0210 07000000  @@@@@=@@@@@@@!@@@@-@@@@@@@@@@@@@
00C0 0248A0  07002493 03002495 0C6E0010 032F0CFD  1AD80796 00000000 01400000 01800000  @@@@@@@@>@@@@@@@Q@@@@@@@ @@@@@@
00E0 0248C0  01C00000 182F061C 082F0634 0D270078  182F02C3 080000E0 1BC8043E 182F061C  @@@@@@@@@@@@@@@@@@@C@@@@@@@@@@@@
0100 0248E0  0ABC0010 082F0634 00000000 182F056A  182F056E 182F0568 0000006A 00000001  @@@@@@@@@@@@@@@@@@@>@@@@@@@@@@@@
0120 024900  00000000 00000004 0000004C 00000008  0AA70008 208204C5 3300016C FFFFFFF8  @@@@@@@@@@@<@@@@@@@@@@E@@@%@@@8
0140 024920  00000000 00000000 00000000 00000000                                    @@@@@@@@@@@@@@@@

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

           PMW0: 80E06000     PMW1: 10000000      PMW2: 490E0210     PMW3: 07000000

           ASW0: 07002493     ASW1: 03002495       EXW: 0C6E0010      SKW: 082F0634

            ICW: 1ABC0068     SBW0: 01400000      SBW1: 01800000     SBW2: 01C00000

            BR0: 182F061C      BR1: 032F0634       BR2: 0D270078      BR3: 182F02C3

            BR4: 080000E0      BR5: 1BC8043E       BR6: 182F061C      BR7: 0ABC0010

            GR0: 082F0634      GR1: 00000000       GR2: 182F056A      GR3: 182F056E

            GR4: 182F0568      GR5: 0000006A       GR6: 00000001      GR7: 00000000

            XR0: 00000004      XR1: 0000004C       XR2: 00000008      XR3: 0AA70008

            XR4: 208204C5      XR5: 3300016C       XR6: FFFFFFF8      XR7: 00000000

RING 0 STACK STN=01   STE=01    SEGDESCR 9C001548  020000BF
 *STACK FRAME 001
  *WRKAREA
     0130     00E00400 0F400070 0140                                                 @@@@@ @@@
  *SAVAREA
     013C         SAM= 0FFFFFF0
     0140     GR0-7: 01400110 01400130 00000000 0BAF003C 00350000 0BE40048 01400110 0C170010
     0160     GR5-7: 00000030 04100400 00000000 0AA70008 00000001 00000000 00000001 00000000
     0180     XR0-7: 00024750 00024320 00000008 0AA70008 00000000 00000004 FFFFFFF8 00000000
     01A0       STR: 00404040
```

Figure 4-2. Start of PCS Dump

```
         DUMP        JB_30       X7.4       CREACTDB MIGHILL  COBL                         10:53:07  MAY 11, 1978   PAGE      30

   00A8         SAM= 0FFF7FF0
   00AC      BR0-7: 01400070 01400090 08000000 02000000 0D350000 0BE4004B 01400070 0A880010
   00CC      GR0-7: 00000000 00000000 3300016C 00000000 ******** 00000000 00000001 00000000
   00E8      XR0-7: 000247E0 00024890 00000008 38180D26 000002F1 00000004 FFFFFFF8 00000000
   0108        STR: 00000000
   010C        PTV: 014000A8
   0110        PSA: 014000A8
   0114        ICC: 0A88040A
 *COMAREA
   0118         NBP= 00000014
   011C    01400096 0A880024 0A880028 0A830077  0A880077
*STACK FRAME 003
 *WRKAREA
 *PAGE=082F0000
   0000    208204C5 3300016C                                                                     @@@E@@@%
 *SAVAREA
   0008         SAM= 0FFF7FF0
   000C      BR0-7: 3300016C 01400000 08000000 08180D26 03000100 03000000 0BE3020A 0BE40010
   002C      GR0-7: 208204C5 00000C02 0000000C 00000C02 ******** 00000000 FFFFFFFD 00000400
   0048      XR0-7: 38180D26 04000C02 03000000 38180D26 000002F1 00000004 FFFFFFF8 00000002
   0068        STR: 80000000
   006C        PTV: 01400008
   0070        PSA: 01400008
   0074        ICC: 0BE4046A
 *COMAREA
   0078         NBP= 00000014
   007C    0BE40047 01400000 0BE40045 01400004  0BE4004B

RING 3 STACK STN=03   STE=00   SEGDESCR 9C000DFD  FE00007F
*STACK FRAME 001
 *WRKAREA
   009C    0FFFFFF0 33000088 0300009C 080D0000  08120000 FFFFFFFF 08000000 33000088   @@@0@@@@@@@@@@@@@@@@@@@@@@@@@@@@
   00BC    080C0010 03000000 0000000E 00000020  09080000 7FFFFD55 00000000 00000000   @@@@@@@@@@@@@@@@@@@@"@@@@@@@@@@@
   00DC    0000000A 00000000 00000000 00000000  00000000 00000000 00000000 00000000   @@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@
   00FC    00000000                                                                   @@@@
 *SAVAREA
   0100         SAM= 0FFFFFF0
   0104      BR0-7: 33000088 0300009C 080D0000 08120166 08120000 08120000 081609BA 080C0010
   0124      GR0-7: 00000000 00000011 00000012 59041000 82051820 020CC606 FFFFF3F0 FFFFFFFF
   0144      XR0-7: 00003710 00000299 00E00000 00001510 00000008 0000005B 00000006 00000002
   0164        STR: 7E000000
   0168        PTV: 03000100
   016C        PSA: 03000100
   0170        ICC: 38180D26
 *COMAREA
   0174         NBP= 00000010
   0178    000002F1 04000C02 03000000 04F11819
*STACK FRAME 002
 *WRKAREA
 *PAGE=33000000
   0000    00000000 FFFFFFFF 38180D26 04000C02  03000000 FFFFFFFF 00080C00 08000000   @@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@
 *SAVAREA
   0020         SAM= 0FFF7FF0
   0024      BR0-7: 33000000 03000000 080C0008 03000000 FFFFFFFF 08000000 00000000 0C220008
   0044      GR0-7: 03000000 0000000E 00000020 09080000 ******** 00000000 00000000 0000000A
```

Figure 4-3. Ring 3 User Stack

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 Figure 4-4 for a sample stack frame 001.

```
*STACK FRAME 001
 *WRKAREA
 *PAGE=33000000
    0000     00000000 FFFFFFFF 38170130 02000000 00000000 FFFFFFFF
 *SAVAREA
    0020      SAM=     0FFF7FF0
    0024     BRO-7:    33000000 03000000 08000008 03000000 FFFFFFFF
    0044     GRO-7:    03000000 00000000 00000020 09080000 ********
    0060     XRO-7:    00000000 00000000 00000000 00000000 00000000
    0080      STR:     3E000000
    0084      PTV:     03000020
    0088      PSA:     03000020
    008C      ICC:     380E009E
 *COMAREA
   0090       NBP=     00000010
   0094    000002F1    02000000 00000000 04F10000
```

Figure 4-4. 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:

```
008C      ICC:     380E009E
```

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 Job Occurrence Report (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 8.0E. From the $LINKER segment list in Figure 4-5 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 COBOLTEST.

SEGMENT LIST

SEG.#	IN CU:	TYPE	SH	RF	RD	WR	EX	WP	EP	G	S	SIZE
08.0C	COBOLTEST.0	..L	3	3	3	3	3					80
08.0D	COBOLTEST.1	.D.	3	3	3	3	3	W				880
08.0E	COBOLTEST.4	C..	3	3	3	3	3		E			576
08.0F	__BLANK	.D.	3	3	3	3	3	W				144
09.00	PGCS	CD.	2	3	3	0	3	W	E			1120
09.04	TERMINATION	.D.	2	3	3	0	0	W			S	96
09.0B	COBOLTEST.2	.D.	2	3	3	3	3	W				4000
09.0C	COBOLTEST.3	.D.	2	3	3	3	3	W				176
09.0E	SEMPH. POOL	.D.	2	3	3	1	1	W			S	192

Figure 4-5. Sample $LINKER Segment List

The procedure map listing produced by $COBOL for the indicated compile unit (in this example COBOLTEST) should be consulted in order to find the internal 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 8.0E in the above example indicates internal segment number 4 of compile unit COBOLTEST. The internal segment number (4) should be concatenated with the sra (009E) from the instruction counter. Thus the address to look for in the procedure map listing is 4:009E. 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. See Figure 4-4. 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 fo find the data-name.

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 listing is needed which includes at least one of the following (see Section II for the associated $COBOL 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	USAGE	PIC-STRING	DEF.
02	A-KRBAB (KR-KRBAHEADER)		1:000EC	DISP	9(1)	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 in Figure 4-5. From this sample it can be seen that the $LINKER segment number (SEG.#) corresponding to internal segment number 1 is 8.0D. 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 8.0D the stn is 8 and the ste is D. 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 = 0D is shown in Figure 4-6. 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 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).

General Information

The following information may also be of interest:

- the segment whose name is program-name.1 (usually pointed to by BR2) contains:
  a) at offset 18 (hexadecimal) the program-name.
  b) at offset 36 (hexadecimal) the version of the compiler used to compile the program.
  c) at offset 4F (hexadecimal) the date of compilation.
  d) at offset 57 (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 the dump listing.

- 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 backet listing" be printed by the compiler (MAP parameter in $COBOL). This listing is explained in Section II.

```
/J=04/E=C0/ /STN=08/ STE=0D/  SEGMENT  DESCRPT:  9C00739A FE000100

SEGM.HEADR:  00073AA0 00074980 C0080410 0002A403  01132841 01178301 00000101 00000000

0000 0739A0  F5000010 00000303 40130000 030C0010  FF000032 080C0A04 E360C3C2 D360C3E3  0@@@@C@@@@@@@@@@@@@@@@@T-CRL-CT
0020 0739C0  C5E2F340 40404040 40404040 40404040  40404040 4040D3F6 F440C3D6 C2D6D340  EST                   L64 COBOL
0040 0739E0  E560F5F0 4BF2F640 40404040 404040F0  F561F0F3 61F7F8F1 F57AF2F0 7AF2F4F6  V-50.26         05/03/7815:20:246
0060 073A00  F4D10F0D C0628800 00663817 0AF60003  08100010 08180018 08180020 08180028  4J@@@@@@@@@@W@@@@@@@@@@@@@@@@@@@
0080 073A20  08180030 08180038 08133914 00034000  0002330D 00CF0058 0818008C 0001380D  @@@@@@@@@@@@@@@ @@@@@H@@@@@@@@@@
00A0 073A40  00A200AA 081300F6 08130DF4 0813015F  0813018A 081301AE 081301D2 081301F6  @@@@@@@W@@OU@@@;@@@@@@@@@@@@K@@6
00C0 073A60  0813021A 0813023E 00F60000 0000E240  40404040 40404040 40404040 40404040  @@@@@@@@@@@@@@@S
00E0 073A80  40404040 40404040 40404040 40404040  40404040 40404040 40404040 40404040
========
0140 073AE0  40404040 40404040 40404040 40404040  40404040 080D0154 080D0158 0816049A                 @@@@@@@@@@@@
0160 073B00  08160500 08160548 08160596 081605BC  081605DA 081605F8 08160616 090C0042  @@@@@@@@@@@@@@@@@@@@@@8@@@@@@@@@
0180 073B20  081606CA 08160766 08160802 0816083F  03000194 0816098A 081609F4 08160A56  @@@@@@@@@@@@@@@@@@@@@@@@@@@4@@@@
01A0 073B40  08160AC0 08160B2A 08160B7A 08160C1F  08000180 08000184 08160D7A 08160DDC  @@@@@@@@@@@:@@@@@@@@@@@@@@@:@@@@
01C0 073B60  08160EA0 0817007E 081700F0 08170146  08170212 081702AC 090C0084 0817038C  @@@@@@@=@@@0@@@@@@@@@@@@@@@@@@@@
01E0 073B80  08170430 081704D4 08170536 030001F0  081705FE 08170692 08170736 08170798  @@@@@@M@@@@@@0@@@@@@@@@@@@@@@@@@
0200 073BA0  081707FA 08170850 081708D8 0817093F  0817097E 08170A32 08170A52 08170A7A  @@@@@@&@@@Q@@@@@@=@@@@@@@@@@@@:
0220 073BC0  08170AA6 08170AD2 08170BDA 08170CF4  08170CFE 08170E1A 08180112 081805FE  @@@@@@K@@@@@@@4@@@@@@@@@@@@@@@@@
0240 073BE0  081809FA 08180DD8 08190310 0819033C  08190474 08190480 08190864 081908E8  @@@@@@G@@@@@@@@@@@@@@@@@@@@@@@@Y
0260 073C00  08190D42 08190D8A 08190DD2 08190E1A  08190E62 C3D6D4D7 C9D3D9C5 E2E4D3E3  @@@@@@@@@@@K@@@@@@@@COMPILRESULT
0280 073C20  D9C5E2E4 D3E3F1D9 C5E2E4D3 E3F2D9C5  E2E4D3E3 F3E3C5E2 E3C4C2D2 C5E87EC3  RESULT1RESULT2RESULT3TESTDBKEY=C
02A0 073C40  C1D9C4C6 C9D3C5C1 C2E2E3D9 C1C3E3E6  D6D9D2D7 00780001 080D02B8 080D02BC  ARDFILEABSTRACTWORKP@@@@@@@@@@@@
02C0 073C60  D7E660E6 D2D6E400 081C0010 C1D7F0F0  F9F50000 001C0B1F 0008C3D6 D4D7C9D3  PY-WKOU@@@@@AP0095@@@@@@@@COMPIL
02E0 073C80  0000C0C6 D540C1C2 D6D9E3F2 F9C3C1D5  D5D6E340 080D02F8 40E260C8 C5C1C4C5  @@@FN ABORT29CANNOT @@@8 S-HEADE
0300 073CA0  D9F0F0C3 C1D5D5D6 E340D9C5 C1C440C3  60C8C5C1 C4C5D9C8 C5C1C4C5 D940D5D6  ROOCANNOT READ C-HEADERHEADER NO
0320 073CC0  E340C6D6 E4D5C4C3 C1D5D5D6 E340C3D9  C5C1E3C5 40C8C5C1 C4C5D9C8 C5C1C4C5  T FOUNDCANNOT CREATE HEADERHEADE
0340 073CE0  D940C3D9 C5C1E3C5 C4000000 1E000000  1C000100 1D000014 0008E3C5 E2E340D5  R CREATED@@@@@@@@@@@@@@@@@@TEST N
0360 073D00  D6E340C1 C3E3C9C6 40C9D540 C4C1E3C1  60C2C1E2 C5D5D640 D7C5D9D4 C1D5C5D5  OT ACTIF IN DATA-BASENO PERMANEN
0380 073D20  E340D9C5 E2E4D3E3 E240E2E3 D6D9C5C4  0001001D C3D6D4D7 C1D9C9E2 D6D540E2  T RESULTS STORED@@@@COMPARISON S
03A0 073D40  E4C3C3C5 E2E2C6E4 D3000000 1EC5E7C5  C3E4E3C9 D6D540C1 C2D6080D 03BA080D  UCCESSFUL@@@@EXECUTION ABO@@@@@@
03C0 073D60  03BED5D5 D6E340C3 C1D5C3C5 D340D6E4  E3D7E4E3 C4000000 08220008 C3D6D4D7  @@NNOT CANCEL OUTPUTD@@@@@@@COMP
03E0 073D80  C9D3090C 0108D7C1 D9C540E2 E4C3C3C5  E2E2C6E4 D3D9080D 03FAE340 C3D6D4D7  IL@@@@PARE SUCCESSFULR@@@@T COMP
0400 073DA0  C1D9C540 E2E4C3C3 C5E2E2C6 E4D30000  001EC5E7 C5C3E4E3 C9D6D540 C1C2D6D9  ARE SUCCESSFUL@@@@EXECUTION ABOR
0420 073DC0  E3E5C9C4 C1C7C5F0 F0F1F05C D3C9E2E3  C1D3D300 00001C00 08250008 08260008  TVIDAGE0010*LISTALL@@@@@@@@@@@@@
0440 073DE0  E2E3D6D9 C5C440C3 D6D4D7C9 D340C4C1  E3C140D5 D6E340C6 D6E4D5C4 4E4EF0F0  STORED COMPIL DATA NOT FOUND++00
0460 073E00  D5C5E640 C3D6D4D7 C9D340C4 C1E3C140  D5D6E340 C6D6E4D5 C4C1C2E2 E3D9C1C3  NEW COMPIL DATA NOT FOUNDABSTRAC
0480 073E20  E340C6C9 D3C540C5 D4D7E3E8 F1F1E2E3  D6D9C5C4 40C5E7C5 C3E4E3C9 D6D540C4  T FILE EMPTY11STORED EXECUTION D
04A0 073E40  C1E3C140 D5D6E340 C6D6E4D5 C4D5C5E6  40C5E7C5 C3E4E3C9 D6D5080D 040A080D  ATA NOT FOUNDNEW EXECUTION@@@@@@
04C0 073E60  040EE340 C6D6E4D5 C4D9C5E2 E4D3E340  C6C9D3C5 40C5D4D7 E3E8F0F0 F1F14040  @@T FOUNDRESULT FILE EMPTY0011
04E0 073E80  405C090C 014AC5D3 C5E3C5C4 40C2C5C6  D6D9C540 5C5C080D 04FA5C5C 5C40C9D5   *@@@CELETED BEFORE **@@@@*** IN
0500 073EA0  E2C5D9E3 C5C440C2 C5C6D6D9 C5405C5C  5CC3D6D4 D7C1D9C9 E2D6D5F1 F0606060  SERTED BEFORE ***COMPARISON10---
0520 073EC0  60606060 60606040 40405C5C 5C40C3C8  C1D5C7C5 C440E3D6 405C5C5C 4040405C  -------     *** CHANGED TO ***   *
0540 073EE0  5C5C40C1 C4C4C5C4 40E3D640 C5D5C440  5C5C5C40 40405C5C 5C40C4C5 D3C5E3C5  ** ADDED TO END ***    *** DELETE
0560 073F00  C440C6D9 D6D440C5 D5C4405C 5C5CC3D6  D4D7C1D9 C540E3C1 C2D3C540 D6E5C5D9  D FROM END ***COMPARE TABLE OVER
0580 073F20  C6D3D6E6 F1F1F0F0 C2C5C7C9 D5D5C9D5  C74006C6 40E2D6E4 D9C3C540 D5D6E340  FLOW1100BEGINNING OF SOURCE NOT
05A0 073F40  C6D6E4D5 C4404040 40F1F0F0 C1C2E2E3  D9C1C3E3 40C3D9C5 C1E3080D 05BA080D  FOUND     100ABSTRACT CREAT@@@@@@
05C0 073F60  05BED9E2 E4D4D4C1 D9E840D6 C640C5D9  D9D6D9E2 40D5D6E3 40C6D6E4 D5C4F200  @@RSUMMARY OF ERRORS NOT FOUND2@
05E0 073F80  08140090C 018CF0F2 D7D9C560 D3C5E7D4  D6D9C540 E3C8080D 05FAF040 D7C8C1E2  @@@@@@U2PRE-LEXMORE TH@@@@0 PHAS
0600 073FA0  C540D5C1 D4C5E2E3 D6E3C1D3 F0F0C1C2  E2E3D9C1 C3E340C3 D9C5C1E3 C5D6E4F9  E NAMESTOTAL00ABSTRACT CREATEOU9
0620 073FC0  F5D6D7C5 D540D6E4 E3D7E4E3 C9D5F1F0  D9C5C1C4 C7C5E340 C3D6D4D7 C9D3C1E3  5OPEN OUTPUTIN10READGET COMPILAT
0640 073FE0  C9D6D540 C8C5C1C4 C5D95C5C 5C5CC4C7  C3D6E2E5 C5D9E2C9 D6D57A40 D7D9D6C7  ION HEADER**** GCOSVERSION: PROG
```

Figure 4-6. Segment Dump

JOB EXECUTION MESSAGES

Messages may be output to the Job Occurrence Report 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.

Messages Output by the System

The general format of messages output by the system in the Job Occurrence Report is as follows:

ccnn.text

where cc is a two letter classification code and 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 Error Messages and Return Codes manual.

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.

Messages Output by COBOL

The following messages may be output by the COBOL run-time package:

CBL11.DISPLAY	console_displayed_string
CBL12.IFN:ifn	RELATIVE KEY CANNOT BE USED
CBL13.ACCEPT	console_accepted_string
CBL14.IFN:ifn	ORGANIZATION OVERRIDEN
CBL15.IFN:ifn	RECORD LENGTH CONFLICT [ACCEPTED IN INPUT] (length ON FILE)
CBL16. {CALL / CANCEL}	program-name RC = xxxxxxxx siuic, retcode AT ADDRESS stn.ste.sra.
CBL17.STOP literal	
CBL18.IFN:ifn	RC = xxxxxxxx siuic, retcode AT ADDRESS stn.ste.sra program-name [ILN = internal-line [XLN=external-line]]
CBL19.IFN:ifn	CONTROL RECORD 101 TRUNCATED
CBL20.USETST	RC = xxxxxxxx siuic, retcode
CBL21.IFN:ifn	DUMMY FILE NOT DECLARED OPTIONAL IN SOURCE, FILE STATUS 9I NEXT RELEASE.

CBL11, CBL13 and CBL17 are simply DISPLAY, ACCEPT and STOP literal messages which are echoed in the Job Occurrence Report when they are directed to or from a CONSOLE, ALTERNATE CONSOLE or TERMINAL (see Section XI). The remaining messages indicate that an inconsistency has been detected. 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 Error Messages and Return Codes manual for a full list of return codes). Abnormal termination will not occur for message CBL15 if the file is an input file. Abnormal termination will not occur for message CBL14 if overriding is permitted.

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 Job Occurrence Report 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.ste.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 nmemonics is given in the Error Messages and Return Codes manual.

Notes:

- stn.ste.sra are discussed in Dump Analysis, above.

- EX01 and EX03 are normally fatal. EX02 and EX04 are non-fatal.

- 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.

- EX03 need not be fatal if the COBOL program contains a USE AFTER ERROR PROCEDURE SECTION in the DECLARATIVES for the relevant file (See Section IX).

- EX04 is printed when more than 99 EX02 messages have been printed for the current step. After EX04 no futher EX02 messages will be printed.

Four of the more common exception messages are:

```
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 Error Messages and Return Codes manual.

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.
         77 ZONE PIC 9(4).
         PROCEDURE DIVISION.
         P1.
             MOVE HIGH-VALUE TO ZONE.
             STOP RUN.
         .
         .
```

ZONE is a numeric data item. In the native collating sequence (EBCDIC), the figurative constant HIGH-VALUE corresponds to hexadecimal "FF", with all bits set to 1. This configuration is not decimal, hence the exception.

This exception disappears if the DEBUG parameter is included in the $STEP 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 manual for details. See Section V of the current manual for the format of decimal data.

EXCEPTION 17-02 OUT OF ARRAY RANGE

This is caused by attempting to access a data item outside the upper or lower limits of a table.

This exception disappears if the DEBUG parameter is included in the $STEP 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 by 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 $COBOL 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.

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.
COBOL 1,RECERR	the maximum record size of the file is not the same as that specified in the program. See Record Size, Section IX.
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,NAMEERR	identifier in a "CALL identifier" statement does not contain a program-name.
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 Error Messages and Return Codes manual for a description of these return codes.

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,NAMEERR	identifier in a "CALL identifier" statement does not contain a program-name.
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 Error Messages and Return Codes manual for a description of these return codes.

Note:

For the COBOL 1,RECERR, the record length "specified in the program" is one of the following:

- For a report file:
  a) If the RECORD CONTAINS clause is present, the specified record length is augmented by 8.
  b) If the RECORD CONTAINS clause is not present, the record length is assumed to be 140.

- For the other files:
  Unless one of the following conditions exists the record length is taken to be the length of the largest record defined for the file. If one of the following conditions exists this length is augmented by 8.
  a) The internal-file-name in the SELECT clause is suffixed by -PRINTER or -SYSOUT and WITH ASA or WITH SARF is not specified.
  b) With SSF is specified in the SELECT clause.
  c) The LINAGE clause is specified in the File Description entry.
  d) The file is referenced in a WRITE statement with the BEFORE/AFTER ADVANCING phrase.

SECTION V

REPRESENTATION OF DATA

This section describes the way in which data descriptions are interpreted by the Level 64 COBOL compiler and the way in which data is held in memory in a COBOL program.

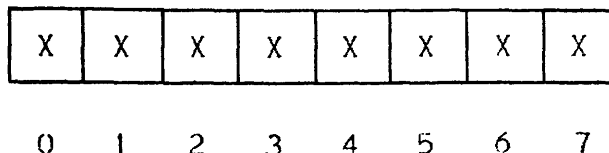
The value of a numeric item may be represented in either binary or decimal form. In addition there are several ways of expressing decimal. The selection of radix is dependent upon factors included in clauses such as USAGE.

The types of data supported by Level 64 COBOL are listed in Table 5-1, according to factors included in the USAGE and PICTURE clauses. The usage of an item spcifies 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.

FORMAT OF DATA IN MEMORY

The basic element of information in Level 64 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:



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.

An unpacked decimal number (PIC 9 or PIC S9) has the following format. Note, for PIC 9 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
-	1101

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.

PACKED DECIMAL NUMBERS

A packed decimal number (USAGE IS COMP, COMP-3 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 or COMP-3 and the PICTURE character string does not have a sign, the rightmost four bits represent the rightmost digit. Unsigned COMP and COMP-3 items should be avoided, if possible, for efficiency reasons (see Section VIII, Efficiency Techniques).

A signed packed decimal number (COMP, COMP-3 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 (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
1010	+
1011	-
1100	+
1101	-
1110	+
1111	+

Signs put in result fields by instructions are encoded in the following manner:

Sign	Sign Encoding
+	1100
-	1101

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	$\frac{L + 1}{2}$

When L is even, the leftmost digit position must be zero.

Table 5-1. Data Representation in Level 64 System

USAGE	MACHINE DESCRIPTION	PICTURE
DISPLAY	EBCDIC byte or unpacked decimal	R
*COMPUTATIONAL or COMP	Packed decimal (possibly without sign position depending on PICTURE)	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-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
INDEX	6 bytes	NA

Notes for Table 5-1:

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 specifies more than 4 digits, a 32 bit fixed-point binary data item is used.

FIXED-POINT BINARY NUMBERS

Fixed-point binary data can be specified as either 16-bit binary (USAGE IS COMP-1 and no PICTURE, or a PICTURE showing less than 5 digits) or 32 bit binary (USAGE IS COMP-2 or COMP-1 with a PICTURE 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.

FLOATING-POINT BINARY NUMBERS

Floating-point binary data can be specified either as 32-bit binary (USAGE IS COMP-9) or 64-bit binary (USAGE IS COMP-10). 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 value of a floating-point binary number, V, is defined by the following equation:

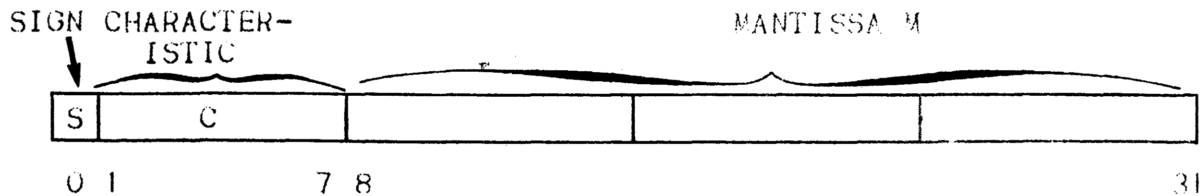
$$V = (-1)^S \times 16^E \times .M$$

Where E = C-64

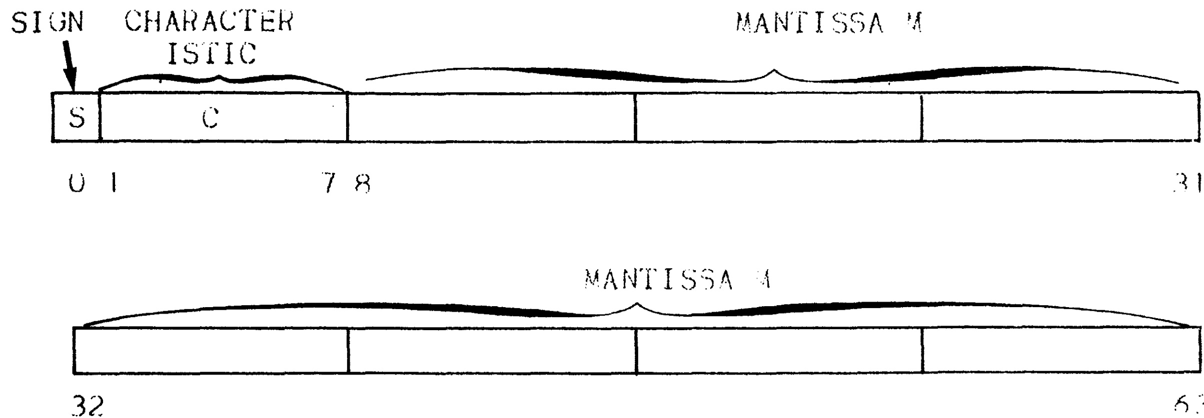
S is the sign, E is the exponent, C is the characteristic, and M is the mantissa of the floating-point binary number.

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.

A short floating-point binary number occupies four bytes. The format is as follows:



A long floating-point binary number occupies eight 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. The radix point is at the left of the high-order digit position.

INDEX DATA ITEM

An INDEX data item (USAGE IS INDEX) consists of 48 bits (six bytes) of binary data: the first four bytes (COMP-2) contain the relative displacement from the beginning of the table and the last two bytes (COMP-1) contain the occurrence number of the table element.

SECTION VI

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.

The use of calling and called programs has the following advantages:

- A called program can be written and compiled and stored in a compile unit library. This program 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 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. However, this facility is not part of the ANS standard. 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 Section III, Linking.

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-ID value. For example:

```
CALL "PROG2".
```

In this example control will transfer to the program whose name is PROG2. (All 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 beween calling programs and called programs. A calling program contains a CALL statement which refers to a separately compiled program. A called program is the subject of a CALL statement in a 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. 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 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 section.

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.
     .
     .
     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 which 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 are unpredictable.

THE EXTERNAL PHRASE

The EXTERNAL phrase may be included 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 constituent data items available to every program in the load module which describes that record. The EXTERNAL phrase is used as an alternative or complement to the LINKAGE SECTION and USING phrase. It is not part of the ANS a standard. 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 data-names). 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.

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, the called compile units must be specified to $LINKER in the INCLUDE command. See Section III, Linking.

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 Section II).

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.

INTERFACE WITH FORTRAN PROGRAMS

Programs written in FORTRAN 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 FORTRAN programs must be in the form of a normal FORTRAN 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 FORTRAN are shown in Table 6-1. No other data formats are permitted.

Table 6-1. Data Formats in FORTRAN Called Programs

Data Format	COBOL Data Description	FORTRAN Declaration
Alphabetic, alphanumeric or edited.	PIC A PIC X PIC $9.9 etc	CHARACTER*n
One word binary.	COMP-1 PIC S99999.. COMP-2	INTEGER
Single word floating point.	COMP-9	REAL
Double word floating point.	COMP-10	DOUBLE PRECISION

COMP-9 and COMP-10 items may only be used in MOVE, INITIALIZE (conversion), PROCEDURE DIVISION USING..., and CALL USING... They have been introduced for communication with FORTRAN programs.

The following example shows a COBOL program calling a FORTRAN program:

```
          .
          .
      WORKING-STORAGE SECTION.
          .
          .
      77  ANGLE             COMP-10.
          .
          .
      PROCEDURE DIVISION.
          .
          .
          CALL "COSINESQ" USING ANGLE.
          .
          .
```

The called FORTRAN program may be as follows:

```
      .
      .
      SUBROUTINE COSINESQ (ARG)
      .
      .
      DOUBLE PRECISION ARG,WORK
      .
      .
      WORK = COS (ARG)
      ARG = WORK**2
      .
      .
      RETURN
```

Programs written in COBOL can also be called by FORTRAN 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 FORTRAN CALL statement. The data formats which can be used for arguments have been shown above.

CONSTRAINTS

The following paragraphs describe some constraints which must be observed when writing calling and called programs.

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. 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, though the COBOL file-names may be different.

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 are local to the program in which they are described. That is, if program A reads a record from a file, this record is not automatically available to program B when it is called even though program B contains an identical description (including EXTERNAL) for this file.

Report Writer

A report description (RD) is local to a program. Therefore, Report Writer statements used in a particular program can refer only to a report description in the same program.

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 Section XII.

GUIDELINES

The benefits of calling and called programs have been listed at the beginning of this section. If none of these benefits applies to a particular program, then calling and called programs need not be used.

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 programs. The transfer of control between source modules can be made using the PERFORM statement rather than the CALL and EXIT PROGRAM statements. The PERFORM statement is more efficient than the CALL and EXIT PROGRAM statements. See Section VII for more details of the use of structured programming with the PERFORM statement.

SECTION VII

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". For a description of Virtual Memory Management see the System Management 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 posssible: third, the CPU time consumed by Virtual Memory Management 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 would be too large to fit into the available memory in one piece can be executed in segments.

The remainder of this section provides guidelines for efficient program segmentation.

Note: For an explanation of COBOL segment number, internal segment number and $LINKER segment number see Section III, Linking.

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 $COBOL 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 $COBOL statement or by clauses in the OBJECT-COMPUTER paragraph (described below).

CONTROL OF SEGMENTATION BY THE PROGRAMMER

Segmentation should be viewed as a tool by which a user programmer can take advantage of Virtual Memory Management. The user program should be segmented according to the logical structure of the program. This will enable Virtual Memory Management 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.

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 Section III, 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.

For Level 64, Virtual Memory Management eliminates the distinction between "permanent" and "overlayable" segments: both are of the same type, and both are swappable. Only the distinction between fixed and independant 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 hierachy. In other words, the program should be "structured". The subject of structured programming will not be discussed further in this manual. The reader is recommended to consult one of the many publications on this subject for more details.

Each PROCEDURE DIVISION module should be written as a separate SECTION and should be activated by a PERFORM. 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 7-1.

Figure 7-1 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 <u>COBOL Language Reference Manual</u>.

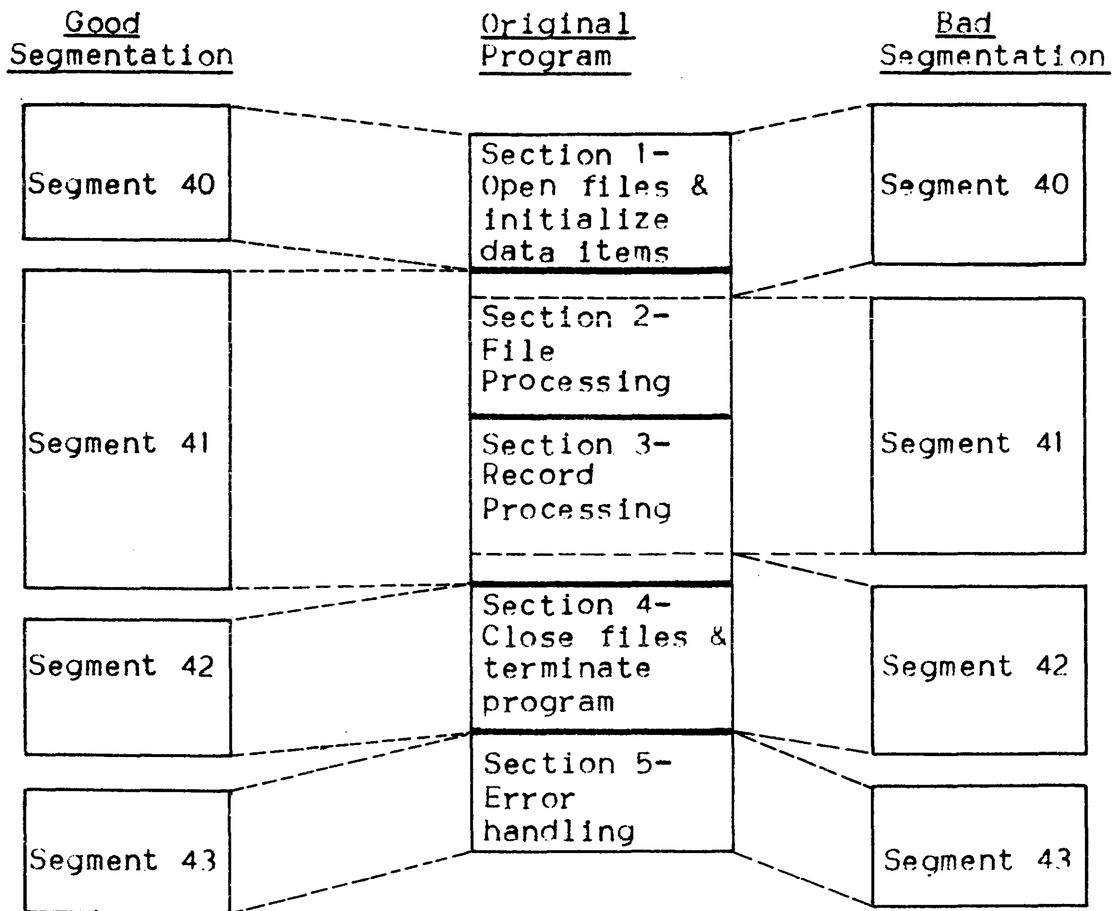


Figure 7-1. PROCEDURE DIVISION Segmentation

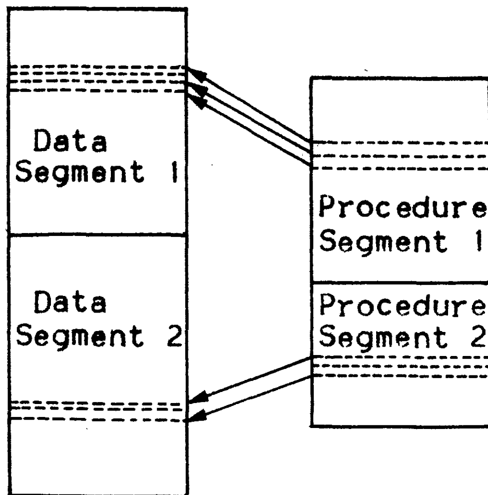
DATA DIVISION Segmentation

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 should be remembered:

- 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).

Examples of good and bad DATA DIVISION Segmentation are given in Figure 7-2.

Good Segmentation



Bad Segmentation

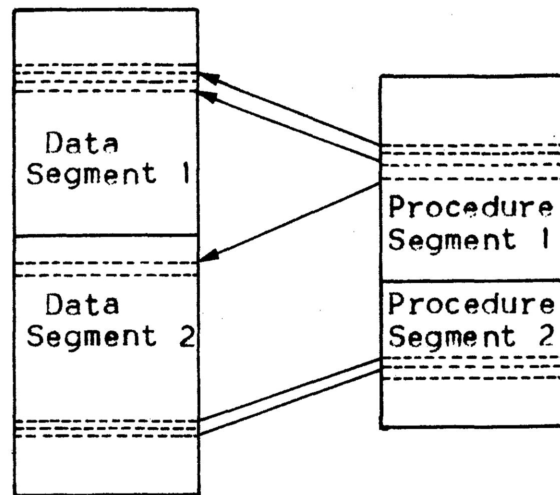


Figure 7-2. DATA DIVISION Segmentation

Figure 7-2 shows an example of good segmentation in which all data referenced by a given procedure segment is in a single data segment (provided this segment is not too large). Also shown is an example of bad segmentation in which a procedure segment references data in more than one data segment. However, such segmentation cannot always be avoided. It may be more efficient in certain cases to segment in the "bad" way shown above in order to reduce the number of segments and produce a set of segments of approximately the same size.

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 $COBOL statement using the PSEGMAX and DSEGMAX parameters.

The use of the $COBOL parameters is recommended, as the MAXIMUM PROCEDURE SEGMENT SIZE and MAXIMUM DATA SEGMENT SIZE clauses are not part of the ANS standard.

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 Virtual Memory Management 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 $COBOL statement can be used to merge linkage and code segments when the code segment is small (see Section II, Compilation).

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 $COBOL statement, and the total size of the code segment and the linkage segment together is not larger than the preferred procedure segment size, the linkage segment and 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.

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 earlier segment until the preferred data segment size is reached.

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 4 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                                |
    LABEL RECORD STANDARD               |
    DATA RECORD ARTOUT.                 |
01  ARTOUT PIC X(80).                   |
FD  CARD                                |
    LABEL RECORD STANDARD               |
    DATA RECORD ARTIN.                  |
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 compiler is as follows:

```
SEGLIM01.0       ..L      176
SEGLIM01.1       .D.     3296
SEGLIM01.2       .D.     4000
SEGLIM01.3       .D.     5000
SEGLIM01.4       .D.     4000
SEGLIM01.5       .D.     3008
SEGLIM01.6       .D.     3008
SEGLIM01.7       .D.     3424
SEGLIM01.8       C..      144
SEGLIM01.9       C..      464
SEGLIM01.10      C..      352
SEGLIM01.11      C..      592
```

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 interactive sequence of code (a loop) to be divided between two segments. In the worst case this can result in both segments being swapped each time the loop is executed (if this happens, there is memory overload).

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.

INTERNAL SEGMENT NUMBERS

A COBOL compile unit can have no more than 128 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 the code segment (CODAPND parameter in the $COBOL statement).

- Each file in the program uses two ISNs.

- If the Program Checkout Facility is to be used (DEBUG parameter in $COBOL) 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 $COBOL statement is active, one or more additional segements are generated. One ISN is given to each segment.

- If a program has EXTERNAL files, one ISN is used for each EXTERNAL file.

- Each EXTERNAL data item with a VALUE clause is given one ISN.

- Each translation table (used for alphabet-name, TRANSFORM...) is given one ISN.

If more than 128 ISNs are needed during a compilation the compiler will print the fatal error message 8-94 and will then terminate.

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 statement. The amount of memory required is known as the declared working set (DWS). If no DWS is specified, the program will be allocated 35K of main memory.

An explanation of DWS together with a description of how to calculate the DWS value for a program is given in Section VII of the System Management Guide.

SECTION VIII

EFFICIENCY TECHNIQUES

The following techniques are recommended to obtain efficient COBOL object programs. Consideration is given to data manipulation and data description techniques. See Section VII, Segmentation, for guidelines on efficient segmentation. See the UFAS User Guide, BFAS User Guide and HFAS User Guide for guidelines 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-1, COMP-2...).

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 COMP-1 or COMP-2 item or else move it to a temporary area in working-storage (described as COMP-1 or COMP-2) 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 $COBOL 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, Section IV, 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 working-storage 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, addition, and subtraction, give the items similar PICTUREs and USAGEs 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 PICTUREs and the same usage. (T)

- Tend to use procedural literals rather than constant values in WORKING-STORAGE. The compiler can optimize the format of procedural literals, but must resort to dynamic format conversions in the object program if WORKING-STORAGE items are not ideally formatted. However, duplicate literals do result in extra memory space requirements. (T)

- Use GO TO...DEPENDING for decisions whenever possible. In any application for which GO TO...DEPENDING 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 following 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.

DATA DESCRIPTION TECHNIQUES

- Use COMP, COMP-3 or COMP-8 for non-integer data items and for data items which interact with other COMP, COMP-3 or COMP-8 data items. 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. In Level 64 the righthand 4 bits of a packed decimal item represent the sign. When an unsigned COMP or COMP-3 data item is used in a computation the run-time package must add a dummy sign position to the data item before computation can be started. (T and S)

- Use COMP-1 or COMP-2 for integer data items which are not involved arithmetically with data items of other usages. (T)

- Specify 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 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 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 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)

- 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 processsing. 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 COMP-1, COMP-2, COMP-9, COMP-10 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).

SECTION IX

FILES

This section contains information which is relevant for all types of files. In addition, the use of unit record files is discussed in detail in Section XI.

FILE NAMES

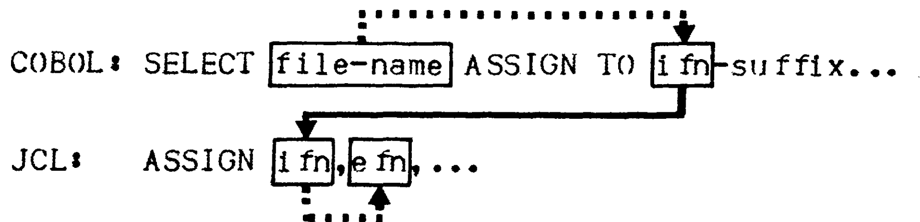
Each file is known by three different names:

- 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 statement each time the job is to be run.

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 Job Control Language (JCL) Reference Manual.

- The rules for external-file-names are different for cataloged files, uncataloged files and temporary files. See the Job Control Language (JCL) Reference Manual Section III 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, -SYSOUT and -TAPE have a significance other than documentation. The significance of -PRINTER and -SYSOUT is explained in Writing SSF Files in COBOL Programs, Section X. -TAPE is only significant for H-2000 files. For these files -TAPE means that the CHARACTERS option of the BLOCK CONTAINS clause implies variable length records. 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 statement.

DATA MANAGEMENT OVERRIDING RULES

The source program normally defines the basic file characteristics. However, a number of file parameters (e.g. file organization, blocksize, device class) can be specified and recorded at different places in the system and at various stages in the creation of the file. Table 9-1 shows where these parameters are specified and where they apply.

Table 9-1. Specification and Applicability of File Characteristics

Where specified	Where Applicable
At system generation	Throughout the job
In source program	Compilation time
In JCL ($ASSIGN & 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 User Guide, BFAS User Guide or HFAS User 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.

The maximum record length 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 Job Occurrence Report:

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 "95" 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 "9U" is returned

- on a WRITE statement, a file-status "92" 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 neither H-2000 KEYED, nor H-2000 INDEXED, and it is opened in input by a program compiled with the current version of the compiler: the following message appears in the Job Occurrence Report:

  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 "9U" is returned to the program.

The actual file organization that may be associated with a file is shown in Table 9-2.

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.

OPTIONAL FILES

An optional file is one which may be absent at execution time, even though OPEN, CLOSE, READ, WRITE etc... may be attempted for the file.

Table 9-2. Permitted File Organizations

ORGANIZATION in COBOL program	Type of File									
	UFAS			BFAS				HFAS		
	Seq	Rel	Ind	Seq	Dir	Ind	Que	Seq	Ind	Rand
SEQUENTIAL no qualifier UFF LEVEL-64 H-200	1	2	2	1	2	2	1	1	2	
RELATIVE no qualifier UFF LEVEL-64	3	1		3	1	4	3	3	4	
INDEXED no qualified UFF LEVEL-64 H-200			1			1			1	
H-200 KEYED										1

Notes for Table 9-2 :

(1) allowed.

(2) allowed if the file is OPENed in INPUT or I-O.

(3) allowed when ACCESS IS SEQUENTIAL if it is a disk file that is OPENed in INPUT or I-O, or, when ACCESS IS RANDOM or DYNAMIC, if it is a disk file that is OPENed in INPUT (START cannot be used if the RELATIVE KEY clause is used).

(4) allowed when ACCESS IS SEQUENTIAL if the file is OPENed in INPUT.

From the point of view of ANS standard COBOL, any optional file (i.e., the OPTIONAL parameter is present in the SELECT clause) must be of sequential organization, and must be used for input only (i.e., only OPEN INPUT, READ and CLOSE may refer to such a file). From the system point of view, all files may be optional. A distinction is made between those which are declared absent when the execution JCL is written, and those which are declared absent by the operator.

Files which are declared optional in the COBOL program must have a corresponding $ASSIGN statement in the execution JCL, regardless of whether the file is to be used or not. The $ASSIGN statement will normally include a DUMMY or OPTIONAL parameter (discussed below). If neither of these parameters is present in the $ASSIGN 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 statement and the OPTIONAL phrase is not present in the corresponding COBOL SELECT clause, the message CBL21 is output in the Job Occurrence Report and the program behaves as if OPTIONAL was present in the SELECT clause. However, the user should normally include the OPTIONAL phrase in the SELECT clause. In a future release of the compiler a file status 9I will be generated if the OPTIONAL phrase is omitted for an optional file.

If the DUMMY or OPTIONAL parameter is specified in the $ASSIGN statement of a file which is not a sequential input file, a status 9I is returned to the program and the situation may be handled by a USE AFTER ERROR PROCEDURE SECTION (see Error Handling, below).

The DUMMY and OPTIONAL parameters are used in the following way:

- JCL declaration using the $ASSIGN statement DUMMY parameter:

  ```
  ASSIGN ifn, DUMMY;
  ```

  This specifies that the file is absent and all references to the file should be ignored.

- 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 media is mounted and the file exists, the file is processed as normal.

The use of optional files is summarized in Figure 9-1.

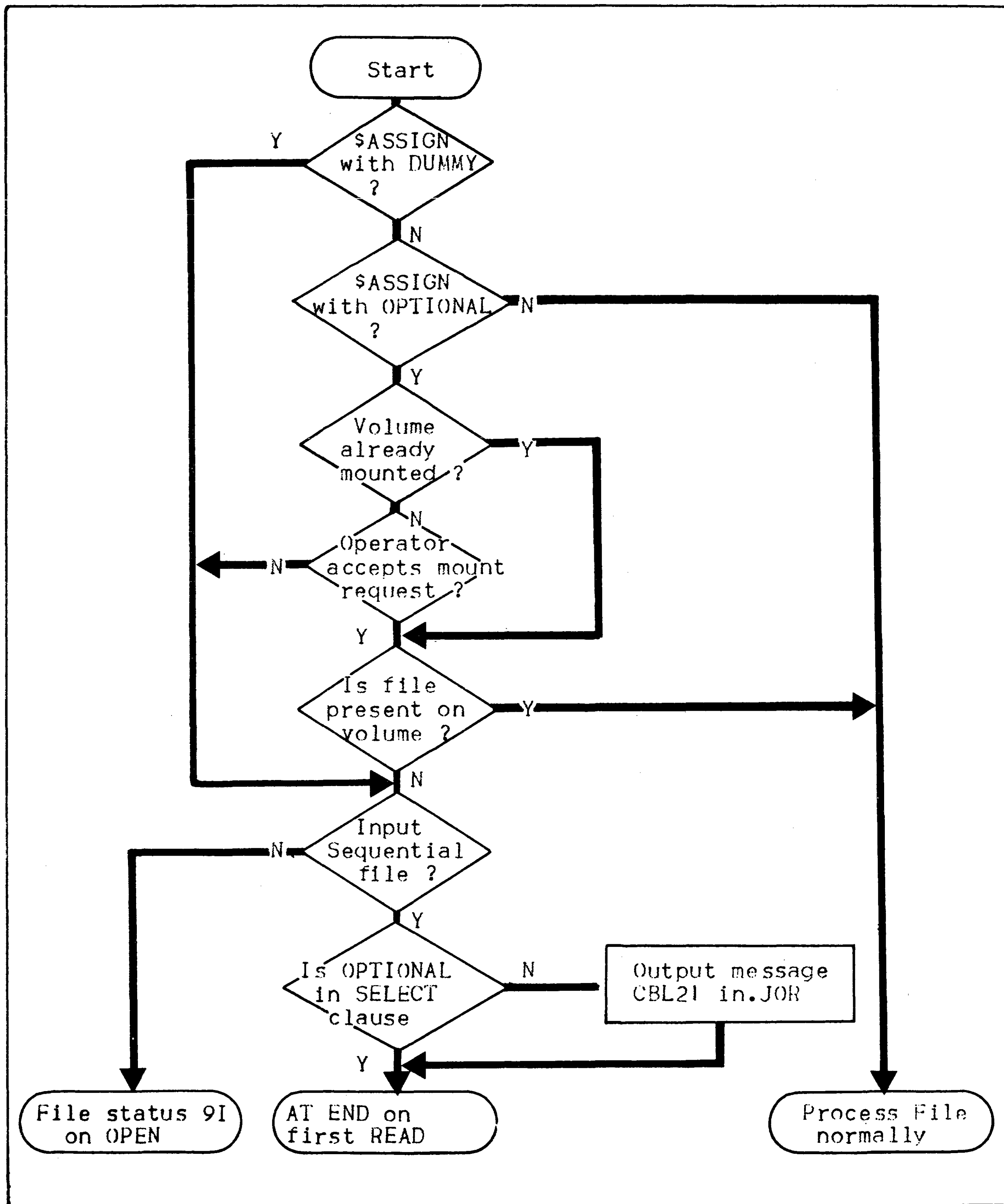


Figure 9-1. The Use of Optional Files

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 reopening 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 9M (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).

THE $POOL STATEMENT

Normally, access to a particular file on a device is granted exclusively to a program for the duration of a job step. Suppose, however, 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. Therefore it would be possible to use the same device for F1 and F2. The user can inform the system that this is possible by using the $POOL statement in conjunction with the POOL parameter in $ASSIGN:

```
POOL 1*MS/M402;
ASSIGN F1, MAX.Z, POOL, FIRST,...
ASSIGN F2, BMY.I, POOL, NEXT,...
```

Thus only one MSU0402 will be reserved for the use of the pooled files.

When a program closes a file, the system is able to free the device assigned to the file for the use of another volume in the pool . In the above COBOL example, the file FILE1 itself has been deassigned 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 $SPOOL statement see the Job Control Language (JCL) User Guide.

MULTIVOLUME 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 multivolume file. The following paragraphs are applicable to sequential multivolume files only. Each part of such a file, disk or tape, is known as a "physical unit".

In the case of disk volumes which contain other files also, only part of each volume will be occupied by a physical unit of the multivolume file. The space for disk files is allocated using the $PREALLOC utility (see the Data Management Utilities Manual). Only one $ASSIGN statement is needed for a multivolume disk file or multitape file. The volume-names must be listed in the MEDIA parameter of the $ASSIGN statement or recorded in the catalog entry for the file.

Boundaries between volumes in a multivolume file are usually invisible to the COBOL program which processes such a file. The COBOL program does not have to contain any special code to handle multivolume 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 multivolume tape files, and for multivolume HFAS sequential disk and 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 multivolume tape files, for multivolume HFAS sequential files and, if the MOUNT=1 parameter is used (or implied) in the $ASSIGN statement, for multivolume BFAS sequential files.

For more information about multivolume files see the Job Control Language (JCL) User Guide.

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, BFAS, HFAS). 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 monovolume or multivolume file. If a logical unit occupies several volumes this may be because the logical unit is a "multivolume 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 multivolume 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 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 a $ASSIGN statement in the JCL for each logical unit to be processed. There may also be a $POOL 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 only one logical unit is to be processed by the COBOL program a normal $ASSIGN statement must be used with no suffix on the internal-file-name. For example:

```
ASSIGN IFILE, E.FILE,...
```

If a COBOL program is to process multi logical unit files the VACSEG (vacant segment) parameter must be used with $LINKER. The format of this command is:

```
VACSEG = (SHARE = +a)
```

where "a" is calculated as follows. For each multi logical unit file in the program the maximum number of logical units is multiplied by 2 and one is added. The resulting values for each multi logical unit file are added together and augmented by 2 to give "a". See Section III for details of $LINKER.

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 open at one time). *

There must be a SELECT clause in the FILE-CONTROL paragraph and a $ASSIGN statement in the JCL for each file in a multiple file tape volume. There may also be a $POOL 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 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,DEVCLASS=MT/T9,MEDIA=TAPEA,FSN=1,POOL,FIRST;
  ASSIGN IFILEB,E.FILEB,DEVCLASS=MT/T9,MEDIA=TAPEB,FSN=2,POOL,NEXT;
  ASSIGN IFILEC,E.FILEC,DEVCLASS=MT/T9,MEDIA=TAPEC,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,DEVCLASS=MT/T9,MEDIA=TAPEA,
    FSN=1,POOL,FIRST;
   ASSIGN IFILE-2,E.FILEB,DEVCLASS=MT/T9,MEDIA=TAPEB,
    FSN=2,POOL,NEXT;
   ASSIGN IFILE-12,E.FILEL,DEVCLASS=MT/T9,MEDIA=TAPEC,
    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 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 which must be used in the $ASSIGN 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 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 Language Reference Manual.

FILE CONCATENATION

File concatenation should be distinguished from the concepts of multivolume files, multi logical unit files and multiple file tape volumes which have been discussed above.

Several UFAS or BFAS sequential tape files may be accessed in sequence by means of file concatenation; the files are treated by the program as if they are 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 multivolume files. These statements cause the next volume to be opened. This volume may be the next volume in a multivolume file or the first volume of the next concatenated file. RERUN on CHECKPOINT-FILE EVERY END OF REEL/UNIT also operates on end of volume.

File concatenation may also be used with cassette files. File concatenation is performed by the specification of the respective $ASSIGN 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 statement be used to allocate a single device for all the files to be concatenated. For example:

```
POOL 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 same RECFORM, BLKSIZE and RECSIZE parameters and must have the same device class and device attributes.

One or more of the files to be concatenated may be a multivolume file if required. Furthermore, file concatenation can be used in a multi logical unit file. One or more of the $ASSIGN statements for a multi logical unit file can use the file concatenation facility.

UFAS, BFAS AND HFAS

The following file access systems are available with Level 64:

- UFAS - Universal File Access System.

- BFAS - Basic File Access System

- HFAS - H200/2000 File Access System.

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.

BFAS is an alternative to UFAS. BFAS is a relatively device dependent file access system which may offer better performance for certain types of application.

HFAS reads and writes files in H200/2000 formats. This file access system should be used for compatibility only.

UFAS and BFAS are the Level 64 native file access systems. They are completely compatible. For magnetic tape, the file access system used is irrelevant because UFAS and BFAS tapes have exactly the same format. Sequential and indexed HFAS files are compatible in certain respects with UFAS and BFAS files.

Some notes concerning UFAS, BFAS and HFAS are given below. For further information refer to the UFAS User Guide, BFAS User Guide, or HFAS User Guide.

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 specified.
- LEVEL-64 (specifies BFAS).
- H-2000 (specifies HFAS).

However, this option is not part of the ANS standard and is significant only in the following circumstances:

- The file is an output disk file.
- A $PREALLOC statement has not been used for the file.

It is recommended that $PREALLOC be used for all permanent output disk files. $PREALLOC provides a centralized and visible method of specifying file attributes.

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 Manual for details of $SORTIDX.

Note that the APPLY NO-SORTED-INDEX clause is not part of the ANS standard.

APPLY NO-RESIDENT-INDEX

Unless otherwise specified, the cylinder index tables of BFAS indexed sequential files are automatically loaded into memory. As a result, access times are reduced, particularly where access is RANDOM.

However, the user may have some reason (e.g.the indexes are very large) for wanting the indexes to be non-resident. In this case the APPLY NO-RESIDENT-INDEX clause must be used in the I-O-CONTROL paragraph for the relevant file.

Note that the APPLY NO-RESIDENT-INDEX clause is not part of the ANS standard.

The $DEFINE statement parameters NRESIDX and RESIDX can be used instead of the APPLY NO-RESIDENT-INDEX clause . $DEFINE can also be used to override the APPLY NO-RESIDENT-INDEX clause.

ERROR HANDLING

If a system procedure returns to the COBOL program an abnormal return code as a result of an I/O operation, the step is abnormally terminated and an exception report is printed in the Job Occurrence Report. 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 associated with the file or the system return code obtained by calling the routine H_CBL_UGET4. These subjects are discussed in the following paragraphs.

The FILE STATUS

The following example shows the way in which a USE AFTER ERROR PROCEDURE SECTION and FILE STATUS 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 Language 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 FILE STATUS can be set are given in the COBOL Language 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 Job Occurrence Report.

```
CBL18 IFN:ifn rc AT ADDRESS address [ILN=iln [XLN=xln] ]
```

Return Code

The FILE STATUS facility is part of ANS standard COBOL and is described in the COBOL Language Reference Manual. The information provided in the FILE STATUS 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_UGET4 in the COBOL run-time package. This facility is not part of the ANS standard and for this reason should be avoided whenever FILE STATUS provides sufficient information. The following example shows the use of H_CBL_UGET4:

```
.
WORKING-STORAGE SECTION.
77    RET-CODE-1    USAGE COMP-1.
77    RET-CODE-2    USAGE COMP-1.
77    RET-CODE-1X   PIC S9(5).
77    RET-CODE-2X   PIC S9(5).
.
.
PROCEDURE DIVISION.
DECLARATIVES.
FILEA-ERROR SECTION
      USE AFTER ERROR PROCEDURE ON FILEA.
P1.
      CALL "H_CBL_UGET4" USING RET-CODE-1 RET-CODE-2.
      MOVE RET-CODE-1 TO RET-CODE-1X.
      MOVE RET-CODE-2 TO RET-CODE-2X.
      DISPLAY "RET.CODE = " RET-CODE-1X RET-CODE-2X.
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 Error Messages and Return Codes Manual.

RESTRICTIONS ON CERTAIN FILE ORGANIZATIONS

Some features, though described in the COBOL Language Reference Manual, are not available for certain file organizations. These are listed in Table 9-3.

Table 9-3. Features Not Available with Certain File Organizations.

Feature not available	File organization
Physical units	UFAS disk sequential BFAS disk sequential
RERUN EVERY END OF UNIT	UFAS sequential
Variable length records	BFAS indexed and relative HFAS sequential(disk)indexed, and keyed.
Creation of files when ACCESS IS RANDOM and keys are not in ascending order	HFAS indexed
ALTERNATE RECORD KEY	BFAS indexed HFAS indexed
START subkey	HFAS indexed
START > , START NOT <	BFAS relative

When such a feature is used for a file organization where it is not available, it generally results in a file states "30" returned to the program and a return code mnemonic FUNCNAV reported in the Job Occurrence Report.

RECORD SIZE

The maximum record size used on a file is determinated according to the following criteria:

- For a report file:
  a) If the RECORD CONTAINS clause is present, the specified record length is augmented by 8.
  b) If the RECORD CONTAINS clause is not present, the record length is assumed to be 140.

- For the other files:
  Unless one of the following conditions exists, the record length is taken to be the length of the largest record defined for the file. If one of the following conditions exists, this length is augmented by 8.
  a) The internal-file-name in the SELECT clause is suffixed by -PRINTER or -SYSOUT and WITH ASA or WITH SARF is not specified.
  b) WITH SSF is specified in the SELECT clause.
  c) The LINAGE clause is specified in the File Description entry.
  d) The file is referenced in a WRITE statement with the BEFORE/AFTER ADVANCING phrase.

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 $SORT. 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. That is, the data file is not actually sorted by $SORT, but it can be read in the sorted sequence by a COBOL program, using the address file produced by $SORT.

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=L64 parameter is specified in the $COBOL statement when the program is compiled. Note that the ACTUAL KEY phrase is not part of the ANS Standard.

$SORT 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 statement it must contain the value TTRDD (this is the default value) which specifies the format of the address.

SECTION X

STANDARD RECORD FORMATS

There are four standard record formats recognized by Level 64 Data Management. These formats may be used in magnetic tape or disk files. They are:

- 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 which are passed between COBOL programs.

- System Standard Format (SSF)
  In this format each record comprises on 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 on tape to any kind of I/O device. This format provides the Stream Reader, compilers, $LIBMAINT and Output Writer with a standard method of handling their input and output data.

- 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 which are to be processed solely within the Level 64 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.

- Device Oriented Format (DOF)
  In this format each record comprises an eight byte header followed by normal data. The header contains device oriented control information in the form used by the various unit record devices. This format is only used by Level 64 "Program Mode" PM100 and PM200 systems.

ASA and DOF are rarely used by the COBOL programmer and will not be discussed further in this section. The remainder of this section discusses SSF and SARF.

Note: Whenever the word "files" is used in the remainder of this section it should be taken to mean "files or subfiles".

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 which have been truncated at the rightmost end of the record. Truncation (packing) occurs only in records which 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 an SSF 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 implicity or explicity specified as SSF.

The following paragraphs discuss the relationship between SSF and the Stream Reader, $LIBMAINT, the COBOL compiler, COBOL programs and the Output Writer.

The Stream Reader, $LIBMAINT and the COBOL Compiler

An SSF file can be created from cards contained in 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.

The standard SYSIN subfile may then be read by any job step or utility. For example it can be read by $LIBMAINT 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. Card images from the input enclosure are 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 Section I, Input and Maintenance of Source Programs.

An SSF library member may be read by a user program. 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 Section I and will not be discussed further in the current section.

If the SSF library member contains a COBOL program it can be processed by the COBOL compiler. 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 Section II.

If the SSF library member contains JCL it can be used by a $INVOKE, $EXECUTE or $RUN statement (see Job Control Language (JCL) User Guide).

Reading SSF Files in COBOL Programs

In a COBOL program any input file may be SSF or SARF irrespective of the way in which it is described in the COBOL SELECT clause. The presence or absence of a type 101 control record at the start of the file indicates the format of the file. Level 64 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. 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 it will be treated as an SSF file by Data Management. 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 nonzero 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 $LIBMAINT 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.

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 Section XI, 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 Section XI and will not be discussed in the present section.

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 which are

passed between COBOL programs. However, SARF files may also be handled by the Steam Reader, $LIBMAINT, the compilers and the Output Writer. The following paragraphs discuss the use of SARF format.

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 tre $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 card images from the input enclosure are 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, it can be processed by the COBOL compiler. The use of the compiler is covered in Section II. If the SARF library member contains Job Control Language it can be used by a $INVOKE, $EXECUTE or $RUN statement (see <u>Job Control Language (JCL) User Guide</u>).

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.

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 Section XI, Using Unit Record Files.

GENERAL POINTS CONCERNING SSF AND SARF

The Output Writer

The Output Writer can print or punch any SSF or SARF file. It is called by the statements $SYSOUT and $WRITER. The Output Writer is normally used to output print or punch files produced by user programs. However, it can also print or punch files which 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 Section XI.

Summary of Rules for the SELECT Clause

Table 10-1 summarizes the rules concerning the COBOL SELECT clause when using SARF or SSF files.

Table 10-1. 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 thet 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).

SECTION XI

USING UNIT RECORD FILES

This section 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);

*

- Cassette files.

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, Section XII, for information concerning printed reports produced using the Report Writer facility.

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 need be made 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 or tape 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). A permanent SYSOUT file must be assigned by the user.

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 statement or later in the $DEFINE 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 which 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 then when the file is written, is relatively inefficient and should be avoided. See the Job Control Language (JCL) User 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 $LIBMAINT), 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 Section X, Standard Record Formats.

The method of producing permanent and standard SYSOUT files with or without edited SYSOUT format is summarized in Table 11-1.

Table 11-1. Methods of Producing SYSOUT Print Files

TYPE OF SYSOUT FILE CREATED	JCL		COBOL
	$ASSIGN	$SYSOUT	-SYSOUT
PART 1 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 statement will be ignored. This parameter is used only with permanent SYSOUT files.	NO	YES	OPTIONAL
	NO	OPTIONAL	YES
PART 2 The step is abnormally terminated when the file is opened because no implicit or explicit file assignment has been made(RC=IFNNASG).	NO	NO	NO
PART 3 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 statement.	YES	YES	NO
PART 4 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 not printed by the Output Writer. A $WRITER statement must be given to print the file.	YES	NO	YES
PART 5 A permanent SYSOUT file is written. The file will not be in edited SYSOUT format, irrespective of the record size. The file is not printed by the Output Writer. A $WRITER statement must be given to print the file.	YES	NO	NO

The following notes explain the headings used in Table 11-1:

$ASSIGN	is a $ASSIGN of a permanent SYSOUT file to be included in the JCL (YES or NO) ?
$SYSOUT	is a $SYSOUT of 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) ?

In part 1 of Table 11-1 a standard SYSOUT subfile is automatically assigned by the system. As can be seen from the table, this only happens when the user does not explicitly assign a SYSOUT file and when one or both of the following conditions apply:

- The $SYSOUT statement is used.

- The -SYSOUT suffix in the COBOL SELECT clause is used.

If neither $SYSOUT nor -SYSOUT is specified, the user must assign a permanent SYSOUT file using the $ASSIGN statement. This is done in part 5 of the table. In part 2 of the table the user does not assign a permanent SYSOUT file and the step is abnormally terminated.

In part 1 of the table the SYSOUT file will be written in edited SYSOUT format. This will also be the case in parts 3 and 4 if the SYSOUT file has been preallocated 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. This is the case in part 5 of the table.

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 statement (see part 1 of the table). Permanent files are printed automatically only if there is a $SYSOUT statement in the job step JCL and if this statement does not contain the WHEN = DEFER parameter (see part 3 of the table). In all other cases the permanent SYSOUT file should be printed in a separate job step by using the $WRITER statement (see parts 4 and 5 of the table)

All the SYSOUT files written according to the rules in Table 11-1 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.

Printing Directly

When the printer is used directly, a $ASSIGN statement must be present at execution time which 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 $DEFINE is optional. See the Job Control Language (JCL) Reference Manual for details of the relevant $DEFINE parameters.

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, vertical paper movement is controlled by a software simulated vertical format unit (VFU). 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 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 $ASSIGN, $OUTVAL, $SYSOUT and $WRITER in order to ensure that the correct VFU, form height etc. are used when the file is printed. See the Job Control Language (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 statement. See the Job Control Language (JCL) Reference Manual for details.

Note that all form control parameters specified in SYS.URCINIT and in $DEFINE for a given file are ignored at execution time if either the LINAGE clause or the Report Writer is used with that file.

The LINAGE Clause

The LINAGE Clause can be used in an FD statement 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):
a. The WRITE is done either before or after advancing nn lines, as specified in the program.
b. LINAGE-COUNTER is increased by nn.
c. 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. A new value is set-up for LINES AT TOP, if the COBOL program has changed this value.
b. 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.
c. LINAGE-COUNTER is set to 1.
d. New values are set-up for LINAGE, FOOTING and LINES AT BOTTOM, if the COBOL program has changed these values.
e. The END-OF-PAGE imperative is obeyed, if one was specified.

Note that 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.

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.

Using Standard 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. Cards 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 a $ASSIGN 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. They are ignored by the compiler.

Reading Cards Directly

To read cards directly from the card reader, there must be a $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 $DEFINE is optional. See the Job Control Language (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".

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.

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 Table 11-1.

The JCL and COBOL are the same as that shown in Table 11-1 except that $SYSOUT is mandatory in all cases shown in part 1 of the table (otherwise the file will be printed instead of punched).

The $SYSOUT statement used in parts 1 and 3 of Table 11-1 should specify a card punch device class. For example:

```
SYSOUT PUNCHER, DEVCLASS = CD/P, MEDIA = PUNCHOUT;
```

The $WRITER statement, used to punch the files as shown in parts 4 and 5 of Table 11-1, must also specify a card-punch device-class. For example:

```
WRITER C.PUNCHER, DEVCLASS = CD/P, MEDIA = PUNCHOUT;
```

As shown in parts 1,3 or 4 of Table 11-1, 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 handle such SYSOUT files, which are to be punched but not printed, in one of the following ways:

- As shown in part 5 of Table 11-1.
- As shown in parts 3 or 4 of Table 11-1 provided that the permanent SYSOUT file has a record size of less than 600 bytes (it has to be at least 600 bytes for the file to be written in edited SYSOUT format).

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 Table 11-1 can be simplified as shown in Table 11-2.

Table 11-2 Methods of Producing SYSOUT Punch Files

TYPE OF SYSOUT FILE CREATED (RECORD SIZE<600 BYTES)	JCL		COBOL
	$ASSIGN	$SYSOUT	-SYSOUT
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 $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 not punched by the Output Writer. A $WRITER statement must be given to punch the file.	YES	NO	NO

Punching Cards Directly

To punch cards directly on the card punch, there must be a $ASSIGN 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 $DEFINE is optional. See the Job Control Language (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".

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 which enables the program to continue. The use of these statements is described in the following paragraphs.

The ACCEPT Statement

The format of the ACCEPT statement to be dicussed is as follows:

```
ACCEPT identifier [FROM mnemonic-name]
```

The standard options DATE, DAY, 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 Language Reference Manual). The options SYSIN, CONSOLE, TERMINAL and ALTERNATE CONSOLE 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               }
{ CONSOLE             }
{ ALTERNATE CONSOLE   }  IS mnenonic-name
{ TERMINAL            }
```

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. However, CONSOLE is assumed if SOURCE-COMPUTER is LEVEL-62 and if the LEVEL = L62 parameter is used in the $COBOL statement. These defaults can be overridden by using the ACCEPT IS phrase in the DEFAULT SECTION (not part of the ANS standard).

If mnemonic-name specifies SYSIN, a special $ASSIGN 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 catalogued sequential file.

When data is being accepted with a mnemonic-name SYSIN, as many records as necessary are read to fill up the receiving item. The last such record is truncated if necessary and the truncated bytes are lost. However, if the first record in the SYSIN file for a given ACCEPT begins with an ampersand (&) and is followed by 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, no $ASSIGN 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 which 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 which occurs on the console will be echoed in the Job Occurrence Report prefixed by a report code "CBL13".

If mnemonic-name specifies ALTERNATE CONSOLE, data will be accepted from the alternate operator's console specified in the $CONSOLE statement (see the Job Control Language (JCL) Reference Manual). If no $CONSOLE 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 Job Occurrence Report:

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 the ACCEPT will behave as if mnemonic-name specified ALTERNATE CONSOLE. However, in a future release of the COBOL compiler, it is intended to implement the following. If mnemonic-name specifies TERMINAL and the load module is interactively executed from a terminal under the Interactive Operation Facility, data will be accepted from the terminal being used. If the load module is not executed from a terminal but is executed as a batch job, the ACCEPT will behave as if mnemonic-name specifies ALTERNATE CONSOLE.

Note that 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.

The DISPLAY Statement

The format of the DISPLAY statement is as follows:

```
DISPLAY  {identifier-1}  [,identifier-2]
         {literal-1   }  [,literal-2   ] ... [UPON mnemonic-name]
```

The options SYSOUT, CONSOLE, TERMINAL and ALTERNATE CONSOLE 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 SYSOUT, CONSOLE, TERMINAL or ALTERNATE CONSOLE.

Mnemonic-name can be defined in the SPECIAL-NAMES paragraph of the ENVIRONMENT DIVISION as follows:

```
{ SYSOUT                    }
{ CONSOLE                   }   IS mnemonic-name
{ ALTERNATE CONSOLE         }
{ TERMINAL                  }
```

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. However, CONSOLE is assumed if SOURCE-COMPUTER IS LEVEL-62 and if the LEVEL= L62 parameter is used in the $COBOL statement. These defaults can be overriden by using the DISPLAY IS phrase 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 statement is needed when the program is executed . However, if the output file is a permanent SYSOUT file a $ASSIGN 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=DISPOUT,FILESTAT=UNCAT;
```

In this example OUTFILE is an uncatalogued 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 no $ASSIGN 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 which is made on the console will be echoed in the Job Occurrence Report prefixed by the report code "CBL11".

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 statement when the program is executed, the hexadecimal value of this character is printed on the two lines below the erroneous character.

If mnemonic-name specifies ALTERNATE CONSOLE, data will be displayed on the alternate operator's console specified in the $CONSOLE statement (see the Job Control Language (JCL) Reference Manual). If no $CONSOLE statement is used data will be displayed on the console which submitted the program. If the submitting console is no longer logged the message is stored in the users mail box. The format of the console dialog is the same as that on the main console.

If mnemonic-name specifies TERMINAL, the DISPLAY will behave as if mnemonic-name specified ALTERNATE CONSOLE. However, in a future release of the COBOL compiler it is intended to implement the following. If mnemonic-name specifies TERMINAL and the load module is interactively executed from a terminal under the Interactive Operation Facility, data will be displayed 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.

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 Table 11-3.

Table 11-3. Variables Governing the Selection of I/O Devices

Device specified by mnemonic-name in ACCEPT or DISPLAY	Is there a $CONSOLE statement in the job step?	Type of Job submission		
		Batch	ROF	IOF
CONSOLE	No	M	M	M
	Yes	M	M	M
ALTERNATE CONSOLE	No	M	T(1)	T
	Yes	user-name	user-name	user-name
TERMINAL	No	M	T	T
	Yes	user-name	user-name	user-name

Notes for Table 11-3:

M - The main system console is used for I/O;

User-name - The terminal identified by user-name in the $CONSOLE statement is used for I/O;

T - The terminal which submitted the program for execution is used for I/O;

(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.

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 which 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 Job Occurrence Report 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.

*

USING CASSETTES

In a COBOL program, cassette files are handled as UFAS or BFAS tape files. See the UFAS User Guide or BFAS User Guide for details. Some additional rules for cassette files are given below.

Types of Cassette File

The following types of cassette file can be handled by a COBOL program:

- GCOS 64 standard cassette file;
- GCOS 62 standard cassette file;
- Foreign cassette file.

Cassette files can contain standard or nonstandard labels. Such cassettes may be processed as GCOS 64 or GCOS 62 standard cassette files or they may be processed as foreign cassette files.

When a cassette with nonstandard labels is read as a standard cassette file the LABEL = NSTD or LABEL = NONE parameter must be used in the $ASSIGN statement. Any labels on the cassette will be passed to the COBOL program as normal data records. An "AT END" condition will be generated when the first tape mark following the first block on the cassette is read. Note that a cassette file may be opened in I-O mode only if the LABEL = NATIVE parameter is used.

A $ASSIGN statement for each cassette file must be included in the $STEP JCL. DEVCLASS = CS must be specified in each such statement. Note that no repositioning of cassettes will be carried out by the system during a restart.

GCOS 64 Standard Cassette File

A GCOS 64 standard cassette file is created under a GCOS 64 system. The FILEFORM = NSTD parameter must not be used in the $DEFINE statement for this type of file. No distinction is made between UFAS and BFAS cassette files.

The cassette volume may be prepared by the $VOLPREP utility (see Data Management Utilities Manual) before a GCOS 64 standard cassette file is written. $VOLPREP will write a volume label on the cassette. In this case, the file will also be written with labels. A GCOS 64 standard cassette file may also be created without labels. This is the case if $VOLPREP is not used or if the LABEL = NONE parameter is used with $VOLPREP.

GCOS 64 standard cassette files may reside on one or more volumes and may be written with native or compact labels. The type of label is specified in the LABEL parameter of the $ASSIGN statement. When a GCOS 64 standard NATIVE labelled cassette file is written, the RECFORM, RECSIZE, BLKSIZE and NBSN (if required) parameters are recorded in the label. Consequently the user does not have to specify these parameters when the file is being read (INPUT or I-O mode). These parameters will be ignored by the system if they are specified. However, compact labels do not contain the RECFORM, RECSIZE, BLKSIZE and NBSN parameters. Therefore, these parameter must be supplied in the COBOL program FD (RECORD CONTAINS, BLOCK CONTAINS) or in a $DEFINE statement. Note that for compact standard labelled cassette files, RECFORM can only be F, FB or U.

GCOS 64 standard cassette files must have a minimum BLKSIZE of 2 bytes. The maximum value for BLKSIZE when writing to a cassette labelled NATIVE or COMPACT is 800 bytes including block header and BSN (if any).

GCOS 62 Standard Cassette File

A GCOS 62 standard cassette file is created under a GCOS 62 system using the sequential GCOS 62 access method.

The RECFORM, RECSIZE, BLKSIZE and NBSN (if required) parameters must be specified for GCOS 62 standard cassette files in the COBOL program's FD statement or in a $DEFINE statement.

Foreign Cassette Files

A foreign cassette file is a cassette file created under any system other than GCOS 64 or a cassette file created under GCOS 64 with the FILEFORM = NSTD parameter specified in the $DEFINE statement. The only exception is a GCOS 62 standard cassette file (see above).

A foreign cassette file has a data structure which cannot be accessed by the standard access methods of a GCOS 64 system. This situation results from one or more of the following:

- Nonstandard labels exist on the file;
- Tape marks are embedded between data blocks;
- Block and record structure is nonstandard;
- Recording mode is nonstandard (pack, depack, datacode).

When a foreign cassette file is read by a COBOL program, either a data block or a label block or a tape mark is handed to the program each time a READ statement is encountered. Similarly, each time a WRITE statement is encountered, either a data block or a label block or a tape mark will be written from the COBOL program to the cassette. End of file will not be detected by the system on input and the AT END clause will not be obeyed. When a foreign cassette file is opened, the cassette is positioned at the beginning of the tape and it is the user's responsibility to read or write labels, data and tape marks.

After each read operation, the identifier specified in the DEPENDING ON option of the RECORD CONTAINS clause will contain the length of the block read (in bytes). A tape mark or a long gap will have a length of one byte. A tape mark will appear as FF (hexadecimal) in the first byte of the record area. A long gap will appear as 00 (hexadecimal) in the first byte of the record area.

Before each write operation the length of the block to be written must be specified (in bytes) in the identifier specified in the DEPENDING ON option. A tape mark must be coded as FF (hexadecimal) in the first byte of the record area and must be given a length of one byte.

The maximum blocksize may be specified in the COBOL program in the BLOCK CONTAINS clause. All other parameters must be specified in the JCL. The BLKSIZE parameter must be specified in the COBOL program's FD statement or in a $DEFINE statement. A foreign cassette may have standard labels. If this is not the case, the LABEL = NONE parameter must be specified in the $ASSIGN statement. A $DEFINE statement with the parameter FILEFORM = NSTD is mandatory for foreign cassette files. The maximum value for BLKSIZE when writing to a cassette labelled NSTD or NONE is 256 bytes including block header and BSN (if any).

SECTION XII

MISCELLANEOUS

This section includes various topics which are not discussed at length in this manual and therefore do not warrant individual sections.

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 $SORT and $MERGE see the Sort/Merge Manual.

By using the OUTPUT command in $SORT a sequential file of disk addresses can be output. This file can later be read by a COBOL program and can be used to access the file that was input to $SORT. This is discussed in The ACTUAL KEY Phrase, Section IX.

Comparison of COBOL SORT/MERGE and $SORT/$MERGE

The choice between using the COBOL SORT/MERGE statements and executing $SORT/$MERGE as separate utilities is basicaly the choice between flexibility and performance. The following points amplify this.

- Execution of $SORT as a utility rather than as a COBOL statement saves up to half the central processor time. However, the MERGE statement in COBOL executes faster than the corresponding $MERGE utility.

- The commands available with $SORT/$MERGE are not as powerful and flexible as COBOL statements.

- 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).

Therefore, $SORT 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 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 $SORT do not have to be written or read.

The above rules for using SORT and $SORT can be applied to MERGE and $MERGE except that processing of input files (INPUT PROCEDURE) is not possible when merging.

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.

In the SELECT clause for the sort file in the COBOL program, the internal-file-name H-SORT is usually used. Note that the ORGANIZATION, ACCESS MODE, RESERVE, FILE STATUS and RECORD KEY phrases cannot be used in the SELECT clause for a sort file. This file may be assigned in the JCL to the external file name H_SRTWKD using the $SORTWORK statement in the same job step. It is used as a work file by the sorting routines. The format of the $SORTWORK statement is as follows:

```
           / WKTAPE[S] = (NBDV=n,DEVCLASS=device-class)                        \
           |                                                                   |
           |               {external-file-name}                                |
           | WKDISK[S] = ({SIZE = nnn         }                                |
           |                                                                   |
SORTWORK  <                 [                  {CAT  }]                         >
           |                [,FILESTAT =       {UNCAT}]                        |
           |                [,CATALOG = n]                                     |
           |             [  / RESIDENT                              \ ]        |
           |             [ ,< DEVCLASS= device-class                 >]  )     |
           \             [  \ [,MEDIA =(  {WORK                }) ] / ]        /
                                          {volume-name[,...]   }
```

The $SORTWORK statement can be used to assign the sort file to tape(s) or disk(s) but not to both.

If the sort is to be tape based the WKTAPE[S] keyword should be used. The number of devices to be assigned is specified in the NBDV = n parameter. A minimum of three devices and a maximum of six may be used. Note that a tape sort normally has a much longer elapsed execution time that a disk sort. However, for input files that have relatively few records out of sequence, a tape sort usually has an elapsed time close to that of a disk sort. Moreover, elapsed time reduces with any increase in the number of devices used, in the record-blocking factor, and in the recording density.

If the sort is to be disk based the WKDISK[S] keyword should be used. The parameters which can be used with the WKDISK[S] keyword have the same significance as in the $ASSIGN and $ALLOCATE statements.

If the $SORTWORK parameter is not used, a temporary file of 10 cylinders will be allocated on a resident volume, with the name H_SRTWKD. In a multiprogramming environment the use of a resident disk for the work file can cause a considerable increase in arm movement. It is therefore preferable to preallocate a permanent file (of sufficient size, but on a single volume) on a disk other than that used for input and output files, and, if possible, on a disk not used concurrently by another job step. See the Sort/Merge Manual, Appendix D, for more information about $SORTWORK.

The size of the declared working set can be changed for a COBOL SORT by the keyword SORTMEMORY in the $STEP statement OPTIONS string. By increasing this value (which varies from 8K to 512K, with a default value of 28K), elapsed time is reduced. The format of the OPTIONS parameter containing SORTMEMORY is:

```
OPTIONS = '...SORTMEMORY = nnn...'
```

Where nnn is the number of bytes in units of 1024. User options may accompany the SORTMEMORY option (see below, $STEP OPTIONS). If the options string is passed to the user program, the SORTMEMORY option is passed to the program with any other options which are used.

USER JCL STATUS

The system sets a status value, which can be used by $JUMP in the event of an abnormal step termination (STATUS = 60000), or an operator-requested end of step (STATUS=50000). The COBOL compiler also sets the status value at the end of compilation, according to errors detected (see Sections II, The Compiler).

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 32768.

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=ALL;
     ENDSTEP;
     JUMP LAB1,STATUS,EQ,64;
     SEND  'STATUS DIFFERENT FROM 64';
     COMMENT  'STATUS DIFFERENT FROM 64';
     JUMP LAB2;
     LAB1: SEND 'STATUS = 64';
     COMMENT  'STATUS = 64';
     LAB2: SEND 'END OF TEST';
     COMMENT  'END OF TEST';
     $ENDJOB;
```

The Job Occurrence Report will then show:

```
    PROCESS GROUP TERMINATED STATUS = 64
```

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 SPECIAL-NAMES, 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 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".
          .
          .
```

CHECKPOINT, RESTART AND JOURNALIZATION

The RERUN clause in the I-O-CONTROL paragraph allows the user to specify the frequency with which checkpoints are to be taken during program execution, in terms of the number of records read or written in a specified file. This value is communicated to Data Management, which decrements the value by one for each record processed. When the value reaches zero, a special return code is sent to the COBOL run-time package.

The COBOL run-time package then calls the system procedure to perform the checkpoint. 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 at other times (e.g. at the end of each tape or disk volume). See the $DEFINE statement in the Job Control Language 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 temination, which also appears in the Job Occurrence Report.

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 system other than Level 64.

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.

Details of the use of the above facilities are given in the System Management Guide.

ALPHABETS

The following COBOL Language elements are discussed below:

- In the SPECIAL-NAMES paragraph

alphabet-name <u>IS</u> { <u>STANDARD-1</u> | <u>NATIVE</u> | <u>ASCII</u> | <u>EBCDIC</u> | <u>HBCD</u> | <u>IBCD</u> | <u>JIS</u> | <u>GBCD</u> | user-specified-alphabet }

- In the OBJECT-COMPUTER paragraph

```
PROGRAM COLLATING SEQUENCE IS { alphabet-name
                                STANDARD-1
                                NATIVE
                                ASCII
                                EBCDIC
                                HBCD
                                IBCD
                                JIS
                                GBCD }
```

- In the FILE SECTION

```
CODE-SET IS { alphabetic-name
              STANDARD-1
              NATIVE
              ASCII
              EBCDIC
              HBCD
              IBCD
              JIS
              GBCD }
```

- in the SORT and MERGE statements

```
COLLATING SEQUENCE IS { alphabet-name
                        STANDARD-1
                        NATIVE
                        ASCII
                        EBCDIC
                        HBCD
                        IBCD
                        JIS
                        GBCD }
```

Note that ASCII, EBCDIC, HBCD, IBCD, 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 Language Reference Manual for an explanation of STANDARD-1, NATIVE, ASCII, EBCDIC, HBCD, IBCD, 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 Language 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.

PROGRAM COLLATING SEQUENCE

PROGRAM COLLATING SEQUENCE in the OBJECT-COMPUTER paragraph indicates the collating sequence to be used for non-numeric comparisons of the following type:

```
{identifier-1             } {IS [NOT] GREATER THAN} {identifier-2            }
{literal-1                } {IS [NOT] LESS THAN   } {literal-2               }
{arithmetic-expression-1  } {IS [NOT] >           } {arithmetic-expression-2 }
                            {IS [NOT] <           }
                            {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 no effect on non-numeric comparisons of the following type:

```
{identifier-1             } {IS [NOT] EQUAL TO} {identifier-2            }
{literal-1                } {IS [NOT] =       } {literal-2               }
{arithmetic-expression-1  } {IS UNEQUAL TO    } {arithmetic-expression-2 }
                            {EQUALS           }
```

These comparisons are made without prior conversion.

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.

CODE-SET

The CODE-SET clause enables data to be input from or output to files in code sets other than NATIVE (EBCDIC). This facility can be used only with sequential files and, if CODE-SET is equated to HBCD, for non-sequential H-2000 files. 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 when code conversion is done by hardware (e.g., for cards or ANS magnetic tape).

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 in Table 12-1.

Table 12-1. High Values and Low Values

COLLATING SEQUENCE	HIGH-VALUE	LOW-VALUE
STANDARD-1 (ASCII)	""256""	""1""
NATIVE (EBCDIC)	""256""	""1""
HBCD	"¢"	"0"
IBCD	"9"	" "
JIS	""256""	""1""
GBCD	"!"	"0"

Notes for Table 12-1:

"0" is zero

"9" is nine

"¢" is cent

"!" is exclamation mark

" " is blank

""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.

$STEP OPTIONS

The OPTIONS parameter of the $STEP 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 $LINKER.) The way in which the COBOL program should be written is shown in the following example.

```
.
.
WORKING-STORAGE SECTION.
01 ID1          PIC 999.
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 LONG COMP-2.
01 TEXT.
     02 ELEM PIC X OCCURS 1 TO 256 DEPENDING ON LONG.
.
.
PROCEDURE DIVISION USING LONG TEXT.
DEBUT.
      MOVE SPACE TO OPT1 OPT2 OPT3 OPT4 OPT5.
      UNSTRING TEXT 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 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
```

If the SORTMEMORY option (used with the COBOL SORT statement) is present it is passed to the user program with the user options.

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 Language Reference Manual. See also The Report Writer in Section VI 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.

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, Section XI.

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.

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.

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. 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 LABEL RECORD IS STANDARD
                   RECORD CONTAINS 121 CHARACTERS
                   REPORT IS REPORT-A.
     FD     FILE-2 LABEL RECORD IS STANDARD
                   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.

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.

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.

```
          .
     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  DAYE TYPE CONTROL FOOTING LINE PLUS 2.
          .
          .
         05  SUBTOTAL COLUMN 30 PIC 999
             SUM NO-PURCHASES UPON DETAIL-1.
          .
          .
     01  MONTH TYPE CONTROL FOOTING
         LINE PLUS 2 NEXT GROUP NEXT PAGE.
          .
```

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 D 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 neccessary 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. Figure 12-1 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...
```

Figure 12-1. Sample GROUP INDICATE Clause

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 Section XI) 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 statement specifying the desired code (see the Job Control Language (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
    LABEL RECORD STANDARD 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 statement could then be used to print only the report with code "AA", as follows.

```
WRITER (report-file-description), REPORT=AA, DATAFORM=SSF;
```

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 LAST DETAIL integer to a sufficiently low line number) to allow for the necessary space.

Note also the following example.

```
        .
        .
    RD  EXPENSE-REPORT CONTROLS ARE LAST, 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.

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 NEXT GROUP NEXT PAGE.

  then the following body group

    01 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 NEXT GROUP LINE 12.

  and after it was printed the value of LINE-COUNTER was 40, then the body group

  01 TYPE DETAIL LINE PLUS 5.

  would be printed on line 12 + 1 (i.e., line 13).

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.

TABLE HANDLING

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 Section VIII 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.

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).
```

Figure 12-2 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.                INCORRECT EXAMPLE
SET A2, A1 TO INDX-DATA-ITM.            correct version
SET A3 UP BY 1.                         shown below
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 in Figure 12-3.

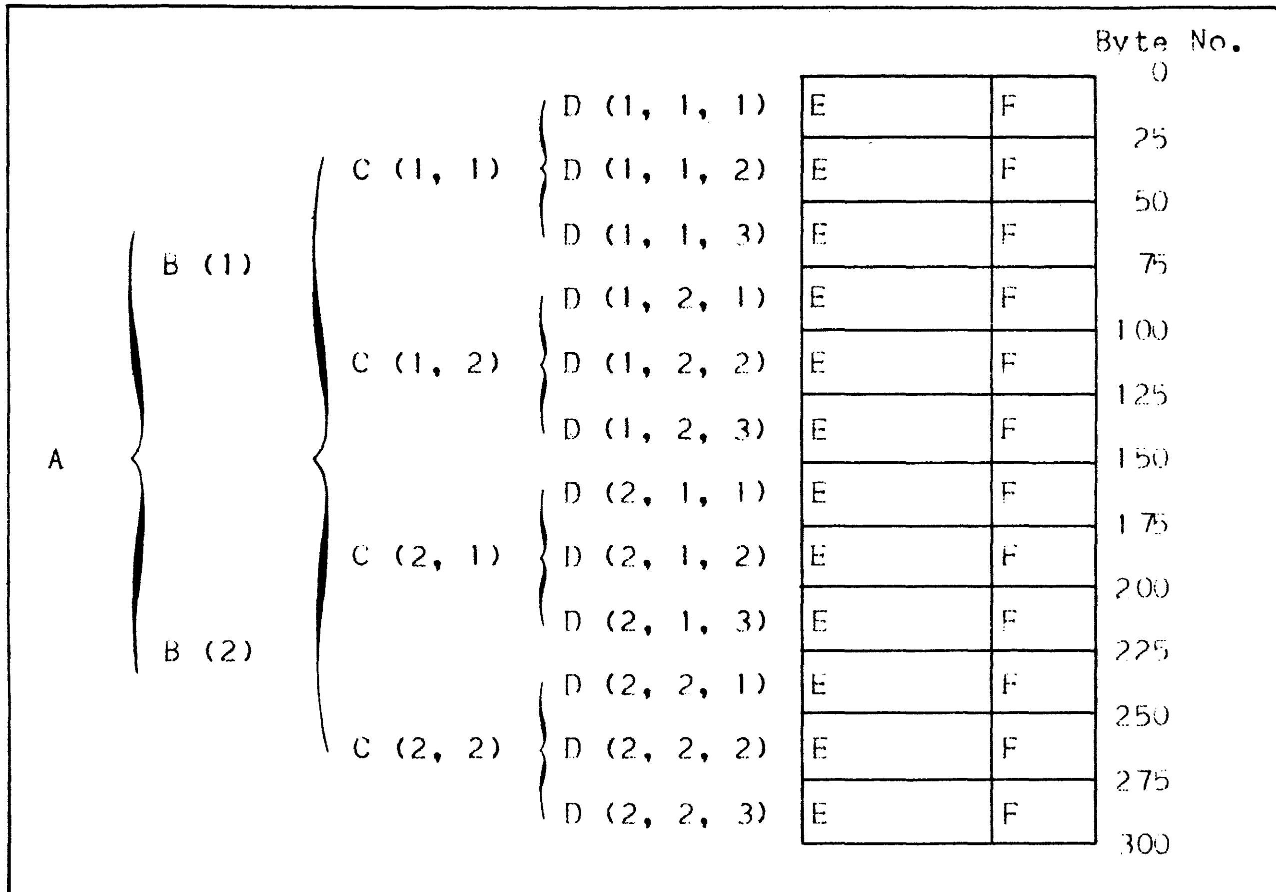


Figure 12-2. Sample Table Layout in Memory

Sending / Receiving	Index-name	Index Data Item	Identifier or Literal
Index-name	Set to value correcponding 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.			

Figure 12-3. Rules for the SET Statement

The SEARCH Statement

Only one level of a table (a table element) can be referenced in one SEARCH statement. Note that SEARCH statements cannot be nested: an imperative statement must follow the WHEN condition and the SEARCH statement is itself conditional.

The SEARCH statement has two formats.

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.

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:

```
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 SEARCH-TEST1 THRU SEARCH-EXIT1
        VARYING A1 FROM 1 BY 1 UNTIL A1 GREATER THAN 2
        AFTER A2 FROM 1 BY 1 UNTIL A2 GREATER THAN 2.
ENTRY-NOENTRY1.
    GO TO ERROR-RECOVERY1.
                .
                .
SEARCH-TEST1.
    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
             ALTER ENTRY-NOENTRY1 TO PROCEED TO ENTRY-PROCESSING1.
SEARCH-EXIT1.
    EXIT.
                .
                .
ERROR-RECOVERY1.
                .
                .
ENTRY-PROCESSING1.
    MOVE E (A5, A6, A3) TO OUT-AREA1.
    MOVE F (A5, A6, A3) TO OUT-AREA2.
                .
                .
```

The PERFORM statement varies 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.

Building Tables

When reading in data to build an internal table the following points should be born 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 of the CONTROL DIVISION. The REPLACE statement is not part of ANS standard COBOL.

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 (31 if the LEVEL = L64 parameter is included in the COBOL statement).

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.

id - 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.

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, 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 dmax, whichever is greater.
**	dmax if op2 is non integral or a data-name; d * op2 if op2 is an integral literal.

If the number of digits in ir is greater than 31, decimal floating-point format will be used.

Table 12-2 indicates the length allocated to ir based upon the values calculated for ip and dp.

Table 12-2. Length of Intermediate Result Fields

Value of ip + dp	Value of ip + dmax	Length allocated for ir
< 32	Any value	ip integer places and dp decimal places are allocated for ir.
> 31	< 32	ip integer places and 31 - ip decimal places are allocated for ir.
	> 31	decimal floating-point format is used.

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.

COBOL Run-Time Package

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. The most significant 31 decimal digits of the result of this multiplication are kept.

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 31.

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 31 if LEVEL = L64 is specified in the $COBOL statement, otherwise the default value is 18.

The ON SIZE ERROR Phrase

Apart from division by zero, 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.

COMMUNICATIONS 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

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 in Table 12-3.

Table 12-3. 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.	21BA2AB	same	3	-	-
EXAMINE X TALLYING ALL 1.	21BA2AB	same	1	-	-
EXAMINE X REPLACING FIRST B BY C.	21BA2AB	21CA2AB	same	-	-
EXAMINE X REPLACING LEADING 2 BY3.	21CA2AB	31CA2AB	same	-	-
INSPECT X TALLYING 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 Language Reference Manual.

APPENDIX A

EXAMPLE COBOL PROGRAM

```
*****************************************************************************************************
*****************************************************************************************************
**** GCOS L64                                                                                    ****
****                                    C  O  B  O  L                                            ****
****                                                              VERSION: 50     DATED: MAR 10, 1978 ****
***************************************************************************2     ********************
*****************************************************************************************************

PROGRAM: FIND-DAY

USER: BOURGAIN

PROJECT: BOURGAIN

DATE: 03/31/78

TIME: 13:22:48

COMPILER VERSION: L64 COBOL V-50.2

USER OPTIONS: COMFILE LIB=1 LEVEL=L64 DCLXREF XREF EXPLIST

ACTIVE OPTIONS: OBJ, NDEBUG, WARN, OBSERV, NMAP, DCLXREF, XREF, LIST, EXPLIST, CKSEQ, CARDID, CASEQ, DIAGIN,
        NCODAPND, NOPT, DDEBUGMD, PSEGMAX=4096(BYTES), DSEGMAX=4096(BYTES).

COMPILATION LEVEL: L64

COMPILER INPUT:
  ALTER FILE
    RSTR (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 RSTR (H_INLIB1)
    CD=01/23/78 CT=10:35:24 MD=03/07/78 MT=13:16:12 SL=DAT MN=11 NM=FIND-DAY
  COPY FILE (COPIED TEXT ON LINES 38 THROUGH 49)
    DAYS IN RSTR (H_INLIB1)
    CD=01/23/78 CT=10:35:24 MD=01/23/78 MT=10:35:24 SL=DAT MN=00 NM=DAYS
```

```
  A.1 ----->        COMPILE;
  A.2
  A.3            R: R FIND-DAY
  A.4
  A.5            R: /DATA/S/DIVISION/&./
  A.6
  A.7            R: /01  DITWEEK-TAB/,/SUNDAY/C               COMMENT
                                              1
* 1 1-44  TEXT FOLLOWS THE 'A', 'C', 'I' CR 'Q' COMMAND ON THE LINE. TEXT IS IGNORED.

  A.8            I:            COPY DAYS
  A.9            I:              REPLACING == PIC X(8) == BY == PIC X(10) ==.
 A.10            I: ¢F
```

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

     S.1         1         IDENTIFICATION DIVISION.
     S.2         2        *                        THIS ROUTINE, STARTING FROM A DATE, GIVES        <-
     S.3         3        *                        THE DAY IN THE WEEK CORRESPONDING TO THE
     S.4         4        *                        DATE
     S.5         5         PROGRAM-ID. FIND-DAY.
     S.6         6        *
     S.7         7         ENVIRONMENT DIVISION.
     S.8         8         CONFIGURATION SECTION.
     S.9         9         SOURCE-COMPUTER. LEVEL-64
    S.10        10         OBJECT-COMPUTER. LEVEL-64.
    S.11        11        *
    S.12       *12         DATA DIVISION.                                                             <-
    S.13        13        *
    S.14        14         WORKING-STORAGE SECTION.
    S.15        15        *                        TEMPORARIES
    S.16        16         01  X PICTURE 9(10).
    S.17        17         01  Y PICTURE 9(5).
    S.18        18        *                        TOTAL NUMBER OF DAYS PRECEDING THE MONTH
    S.19        19        *                        (SHOWN BY ITS ORDINAL NUMBER IN THE LIST)
    S.20        20        *                        IN THE YEAR
    S.21        21         01  PREC-D-TAB.
    S.22        22             02  FILLER PIC 999 VALUE   0.
    S.23        23             02  FILLER PIC 999 VALUE  31.
    S.24        24             02  FILLER PIC 999 VALUE  59.
    S.25        25             02  FILLER PIC 999 VALUE  90.
    S.26        26             02  FILLER PIC 999 VALUE 120.
    S.27        27             02  FILLER PIC 999 VALUE 151.
    S.28        28             02  FILLER PIC 999 VALUE 181.
    S.29        29             02  FILLER PIC 999 VALUE 212.
    S.30        30             02  FILLER PIC 999 VALUE 243.
    S.31        31             02  FILLER PIC 999 VALUE 273.
    S.32        32             02  FILLER PIC 999 VALUE 304.
    S.33        33             02  FILLER PIC 999 VALUE 334.
    S.34        34         01  PREC-D-TAB-REI REDEFINES PREC-D-TAB.
    S.35        35             02  PRECEDING-DAYS PIC 999 OCCURS 12.
    S.36        36        *                        TABLE GIVING THE NAME OF THE DAYS IN THE
    S.37        37        *                        WEEK
    S.38         .             COPY DAYS
    S.39         .               REPLACING == PIC X(8) == BY == PIC X(10) ==.
    S.40       ..1         01  OTHER-UNUSED.
    S.41       ..2             02 FILLER PIC X.
    S.42       ..3             02 FILLER COMP-1 SYNC.
    S.43       ..4         01  DITWEEK-TAB.
    S.44       ..5             02  FILLER PIC X(8)  VALUE "LUNDI   ".

                                  1   2
  * 1 1-32  FIRST WORD OF TEXT REPLACED (OR DELETED).
  * 2 1-33  LAST WORD OF TEXT REPLACED (OR DELETED).

    S.45       ..6             02  FILLER PIC X(10) VALUE "MARDI   ".
    S.46       ..7             02  FILLER PIC X(10) VALUE "MERCREDI".
    S.47       ..8             02  FILLER PIC X(10) VALUE "JEUDI   ".
    S.48       ..9             02  FILLER PIC X(10) VALUE "VENDREDI".
```

```
  ILN        XLN  TEXT  7-10--------20--------30--------40--------50--------60--------70--........<-

 S.50        ..11             02  FILLER PIC X(10) VALUE "DIMANCHE".
 S.51         -46        01  DITWEEK-TAB-RID REDEFINES DITWEEK-TAB.
 S.52          47             02  DAY-IN-THE-WEEK PIC X(10) OCCURS 7 TIMES.
 S.53          48        *                          AREA FOR DATE SPLITTING INTO YEAR, MONTH,
 S.54          49        *                          AND DAY
 S.55          50        01  SPLIT-DATE.
 S.56          51             02  CENTURY         PIC 99.
 S.57          52             02  SHORT-DATE.
 S.58          53             03  FILLER          PIC 99.
 S.59          54             03  MONTH           PIC 99.
 S.60          55             03  DAY-OF-MONTH    PIC 99.
 S.61          56             03  DAY-OF-MONTH-X REDEFINES DAY-OF-MONTH PIC XX.
 S.62          57        01 YEAR REDEFINES SPLIT-DATE PIC 9(4).
 S.63          58        *                          ORDINAL NUMBER OF THE DAY TAKEN INTO
 S.64          59        *                          CONSIDERATION WITHIN THE DAYS OF THE
 S.65          60        *                          CHRISTIAN ERA
 S.66          61        01  DAYS-IN-THE-ERA PIC 9(10).
 S.67          62        *
 S.68          63        LINKAGE SECTION.
 S.69          64        *                          DATE FOR WHICH THE DAY OF WEEK IS LOOKED
 S.70          65        *                          FOR, UNDER THE FORM YYYYMMDD OR YYMMDDBB
 S.71          66        *                          (WHERE B MEANS BLANK
 S.72          67        01  FULL-DATE PIC X(8).
 S.73          68        *                          RETURNED ORDINAL NUMBER OF THE DAY IN THE
 S.74          69        *                          WEEK (1 IS MONDAY, 2 TUESDAY ... )
 S.75          70        01  DAY-OF-THE-WEEK PIC 9.
 S.76          71        *                          RETURNED DAY IN THE WEEK ITSELF
 S.77          72        01  DAY-ITSELF PIC X(10).
```

```
     ILN        XLN  TEXT  7-10--------20--------30--------40--------50--------60--------70--........<-

    S.78         73        /
    S.79         74         PROCEDURE DIVISION USING FULL-DATE DAY-OF-THE-WEEK DAY-ITSELF.
    S.80         75        *
    S.81         76         BEGIN.
    S.82         77             MOVE FULL-DATE TO SPLIT-DATE.
    S.83         78             IF DAY-OF-MONTH-X = SPACE
    S.84         79               MOVE SPLIT-DATE TO SHORT-DATE
    S.85         80               MOVE 19 TO CENTURY.
    S.86         81        *                       LET US COMPUTE THE NUMBER OF DAYS SPENT
    S.87         82        *                       SINCE THE BEGINNING OF THE CHRISTIAN ERA
    S.88         83             COMPUTE DAYS-IN-THE-ERA =
    S.89         84                          DAY-OF-MONTH
    S.90         85                        + PRECEDING-DAYS (MONTH)
    S.91         86                        + (YEAR - 1) * 365.
    S.92         87        *                       LET US ADD 1 FOR EACH LEAP-YEAR, INCLUDING
    S.93         88        *                       THE YEAR OF THE PROCESSED DATE IF THE
    S.94         89        *                       MONTH IS LATER THAN FEBRUARY
    S.95         90             IF MONTH < 3 COMPUTE YEAR = YEAR - 1.
    S.96         91             DIVIDE YEAR BY 4 GIVING X.
    S.97         92             ADD X TO DAYS-IN-THE-ERA.
    S.98         93             DIVIDE YEAR BY 100 GIVING X.
    S.99         94             SUBTRACT X FROM DAYS-IN-THE-ERA.
   S.100         95             DIVIDE YEAR BY 1000 GIVING X.
   S.101         96             ADD X TO DAYS-IN-THE-ERA.
   S.102         97        *                       NOW THE REMAINDER OF THE DIVISION BY 7 OF
   S.103         98        *                       THE DAYS-IN-THE-ERA, AUGMENTED OF THE
   S.104         99        *                       PROPER CONSTANT, IS THE ORDINAL NUMBER OF
   S.105        100        *                       THE DAY IN THE WEEK
   S.106        101             DIVIDE DAYS-IN-THE-ERA BY 7 GIVING X REMAINDER Y.
   S.107        102             IF Y > 4
   S.108        103               SUBTRACT 4 FROM Y
   S.109        104             ELSE
   S.110        105               ADD 3 TO Y.
   S.111        106             MOVE Y TO DAY-OF-THE-WEEK.
   S.112        107             MOVE DAY-IN-THE-WEEK (Y) TO DAY-ITSELF.
   S.113        108         THE-END.
   S.114        109             EXIT PROGRAM.
```

```
 ILN       XLN  TEXT  7-10--------20--------30--------40--------50--------60--------70--........<-

   1         1         IDENTIFICATION DIVISION.
   2         2        *                          THIS ROUTINE, STARTING FROM A DATE, GIVES        <-
   3         3        *                          THE DAY IN THE WEEK CORRESPONDING TO THE
   4         4        *                          DATE
   5         5         PROGRAM-ID. FIND-DAY.
   6         6        *
   7         7         ENVIRONMENT DIVISION.
   8         8         CONFIGURATION SECTION.
   9         9         SOURCE-COMPUTER. LEVEL-64
  10        10         OBJECT-COMPUTER. LEVEL-64.
  11        11        *
  12       *12         DATA DIVISION.                                                             <-
  13        13        *
  14        14         WORKING-STORAGE SECTION.
  15        15        *                          TEMPORARIES
  16        16         01  X PICTURE 9(10).
  17        17         01  Y PICTURE 9(5).
  18        18        *                          TOTAL NUMBER OF DAYS PRECEDING THE MONTH
  19        19        *                          (SHOWN BY ITS ORDINAL NUMBER IN THE LIST)
  20        20        *                          IN THE YEAR
  21        21         01  PREC-D-TAB.
  22        22             02  FILLER PIC 999 VALUE   0.
  23        23             02  FILLER PIC 999 VALUE  31.
  24        24             02  FILLER PIC 999 VALUE  59.
  25        25             02  FILLER PIC 999 VALUE  90.
  26        26             02  FILLER PIC 999 VALUE 120.
  27        27             02  FILLER PIC 999 VALUE 151.
  28        28             02  FILLER PIC 999 VALUE 181.
  29        29             02  FILLER PIC 999 VALUE 212.
  30        30             02  FILLER PIC 999 VALUE 243.
  31        31             02  FILLER PIC 999 VALUE 273.
  32        32             02  FILLER PIC 999 VALUE 304.
  33        33             02  FILLER PIC 999 VALUE 334.
  34        34         01  PREC-D-TAB-RED REDEFINES PREC-D-TAB.
  35        35             02  PRECEDING-DAYS PIC 999 OCCURS 12.
  36        36        *                          TABLE GIVING THE NAME OF THE DAYS IN THE
  37        37        *                          WEEK
  38       ..1         01  OTHER-UNUSED.
  39       ..2             02 FILLER PIC X.
  40       ..3             02 FILLER COMP-1 SYNC.

                            1
 1 2-199 A 1 BYTE TYPE 2 FILLER WAS ALLOCATED TO ALIGN THIS SYNCHRONIZED ITEM (SEE REFERENCE MANUAL).

  41       ..4         01  DITWEEK-TAB.
  42      *..5             02  FILLER                            PIC X(10)         <-
  43      *..5                                     VALUE "LUNDI    ".
  44       ..6             02  FILLER PIC X(10) VALUE "MARDI    ".
  45       ..7             02  FILLER PIC X(10) VALUE "MERCREDI".
  46       ..8             02  FILLER PIC X(10) VALUE "JEUDI    ".
  47       ..9             02  FILLER PIC X(10) VALUE "VENDREDI".
  48       ..10            02  FILLER PIC X(10) VALUE "SAMEDI   ".
  49       ..11            02  FILLER PIC X(10) VALUE "DIMANCHE".
  50       -46         01  DITWEEK-TAB-RED REDEFINES DITWEEK-TAB.
```

```
      51        47               02  DAY-IN-THE-WEEK PIC X(10) OCCURS 7 TIMES.
      52        48          *                          AREA FOR DATE SPLITTING INTO YEAR, MONTH,
      53        49          *                          AND DAY
      54        50           01  SPLIT-DATE.
      55        51               02  CENTURY        PIC 99.
      56        52               02  SHORT-DATE.
      57        53               03  FILLER         PIC 99.
      58        54               03  MONTH          PIC 99.
      59        55               03  DAY-OF-MONTH   PIC 99.
      60        56               03  DAY-OF-MONTH-X REDEFINES DAY-OF-MONTH PIC XX.
      61        57           01 YEAR REDEFINES SPLIT-DATE PIC 9(4).
      62        58          *                          ORDINAL NUMBER OF THE DAY TAKEN INTO
      63        59          *                          CONSIDERATION WITHIN THE DAYS OF THE
      64        60          *                          CHRISTIAN ERA
      65        61           01  DAYS-IN-THE-ERA PIC 9(10).
      66        62          *
      67        63           LINKAGE SECTION.
      68        64          *                          DATE FOR WHICH THE DAY OF WEEK IS LOOKED
      69        65          *                          FOR, UNDER THE FORM YYYYMMDD OR YYMMDDBB
      70        66          *                          (WHERE B MEANS BLANK
      71        67           01  FULL-DATE PIC X(8).
      72        68          *                          RETURNED ORDINAL NUMBER OF THE DAY IN THE
      73        69          *                          WEEK (1 IS MONDAY, 2 TUESDAY ... )
      74        70           01  DAY-OF-THE-WEEK PIC 9.
      75        71          *                          RETURNED DAY IN THE WEEK ITSELF
      76        72           01  DAY-ITSELF PIC X(10).
```

```
    ILN       XLN  TEXT  7-10--------20--------30--------40--------50--------60--------70--........<-

     77        73        /
     78        74         PROCEDURE DIVISION USING FULL-DATE DAY-OF-THE-WEEK DAY-ITSELF.
     79        75        *
     80        76         BEGIN.
     81        77             MOVE FULL-DATE TO SPLIT-DATE.
     82        78             IF DAY-OF-MONTH-X = SPACE
     83        79               MOVE SPLIT-DATE TO SHORT-DATE
                                                    1
 ** 1 5-148 THIS RECEIVING ITEM MAY BE TRUNCATED ON 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.

     84        80               MOVE 19 TO CENTURY.
     85        81        *                          LET US COMPUTE THE NUMBER OF DAYS SPENT
     86        82        *                          SINCE THE BEGINNING OF THE CHRISTIAN ERA
     87        83             COMPUTE DAYS-IN-THE-ERA =
     88        84                            DAY-OF-MONTH
     89        85                          + PRECEDING-DAYS (MONTH)
     90        86                          + (YEAR - 1) * 365.
     91        87        *                          LET US ADD 1 FOR EACH LEAP-YEAR, INCLUDING
     92        88        *                          THE YEAR OF THE PROCESSED DATE IF THE
     93        89        *                          MONTH IS LATER THAN FEBRUARY
     94        90             IF MONTH < 3 COMPUTE YEAR = YEAR - 1.
     95        91             DIVIDE YEAR BY 4 GIVING X.
     96        92             ADD X TO DAYS-IN-THE-ERA.
     97        93             DIVIDE YEAR BY 100 GIVING X.
     98        94             SUBTRACT X FROM DAYS-IN-THE-ERA.
     99        95             DIVIDE YEAR BY 1000 GIVING X.
    100        96             ADD X TO DAYS-IN-THE-ERA.
    101        97        *                          NOW THE REMAINDER OF THE DIVISION BY 7 OF
    102        98        *                          THE DAYS-IN-THE-ERA, AUGMENTED OF THE
    103        99        *                          PROPER CONSTANT, IS THE ORDINAL NUMBER OF
    104       100        *                          THE DAY IN THE WEEK
    105       101             DIVIDE DAYS-IN-THE-ERA BY 7 GIVING X REMAINDER Y.
    106       102             IF Y > 4
    107       103               SUBTRACT 4 FROM Y
    108       104             ELSE
    109       105               ADD 3 TO Y.
    110       106             MOVE Y TO DAY-OF-THE-WEEK.
                                       1
 ** 1 5-156 POSSIBLE LEFT TRUNCATION

    111       107             MOVE DAY-IN-THE-WEEK (Y) TO DAY-ITSELF.
    112       108         THE-END.
    113       109             EXIT PROGRAM.
```

```
LN NAME                                    PN   ADDRESS   USAGE    PIC-STRING    DEF.  REF.  LINES

77 TALLY                                        1:00010   DISP     9(5)                NOREF

01 X                                            1:00064   DISP     9(10)         16    95 96 97 98 99 100 105

01 Y                                            1:00070   DISP     9(5)          17    105 106 107 109 110 111

01 PREC-D-TAB                                   1:00078   GROUP    X(36)         21    NOREF

01 PREC-D-TAB-RED                               1:00078   GROUP    X(36)         34    NOREF
02 PRECEDING-DAYS                               1:00078   DISP     9(3)          35    89

01 OTHER-UNUSED                                 1:000A0   GROUP    X(4)          38    NOREF

01 DITWEEK-TAB                                  1:000A4   GROUP    X(70)         41    NOREF

01 DITWEEK-TAB-RED                              1:000A4   GROUP    X(70)         50    NOREF
02 DAY-IN-THE-WEEK                              1:000A4   DISP     X(10)         51    111

01 SPLIT-DATE                                   1:000F0   GROUP    X(8)          54    81 83
02 CENTURY                                      1:000F0   DISP     9(2)          55    84
02 SHORT-DATE                                   1:000F2   GROUP    X(6)          56    83
03 MONTHR                                       1:000F4   DISP     9(2)          58    89 94
03 DAY-OF-MONTH                                 1:000F6   DISP     9(2)          59    88
03 DAY-OF-MONTH-X                               1:000F6   DISP     X(2)          60    82

01 YEAR                                         1:000F0   DISP     9(4)          61    90 94+ 95 97 99

01 DAYS-IN-THE-ERA                              1:000F8   DISP     9(10)         65    87 96 98 100 105

01 FULL-DATE                               1      00000   DISP     X(8)          71    78 81

01 DAY-OF-THE-WEEK                         2      00000   DISP     9(1)          74    78 110

01 DAY-ITSELF                              3      00000   DISP     X(10)         76    78 111

   BEGIN                                                  PARA-NM                80    NOREF
   THE-END                                                PARA-NM                112   NOREF
```

```
         COBOL    V-50.2      X86.1     LISTING  BOURGAIN BOURGAIN            13:22:48  MAR 31, 1978    PAGE      10
FIND-DAY          CROSS-REFERENCE LISTING (ALPHABETIC ORDER)
LN NAME                                    PN  ADDRESS   USAGE    PIC-STRING   DEF.  REF.  LINES

   BEGIN                                                 PARA-NM                80    NOREF
02 CENTURY (SPLIT-DATE)                        1:000F0   DISP     9(2)          55    84
02 DAY-IN-THE-WEEK (DITWEEK-TAB-RED)           1:000A4   DISP     X(10)         51    111
01 DAY-ITSELF                              3     00000   DISP     X(10)         76    78 111
03 DAY-OF-MONTH (SPLIT-DATE)                   1:000F6   DISP     9(2)          59    88
03 DAY-OF-MONTH-X (SPLIT-DATE)                 1:000F6   DISP     X(2)          60    82
01 DAY-OF-THE-WEEK                         2     00000   DISP     9(1)          74    78 110
01 DAYS-IN-THE-ERA                             1:000F8   DISP     9(10)         65    87 96 98 100 105
01 DITWEEK-TAB                                 1:000A4   GROUP    X(70)         41    NOREF
01 DITWEEK-TAB-RED                             1:000A4   GROUP    X(70)         50    NOREF
01 FULL-DATE                               1     00000   DISP     X(8)          71    78 81
03 MONTHR(SPLIT-DATE)                          1:000F4   DISP     9(2)          58    89 94
01 OTHER-UNUSED                                1:000A0   GROUP    X(4)          38    NOREF
01 PREC-D-TAB                                  1:00078   GROUP    X(36)         21    NOREF
01 PREC-D-TAB-RED                              1:00078   GROUP    X(36)         34    NOREF
02 PRECEDING-DAYS (PREC-D-TAB-RED)             1:00078   DISP     9(3)          35    89
02 SHORT-DATE (SPLIT-DATE)                     1:000F2   GROUP    X(6)          56    83
01 SPLIT-DATE                                  1:000F0   GROUP    X(8)          54    81 83
77 TALLY                                       1:00010   DISP     9(5)                NOREF
   THE-END                                               PARA-NM               112    NOREF
01 X                                           1:00064   DISP     9(10)         16    95 96 97 98 99 100 105
01 Y                                           1:00070   DISP     9(5)          17    105 106 107 109 110 111
01 YEAR                                        1:000F0   DISP     9(4)          61    90 94+ 95 97 99
```

```
FIND-DAY            COMPILATION SUMMARY


SUMMARY OF ERRORS

      *             5        ON LINES A.7 40 2.44 83
    * *             3        ON LINES 83 110
  * * *             0
* * * *             0


CU PRODUCED ON LIBRARY :000086.TEMP.CULIB


SEGMENT NAME        TYPE     SIZE (IN BYTES)

FIND-DAY.0          ..L          99
FIND-DAY.1          .D.         278
FIND-DAY.2          C..         434

STACK                            68


RUN TIME PACKAGE PROCEDURES INVOKED

NONE
```

APPENDIX B

$COBOL ERROR MESSAGES

1 -1	3	ILLEGAL CHARACTER. REPLACED BY BLANK.
1 -2	3	TOO LONG 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 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	2	AREA A IS IGNORED IN A CONTINUATION LINE.
1 -9	3	CONTINUATION LINE NOT ALLOWED AFTER DEBUGGING OR COMMENT LINE, WITHIN A COMMENT ENTRY OR AS FIRST LINE OF SOURCE OR COPIED TEXT. COLUMN 7 IS IGNORED.
1 -10	3	FIRST WORD 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. NOT ENOUGH ROOM TO ACCOMODATE "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	3	THIS "BY" PHRASE WILL NOT PARTICIPATE TO REPLACEMENT BECAUSE OF A PREVIOUS "BY" PHRASE.
1 -16	3	"COMPILE" COMMAND OR TERMINATING SEMI-COLON THEREOF ASSUMED TO BE MISSING. 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. LINE IS IGNORED.
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	5	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	4	THIS FEATURE IS A ¬ FEATURE, NOT INCLUDED IN THE CURRENT COMPILATION LEVEL.
1 -27	4	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 OR BOOLEAN LITERAL. SPACE OR ZERO IS ASSUMED, RESPECTIVELY.
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 COMMAND.

1 -37	4	THE FIRST COMMAND OF THE ALTER ENCLOSURE IS NOT AN "R" COMMAND WITHOUT ADDRESS EXPRESSION.
1 -38	4	AN "R" COMMAND IS ALLOWED ONLY AS THE FIRST COMMAND 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 "C", A "D" OR AN "S" COMMAND.
1 -42	4	END OF COMMAND 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 NEW COMPILATION. REPOSITIONNING ON ¬ CANNOT BE DONE.
1 -56	3	DEBUGGING LINES ARE ALLOWED ONLY AFTER THE 'OBJECT-COMPUTER' PARAGRAPH. LINE IS IGNORED.
1 -57	3	"COMPILE" COMMAND NOT RECOGNIZED. FIRST GROUP OF CONTIGUOUS NON BLANK CHARACTERS IS IGNORED.
1 -58	3	TEXT FOLLOWS SEMI-COLON. TEXT IS IGNORED.
1 -59	3	OPTION CANNOT BE RECOGNIZED. OPTION IS IGNORED.
1 -60	4	SEMI-COLON MISSING AT THE END OF THE "COMPILE" COMMAND.
1 -61	3	"¬", THOUGH SPECIFIED AS AN INPUT LIBRARY, DOES NOT CONTAIN TEXT.
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	"R" COMMAND DOES NOT SPECIFY A SOURCE PROGRAM, AND NONE IS WAITING FOR THIS COMPILATION.
1 -66	3	TEXT FOLLOWS "R" COMMAND. TEXT IS IGNORED.
1 -67	4	LIBRARY-NAME SPECIFIED IN "R" COMMAND IS TOO LONG.
1 -68	4	MEMBER-NAME MISSING IN "R" COMMAND.
1 -69	4	MEMBER-NAME SPECIFIED IN "R" 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	5	ILLEGAL CONTINUATION OF A NAME. COLUMN 7 IS IGNORED.
1 -78	1	LEVEL-64 SPECIFIC SYSTEM NAME.
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 5 SEPARATOR MISSING BEFORE THE WORD. BLANK IS ASSUMED.

1 -86 4 REFERENCED LINE NOT FOUND OR ALREADY PASSED.

1 -87 5 PUNCTUATION CHARACTER IS NOT FOLLOWED BY A BLANK. MISSING BLANK IS ASSUMED.

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.

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 THE FIRST LINE FOLLOWING THE CURRENT COPIED TEXT (IF ANY).

1 -95 1 THE END OF THIS LINE IS NOT PROCESSED FROM THIS POINT ON. IT IS REPEATED 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 NAME. NAME IS TRUNCATED.

1 -99 3 NAME TERMINATES WHITH AN 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.
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 ¬ MUST BE EITHER COBOL, OR COBOLX, OR
          DATASSF.
1 -124 3  SSF FORMAT FOR ¬ MUST BE EITHER COBOL, OR COBOLX, OR
          DATASSF.
1 -125 4  SSF FORMAT FOR ¬ MUST BE 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  "BY" PHRASE MISSING.
1 -130 4  ERROR WHILE READING ¬.
1 -131 3  EXPECTED WORD WAS "BY".
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 COMMAND.
1 -143 4  "¬" IS ASSUMED TO BE AN EXTERNAL FILE-NAME AND AS SUCH
          IS NOT ALLOWED IN AN "R" COMMAND. ONLY "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.
          MISSING BLANK IS ASSUMED.
1 -146 3  COPY STATEMENT NOT FULLY CONTAINED IN A DEBUGGING LINE.
1 -147 3  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 4      ERROR MESSAGES ABOUT THE CURRENT COPY STATEMENT
              HAVE BEEN LOST FROM THIS POINT ON.
1 -153 2      THIS FEATURE IS A LEVEL-62 SPECIFIC 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 REFFRENCE.
1 -157 5      ¬ IS ASSUMED TO BE IN SSF FORMAT.
1 -158 2      LEVEL-62 SPECIFIC COLUMN 7. THE LINE IS PROCESSED AS A
              COMMENT LINE; I.E. IS IGNORED.
1 -159 4      THIS FEATURE (NON CONTIGUOUS SECTIONS OF THE SAME
              PRIORITY) IS A ¬ FEATURE, NOT INCLUDED IN THE CURRENT
              COMPILATION LEVEL.
1 -160 3      EXCESS NUMBER OF CHARACTERS IS SPECIFIED IN PICTURE
              CHARACTER STRING, IT MUST NOT EXCEED 30.
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 CHARACTER IS SPECIFIED IN THE PICTURE
              CHARACTER STRING.
1 -165 1      ¬ DIVISION MISSING.
1 -166 3      ILLEGAL DATA TYPE IN "Z" REQUEST.
1 -167 3      UNABLE TO PROVIDE ALTERED SOURCE.
1 -168 2      THE SYNTAX AND BEHAVIOR OF THE COBOL "Z" REQUEST MAY
              DIFFER FROM THOSE OF LIBMAINT.
```

```
2 -1    2  THIS RESERVED WORD SHOULD BEGIN IN AREA A.
2 -2    3  THE RESERVED WORD DIVISION SHOULD APPEAR HERE.
2 -3    2  MISSING PERIOD.
2 -4    3  THE RESERVED WORD PROGRAM-ID SHOULD APPEAR HERE.
2 -5    3  THE PROGRAM NAME IS MISSING OR INCORRECTLY SPECIFIED.
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 APPEARED IN
           INCORRECT ORDER; ANSI REQUIRES THE CORRECT ORDER.
2 -9    3  CONFLICTING CLAUSES IN THIS SELECT STATEMENT: ¬
2 -10   3  INVALID COMPUTER NAME.
2 -11   3  A MNEMONIC NAME HAS BEEN SPECIFIED PREVIOUSLY FOR THIS
           WORD.
2 -12   3  THE RESERVED WORD SECTION SHOULD APPEAR HERE.
2 -13   2  THE WORD DEBUGGING OR SUPERVISOR SHOULD APPEAR HERE.
2 -14   3  THE RESERVED WORD MODE SHOULD APPEAR HERE.
2 -15   3  THE RESERVED WORD SUPERVISOR SHOULD APPEAR HERE.
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.
2 -23   3  INVALID SEGMENT-LIMIT CLAUSE.
2 -24   3  INVALID ASSIGN CLAUSE.
2 -25   3  THE CURRENCY SIGN LITERAL IS INVALID.
2 -26   2  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  THE ASSIGN CLAUSE IS MISSING FROM THIS SELECT
           STATEMENT.
2 -32   3  FILE INCORRECTLY ASSIGNED.
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  PADDING 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  A BANNER CHARACTER CANNOT BE APPLIED ON THIS FILE.
2 -40   3  THE ORGANIZATION QUALIFIER IS INCOMPATIBLE WITH THE
           ORGANIZATION.
2 -41   4  THIS FEATURE IS A ¬ FEATURE NOT INCLUDED IN THE
           CURRENT COMPILATION LEVEL.
2 -42   1  THE COMPUTER NAME SHOULD BE "LEVEL-64" OR "GCOS".
2 -43   3  INVALID OPTION IN A SELECT PHRASE.
2 -44   3  DUPLICATE CHARACTER IN ALPHABET NAME SPECIFICATION.
2 -45   3  INVALID RECORD PREFIX.
2 -46   3  INVALID INPUT-OUTPUT TECHNIQUE.
2 -47   3  INVALID KEY NAME.
2 -48   3  INVALID FILE NAME.
```

2 -49	2	THE RESERVED WORD ON SHOULD APPEAR HERE.
2 -50	3	INVALID DEVICE.
2 -51	3	THE RESERVED WORD CHECKPOINT-FILE SHOULD APPEAR HERE
2 -52	3	INVALID CONDITION-NAME.
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 IS NOT ACCESSED SEQUENTIALLY
2 -56	3	INVALID RERUN CLAUSE.
2 -57	3	ONLY ONE FILE NAME WAS SPECIFIED IN THIS SAME CLAUSE.
2 -58	1	NO SYNTAX CHECKING FROM THE LAST DIAGNOSTIC TO THIS POINT.
2 -59	3	INVALID 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 IFN 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 SELECT CLAUSE.
2 -67	1	INCORRECT ORDER OF CLAUSES IN THIS SELECT STATEMENT.
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	3	INVALID FILE STATUS NAME.
2 -72	2	DUPLICATE BANNER CLAUSE FOR THIS FILE; FIRST CLAUSE ACCEPTED.
2 -73	3	NO-RESIDENT-INDEX 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	3	THIS I-O TECHNIQUE IS INCOMPATIBLE WITH A TECHNIQUE PREVIOUSLY SPECIFIED FOR THE SAME FILE.
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	2	MISPLACED DECIMAL-POINT CLAUSE HAS AFFECTED LEXICAL ANALYSIS OF PROGRAM.
2 -81	2	THE FILE WAS ALREADY REFERENCED IN A PREVIOUS RERUN CLAUSE; FIRST CLAUSE ACCEPTED
2 -83	3	THIS FILE HAS ALREADY APPEARED IN A MULTIPLE FILE CLAUSE.
2 -84	3	A PREVIOUSLY SELECTED FILE IS ASSIGNED TO SYS-WRITE.
2 -85	3	INVALID QUALIFIER ON KEY OR FILE STATUS NAME.
2 -86	3	INVALID IMPLEMENTOR-NAME.
2 -87	3	INVALID COLLATING SEQUENCE CLAUSE.
2 -88	2	THE RESERVED WORD RECORD SHOULD APPEAR HERE.
2 -89	3	INVALID DUPLICATES CLAUSE.
2 -90	1	INCORRECT ORDER OF SOURCE-COMPUTER CLAUSES.
2 -91	3	INVALID SEGMENT SIZE CLAUSE.
2 -92	3	INVALID ADDRESS FORMAT.
2 -93	3	THE SPECIFIED ADDRESS FORMAT CANNOT BE APPLIED ON THIS FILE.
2 -94	2	DUPLICATE ADDRESS FORMAT CLAUSE FOR THIS FILE; FIRST

CLAUSE ACCEPTED.

2 -95 3 DEFAULT CLAUSE CANNOT BE RECOGNIZED

2 -96 2 PROGRAM-NAME EXCEEDS 12 CHARACTER IN LENGTH

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 "IFN" SUFFIX

2 -101 2 THIS CLAUSE IS USED FOR DOCUMENTATION ONLY

2 -102 2 THIS FEATURE WILL NOT BE ALLOWED WITH THE NEXT RELEASE

2 -103 3 SYNTAX ERROR

2 -104 3 THE SPECIFIFIED SIZE MAY NOT EXCEED 32K BYTES FOR PROCEDURE SEGMENTS OR 4M BYTES FOR DATA SEGMENTS.

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 2 DUPLICATE DEFAULT FOR SYMBOLIC QUEUE CLAUSE

2 -109 2 DUPLICATE DEFAULT FOR TEMP CLAUSE; FIRST CLAUSE ACCEPTED

2 -110 2 DUPLICATE DEFAULT FOR ACCEPT CLAUSE; FIRST CLAUSE ACCEPTED

2 -111 2 DUPLICATE DEFAULT FOR DISPLAY CLAUSE; FIRST CLAUSE ACCEPTED

2 -112 3 THE RESERVED WORD IDENTIFICATION SHOULD APPEAR HERE

2 -113 3 SPECIFIED ORGANIZATION IS ILLEGAL

2 -114 3 DEFAULT FOR TEMP MUST BE BETWEEN 18 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 IFN GIVEN TO THIS FILE.

2 -119 3 THE MEMBERS OF A MULTIPLE FILE MUST HAVE THE SAME DEVICE AND ORGANIZATION CLAUSE.

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 ORGANIZATION SEQUENTIAL.

2 -123 2 NON SEQUENTIAL ORGANIZATION OPTIONAL FILE IS A LEVEL-62 SPECIFIC FEATURE.

2 -124 2 THIS MAY BE A LEVEL-62 INTERNAL DEVICE DESIGNATOR.

2 -125 2 THIS IS A LEVEL-62 SPECIFIC FEATURE.

2 -126 2 THIS LEVEL-62 SPECIFIC FEATURE IS IGNORED.

2 -127 3 THE RESERVED WORD CONSOLE SHOULD APPEAR HERE.

2 -128 4 THIS FEATURE (OPTIONAL OR EMPTY SECTION OR PARAGRAPH) IS A LEVEL-64 FEATURE NOT INCLUDED IN THE CURRENT COMPILATION LEVEL.

2 -129 2 A LEVEL-62 SWITCH STATUS NAME ASSUMED, THE RERUN CLAUSE IS IGNORED.

2 -130 3 THIS LEVEL-62 SPECIFIC FEATURE IS NOT IMPLEMENTED.

2 -131 4 THE INTERACTIVE MODE IS NOT AVAILABLE ON YOUR SITE, PLEASE CONTACT SUPPLIER.

2 -132 5 CONFLICTING CLAUSES IN THIS SELECT STATEMENT: ¬

2 -150 3 UNEQUAL SIZE 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 CONTAINS... 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 3 LEVEL-68 IS ALLOWED ONLY IN LEVEL-68 COBOL.

2 -160 3 UNBANNERED 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 THE INTERNAL-FILE-NAME MUST BE H_SORT AS RANDOM APPLIES ONLY TO SORT FILES.

2 -163 3 INVALID CATALOGUE-NAME.

2 -164 3 THE RESERVED WORD TEMPORARY OR PERMANENT SHOULD APPEAR HERE.

2 -165 3 NO-SORTED-INDEX APPLIES ONLY TO INDEXED FILES DESCRIBED WITH ALTERNATE KEYS.

2 -166 2 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 2 DUPLICATE DEFAULT FOR COMP CLAUSE, THE FIRST CLAUSE WAS ACCEPTED.

2 -169 3 THE DEFAULT FOR COMP CLAUSE IS IMPROPERLY STATED.

2 -170 2 A FILLER IS MISSING TO ACCOMODATE SYNCHRONIZED IN THE CURRENT REDEFINITION: FILLER IS PROVIDED

2 -171 2 THE SYNCHRONIZATION CANNOT BE ACCOMODATED FOR ALL OCCURRENCES OF THIS ITEM

2 -172 2 THIS FEATURE IS NOT IMPLEMENTED

2 -173 2 THE SIZE OF THE 01 OR 77 LEVEL ITEM EXCEEDS THE SPECIFIED OR IMPLIED MAXIMUM SEGMENT SIZE

2 -174 2 IMPLEMENTATION RESTRICTION: TOO MANY ITEMS SUBORDINATE TO THIS ITEM OR REDEFING 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 IMPLICITELY REDEFINES

2 -176 2 THE VALUE CLAUSE HAS BEEN DISREGARDED BECAUSE OF THE INITIAL ATTRIBUTE OF THE CD, THOUGH PART OF IT MIGHT BE SIGNIFICANT

2 -177 2 A VALUE CLAUSE CANNOT BE SPECIFIED FOR A 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 CONTAINS CLAUSE

2 -181 2 THE LENGTH OF THIS RECORD IS NOT EQUAL TO THE SPECIFIED IN THE RECORD CONTAINS 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 CONTAIN CLAUSE IS NOT A MULTIPLE OF THE RECORD SIZE

2 -187 3 LINAGE CLAUSE MAY ONLY BE USED FOR AN SSF FILE

2 -188 2 THIS SYNTAX OF THE RECORD PREFIX PHRASE WILL NOT BE ACCEPTED AFTER THIS RELEASE

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 REPEATITITION NUMBER IS GREATER THAN 65535.

2 -192 2 THE CODE-SET CLAUSE IS NOT ALLOWED WITH THE FILE ORGANIZATION.

2 -193 4 NO SPACE AVAILABLE TO PROCESS THE SYNCHRONIZED ATTRIBUTE.

2 -194 3 INTERNAL FILE NAME "H-SORT" IS RESERVED FOR SORT FILES.

2 -195 3 THE SELECT CLAUSE FOR A SORT FILE CAN ONLY CONTAIN THE ASSIGN(MANDATORY) CLAUSE AND 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 INSERTED FOR SYNCHRONIZATION WAS NOT TAKEN CARE OF IN THE REDEFINITION.

2 -198 1 A ¬ TYPE 1 FILLER WAS ADDED AT THE END OF THIS ITEM (SEE REFERENCE MANUAL).

2 -199 1 A ¬ TYPE 2 FILLER WAS ALLOCATED TO ALIGN THIS SYNCHRONIZED ITEM (SEE 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 CONTAINS CLAUSE IS GREATER THAN THE SIZE OF THE LARGER RECORD DESCRIBED FOR THIS FILE; IT WILL BE TAKEN CARE OF, FROM THE NEXT RELEASE ON, IN DETERMING THE RECORD (AREA) SIZE.

3 -1 3 UNRECOGNIZABLE SECTION SPECIFICATION HAS OCCURRED.

3 -2 2 RESERVED WORD SECTION IS MISSING.

3 -3 2 PERIOD IS MISSING.

3 -4 3 REDUNDANT FILE SECTION HAS DETECTED, ONLY ONE FILE SECTION IS ALLOWED PER PROGRAM.

3 -5 3 SECTIONS PRECEDENCE SYNTAX ERROR IS DETECTED, CHECK COBOL MANUAL FOR CORRECTION.

3 -6 3 UNRECOGNIZABLE FILE SECTION LEVEL INDICATOR HAS OCCURRED, IT MUST BE FD, SD.

3 -7 3 THE RECORD NAME OF THIS FILE IS IN ERROR.

3 -8 3 RECORD HAS FATAL SYNTAX ERROR, SYNTAX ANALYSIS OF THIS RECORD IS NOT COMPLETED.

3 -9 3 IN THE PRESENT DATA ENTRY- THE FOLLOWING DATA PROPERTIES ARE INCONSISTENT WITH ¬

3 -10 3 FILE NAME IS NOT DEFINED IN ENVIRONMENT DIVISION.

3 -11 3 UNRECOGNIZABLE FD CLAUSES ARE ENCOUNTERED.

3 -12 3 LABEL CLAUSE IS MISSING IN CURRENT FD ENTRY.

3 -13 1 RECORD DESCRIPTION IS MISSING .

3 -14 3 FILE RECORDING CODE NAME IS IN ERROR.

3 -15 1 CHARACTERS OPTION IS ASSUMED FOR THE BLOCK CLAUSE.

3 -16 3 MAXIMUM BLOCK SIZE INTEGER IS MISSING.

3 -17 3 MAXIMUM RECORDS SIZE INTEGER IS MISSING.

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 OR DATA NAME IS MISSING.

3 -22 3 REDUNDENT TOP PHRASE IS SPECIFIED FOR LINAGE CLAUSE.

3 -23 3 LINAGE SPECIFICATION IS IN ERROR.

3 -24 3 SD DESCRIPTION CONTAINS FATAL SYNTAX ERROR, SYNTAX ANALYSIS IS NOT COMPLETED.

3 -25 3 LITERAL(INTEGER) IS MISSING.

3 -26 3 AREA CLAUSE IS MISSING.

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 REDEFINES CLAUSE WHEN USED MUST IMMEDIATLY FOLLOW THE SUBJECT OF REDEFINES .

3 -30 3 DD CLAUSE HEADER IS IN ERROR.

3 -31 3 THE OCCURS DEPENDING ON ITEM 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 DATA ITEM IS NOT FOUND AT EQUAL LEVEL, OR IS ITSELF THE SUBJECT OF REDEFINES.

3 -33 3 VALUE OF THE 88 CONDITION ITEM IS INCONSISTENT WITH THE PICTURE.

3 -34 3 .THE 66 RENAMES 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 IMMEDIATLY 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 88 CONDITION NAME ITEM CANNOT BE ASSOCIATED WITH A 66

LEVEL ITEM.

3 -40	3	88 CONDITION ITEM CANNOT BE ASSOCIATED WITH AN INDEX DATA ITEM.
3 -41	3	THE LITERAL VALUE AFTER THE THRU MUST BE GREATER THAN THE LITERAL VALUE BEFORE THRU.
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 1 OR 77 ITEM, NOR CAN IT BE REDUNDENT.
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	OBJECT OF REDEFINES DATA NAME CANNOT BE AN ITEM OF VARIABLE LENGTH.
3 -56	5	THE OBJECT OF REDEFINES DATA NAME IN FILE SECTION OR COMMUNICATION SECTION CANNOT BE AN 01 LEVEL ITEM,THE REDEFINITION IS IMPLIED.
3 -57	3	THE SUBJECT OF REDEFINES DATA ITEM CANNOT BE OF VARIABLE LENGTH.
3 -58	3	COMPILER LIMIT : TOO MANY VALUES IN 88 ENTRY.
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	THE OBJECT OF RENAMES DATA NAME CANNOT BE FOUND IN THE PREVIOUS RECORD.
3 -62	3	A 66 LEVEL ENTRY CANNOT RENAME ANOTHER 66, 01, 88, OR 77 LEVEL DATA ITEM.
3 -63	3	THE OBJECT OF RENAMES DATA ITEM CANNOT CONTAIN AN OCCURS CLAUSE, NOR CAN IT BE SUBORDINATE TO AN ITEM WHICH CONTAINS AN OCCURS CLAUSE.
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 SPECIFIEDTHE SUBORDINATE ITEM CANNOT HAVE ADDITIONAL INITIAL VALUE.
3 -69	3	THE INITIAL VALUE IS INCONSISTENT WITH THE PICTURE OF THE DATA ITEM.
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 1 JUST RIGHT IS ASSUMED.

3 -73 3 WHEN THE GROUP DATA ITEM HAS INITIAL VALUE,THE SUBORDINATE ITEM

3 -74 1 SYNC RIGHT IS ASSUMED.

3 -75 3 WHEN THE GROUP DATA ITEM HAS INITIAL VALUE,THE SUBORDINATE ITEM CANNOT CONTAIN SYNC CLAUSE. JUST CLAUSE

3 -76 4 THIS FEATURE IS A ¬ 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 DATA 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 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 MIXED INDEXING IS NOT ALLOWED, WHEN A TABLE ITEM HAS ONE LEVEL INDEXED, ALL LEVELS MUST ALSO BE INDEXED.

3 -91 3 THE SIGN TYPE OF A SUBORDINATE ITEM MUST BE CONSISTENT WITH THAT OF THE GROUP ITEM.

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 CD OUTPUT DESTINATION TABLE INDEX NAME OVER FLOW.

3 -96 2 THE OBJECT OF REDEFINES DATA NAME MAY BE IN ERROR.

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 CD CLAUSE IS ENCOUNTERED
3 -105 3 REDUNDANT CD INPUT SYMBOLIC SUBQUEUE-1 CLAUSE
3 -106 3 REDUNDANT CD INPUT SYMBOLIC SUBQUEUE-2 CLAUSE.
3 -107 3 REDUNDANT CD INPUT SYMBOLIC SUBQUEUE-3 CLAUSE
3 -108 3 REDUNDANT CD INPUT SYMBOLIC QUEUE CLAUSE
3 -109 3 REDUNDANT CD INPUT MESSAGE DATE CLAUSE
3 -110 3 REDUNDANT CD INPUT MESSAGE TIME CLAUSE
3 -111 3 REDUNDANT CD INPUT TEXT LENGTH CLAUSE
3 -112 3 REDUNDANT CD INPUT END KEY CLAUSE
3 -113 3 REDUNDANT CD INPUT STATUS KEY CLAUSE
3 -114 3 REDUNDANT CD INPUT QUEUE DEPTH CLAUSE
3 -115 3 REDUNDANT CD INPUT SYMBOLIC SOURCE CLAUSE
3 -116 3 REDUNDANT CD INPUT MESSAGE COUNT CLAUSE.
3 -117 3 UNRECOGNIZABLE CD OUTPUT ATTRIBUTE IS ENCOUNTERED
3 -118 3 THE MAXINUM OCCURRENCE NUMBER MUST BE NUMERIC INTEGER GREATER THAN 0.
3 -119 3 REDUNDANT CD OUTPUT DESTINATION COUNT CLAUSE
3 -120 3 REDUNDANT CD OUTPUT TEXT LENGTH CLAUSE
3 -121 3 REDUNDANT CD OUTPUT STATUS KEY CLAUSE
3 -122 3 REDUNDANT CD OUTPUT DESTINATION TABLE CLAUSE
3 -123 3 REDUNDANT CD OUTPUT ERROR KEY CLAUSE
3 -124 3 REDUNDANT CD OUTPUT SYMBOLIC DESTINATION CLAUSE
3 -125 3 THIS CD 01 RECORD HAS FATAL ERROR
3 -126 3 CD INPUT RECORD LENGTH MUST BE 87 CHARACTERS
3 -127 3 CD RECORD NAME ERROR
3 -128 3 CD OUTPUT RECORD LENGTH MUST BE GREATER THAN 23 CHARACTERS
3 -129 3 WORKING_STORAGE SECTION HAS FATAL SYNTAX ERROR, PARSING IS NOT COMPLETED.
3 -130 4 THIS FEATURE (LESS THAN 11 DATA NAMES) IS A LEVEL-64 SPECIFIC FEATURE NOT INCLUDED IN THE CURRENT COMPILATION LEVEL.
3 -131 3 REDUNDANT LINKAGE SECTION IS DETECTED, ONLY ONEIS SLLOWED PER PROGRAM.
3 -132 2 CODE-SET CLAUSE ILLEGAL ON NON SEQUENTIAL FILE.
3 -133 3 REDUNDANT CONSTANT SECTION IS DETECTED, ONLY ONE IS ALLOWED PER PROGRAM.
3 -134 5 CODE-SET CLAUSE ILLEGAL ON NON SEQUENTIAL FILE.
3 -135 3 THE USAGE SPECIFIED IS UNRECOGNIZABLE.
3 -136 3 CODE-SET CLAUSE ILLEGAL ON NON SEQUENTIAL FILE.
3 -137 3 CODE-SET CLAUSE REDUNDANT ON FD OR ILLEGAL ON SD.
3 -138 3 THE DESCRIPTION OF THIS 66 RENAMES ENTRY HAS FATAL SYNTAX ERROR, PARSING OF THIS ENTRY IS NOT COMPLETED.
3 -139 3 RECORDING MODE CLAUSE REDUNDANT ON FD OR ILLEGAL ON SD.
3 -140 3 BLOCK CONTAINS CLAUSE REDUNDANT ON FD OR ILLEGAL ON SD.
3 -141 3 RECORD CONTAINS CLAUSE REDUNDANT ON FD OR SD.
3 -142 3 LABEL RECORD CLAUSE REDUNDANT ON FD OR ILLEGAL ON SD.
3 -143 3 VALUE OF CLAUSE REDUNDANT ON FD OR ILLEGAL ON SD.
3 -144 3 DATA RECORD CLAUSE REDUNDANT ON FD OR SD.
3 -145 3 REPORT IS CLAUSE REDUNDANT ON FD OR ILLEGAL ON SD.
3 -146 3 LINAGE IS CLAUSE REDUNDANT ON FD OR ILLEGAL ON SD.
3 -147 3 REDUNDANT DEPENDING ON CLAUSE IS DETECTED, ONLY ONE IS ALLOWED.
3 -148 3 THE SPECIFICATION FOR LABEL RECORD 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 1 SYNTAX ERROR IS ENCOUNTERED AT THIS POINT, PARSING IS DISCONTINUED.
3 -153 1 A DUMMY RECORD NAME IS SUPPLIED, SYNTAX CHECKING IS RESUMED.
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 CONTAINS 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 HAS FATAL ERROR.
3 -160 3 RESERVED WORD KEY IS MISSING.
3 -161 1 SSF IS ASSUMED WITH A LINAGE OR A REPORT CLAUSE.
3 -162 3 THE LITERAL FOLLOWING THE THRU OPTION IS MISSING.
3 -163 3 THE 66 RENAMES ENTRY IS NOT PROPERLY POSITIONED, IT MUST IMMEDIATLY 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 1 THE RECORD DESCRIPTION FOR THE ABOVE FILE DESCRIPTION ENTRY IS MISSING.
3 -167 3 FILE NAME IS MISSING OR IN ERROR.
3 -168 3 THE RECORD PREFIX SPECIFIED IN THE SELECT PHRASE CONFLICTS WITH A LINAGE OR REPORT CLAUSE.
3 -169 3 WHEN THE GROUP DATA ITEM IS ASSOCIATED WITH LEVEL 88 ITEMS, THE SUBORDINATE ITEMS MUST BE USAGE DISPLAY.
3 -170 3 UNRECOGNIZABLE ATTRIBUTE IN CD ENTRY IS ENCOUNTERED
3 -171 3 RW TIMES IS MISSING
3 -172 3 RW KEY IS MISSING
3 -173 3 RW LENGTH IS MISSING
3 -174 3 RW TOP OR BOTTOM IS MISSING.
3 -175 3 REDUNDENT BOTTOM PHRASE IS SPECIFIED FOR LINAGE CLAUSE.
3 -176 3 REDUNDENTFOOTING PHRASE IS SPECIFIED FOR LINAGE CLAUSE.
3 -177 5 A NON ZERO UNSIGNED INTEGER SHOULD APPEAR HERE.
3 -178 3 RW FOOTING IS MISSING.
3 -179 3 THE FOOTING INTEGER MUST NOT BE GREATER THAN THE BODY INTEGER IN LINAGE CLAUSE
3 -180 3 THE SUBJECT OF REDEFINES MUST NOT BE A FILLER 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 5  AN UNSIGNED INTEGER SHOULD APPEAR HERE.
3 -188 3  WHEN FD HAS JIS CODE-SET, SIGNED NUMERIC DATA MUST HAVE
          SIGN IS SEPARATE CLAUSE.
3 -189 2  RECORD CONTAINS...DEPENDING ON IS SPECIFIED IN FD, BUT
          FILE-CONTROL ENTRY SPECIFIES FLR.
3 -190 3  THE SIGN CLAUSE MUST BE ASSOCIATED WITH AT LEAST ONE
          NUMERIC ITEM WITH PICTURE CONTAINING S.
3 -191 1  RECORD DESCRIPTION ASSUMED TO BE DATA RECORD FOR
          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 4  OVERFLOW IN HIERARCHY TABLE: PROCESSING OF DATA
          DIVISION CEASES HERE!
3 -196 4  OVERFLOW IN INDEXNAME TABLE: PROCESSING OF DATA
          DIVISION CEASES HERE!
3 -197 3  RW "DEPENDING" 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  MUST  BE  USAGE  IS
          DISPLAY.
3 -200 1  IN COMPLIANCE WITH STANDARD: CODE-SET CLAUSE ON FD
          SHOULD BE ACCOMPANIED BY ALL DATA USAGE IS DISPLAY.
3 -201 3  ONLY NUMERIC LITERALS ARE ALLOWED IN THE LINAGE
          CLAUSE FOR AN EXTERNAL FILE.
3 -202 3  DESTINATION TABLE MAY ONLY OCCUR 1 TIME IN THIS
          IMPLEMENTATION.
3 -203 3  A RESERVED WORD HAS BEEN USED AS A USER WORD OR
          DATA-NAME IS MISSING.
3 -204 1  REMAINDER OF VALUE OF CLAUSE IS SCANNED OFF
3 -205 3  ALL MAY NOT BE USED WITH A NUMERIC LITERAL.
3 -206 3  THIS RELEASE REQUIRES SEPARATE SIGN FOR SIGNED NUMERIC
          ITEMS.
3 -207 2  NOT SUPPORTED IN THIS RELEASE, WILL BE IGNORED.
3 -208 3  THIS RELEASE REQUIRES THAT DEFAULT COMP BE DISPLAY.
3 -209 3  THIS FEATURE IS NOT IMPLEMENTED.
3 -210 2  LABEL RECORD FORMAT NOT SUPPORTED BY THIS RELEASE.
3 -211 2  TOO MANY RECORD-NAMES IN DATA RECORDS.
3 -212 2  RESERVED WORD DIVISION IS MISSING.
3 -213 2  LEVEL NUMBER HIERARCHY INCORRECT
3 -214 3  THE NUMBER OF DIGIT PORTIONS SPECIFIED FOR THIS ITEM
          EXCEEDS THE MAXIMUM ALLOWED.
3 -215 2  THE RECORD CONTAINS CLAUSE SPECIFIES TOO LARGE A RECORD
          SIZE
3 -216 2  INCONSISTENT VALUES IN THE RECORD CONTAINS CLAUSE
3 -217 2  INCONSISTENT VALUES IN THE BLOCK CONTAINS CLAUSE
3 -218 3  USAGE IS COMP-1 OR COMP-2 DOES NOT ALLOW A PICTURE
          CHARCTER STRING WITH A SCALING FACTOR
3 -219 5  ONLY USAGE DISPLAY IS ALLOWED WHEN ORGANIZATION IS
          H-2000 OR ANSI.
3 -220 2  SIGN CLAUSE CANNOT APPLY TO ANY CONTAINED NUMERIC
```

```
         DISPLAY ELEMENTARY ITEM (IF ANY).
3 -221 5 THE SIGN CLAUSE MUST BE ASSOCIATED WITH USAGE DISPLAY
         AND PICTURE 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 SPECIFIC FEATURE IS NOT IMPLEMENTED.
3 -226 2 THIS IS A LEVEL-62 SPECIFIC FEATURE.
3 -227 4 THIS FEATURE (FILLER AT GROUP LEVEL) IS A LEVEL-64
         FEATURE NOT INCLUDED IN THE CURRENT COMPILATION LEVEL.
3 -228 2 COMP-8 IS ASSUMED FOR THIS LEVEL-62 COMP-3 ITEM.
3 -229 3 REDUNDANT EXTERNAL CLAUSE IS DETECTED, ONLY ONE IS
         ALLOWED PER DATA ITEM.
3 -230 3 ONLY WORKING-STORAGE OR CONSTANT SECTIONS 01 OR 77
         ENTRIES WITHOUT REDEFINES CAN HAVE THE EXTERNAL CLAUSE.
3 -231 4 THIS FEATURE (MISSING DATA NAME) IS A LEVEL-64 FEATURE
         NOT INCLUDED IN THE CURRENT COMPILATION LEVEL.
3 -232 3 THIS LEVEL-62 SPECIFIC FEATURE IS NOT IMPLEMENTED,
         SUBSEQUENT ENTRIES ARE ASSUMED TO BE WORKING-STORAGE.
3 -233 3 THE VALUE CLAUSE CANNOT BE USED TO DESCRIBE A FLOATING
         POINT NUMBER.
3 -234 3 A FLOATING POINT NUMBER CANNOT BE A CONDITIONAL
         VARIABLE.
3 -235 4 THE REPORT WRITER IS NOT AVAILABLE ON YOUR SITE, PLEASE
         CONTACT SUPPLIER.
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.
```

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 FROM A AREA.
4 -5	2	THIS ITEM SHOULD BE WRITTEN IN AREA B.
4 -6	2	SYNTAX CHECK DISCONTINUED FROM THIS ITEM.
4 -7	2	SYNTAX CHECK IS RESUMED.
4 -9	2	END OF DATA DIVISION WAS DETECTED.
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 UNMATCH.
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"/"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	NO SPACE AVAILABLE TO ACCOMODATE THIS REPORT DESCRIPTION.
4 -31	4	IMPLEMENTATION LIMIT - NOT ENOUGH SPACE AVAILABLE TO ACCOMODATE THIS REPORT DESCRIPTION.
4 -32	3	THIS FEATURE IS NOT IMPLEMENTED.
4 -40	3	OPERAND OF CONTROL CLAUSE IS MISSING.
4 -41	3	DUPLICATE OPERAND IN CONTROL CLAUSE.
4 -42	2	FINAL SHOULD BE THE FIRST CONTROL.
4 -51	3	SYNTAX ERROR IN PAGE CLAUSE.
4 -52	2	"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 -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 -80	3	SYNTAX ERROR IN TYPE CLAUSE.
4 -81	3	"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	"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	ABSOULTE 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	ABSOULTE 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 BODY AND RF.
4 -110	3	SYNTAX ERROR IN PICTURE CLAUSE.
4 -111	3	ILLEGAL CHARACTER IN PICTURE STRING.
4 -112	3	JUSTIFIED CLAUSE CONFLICTS WITH OTHER CLAUSE WITHIN THIS ITEM.
4 -113	3	"ZERO" IS MISSING AFTER "BLANK".
4 -114	3	BLANK WHEN ZERO CONFLICTS WITH OTHER CLAUSE WITHIN THIS ITEM.
4 -115	3	GROUP INDICATE CONFLICTS WITH OTHER CLAUSE WITHIN THIS ITEM.
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 OPERAND IS MISSING.
4 -126	3	LITERAL AFTER "ALL" IS MISSING.
4 -127	3	VALUE OPERAND IS INCONSISTENT WITH ITEM CLASS.
4 -130	3	SUM OPERAND IS MISSING.
4 -131	3	SUM CLAUSE SHOULD BE SPECIFIED WITHIN CF.
4 -132	3	UPON OPERAND IS MISSING.
4 -133	3	RESET OPERAND IS MISSING.
4 -134	3	RESET CONTROL LEVEL CANNOT BE DEFINED FOR THIS ITEM.
4 -135	3	SUM 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 OPERAND SHOULD BE DETAIL GROUP WITHIN SAME REPORT.

4 -138 3 RESET CLAUSE SHOULD SPECIFY HIFHER OR SAME CONTROL LEVEL.

4 -139 3 MULTI-DEFINED DATA NAME WITHIN SUM OR UPON OPERAND.

4 -140 3 SUM OPERAND FOR SUM CLAJSE WITH "UPON" SHOULD NOT BE SUM COUNTER.

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 "SOURCE" / "SUM" / "VALUE".

4 -174 3 FORMAT-3 ITEM WITHOUT MANDATORY CLAUSE.

4 -175 3 NO ITEM CAN BE SUBORDINATE TO FORMAT-3 ITEM.

5 -1	3	EXPECTED WORD WAS "PROCEDURE".
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	2	NUMBER OF USING PARAMETERS NOT EQUAL TO LINKAGE SECTION COUNT.
5 -7	2	PERIOD EXPECTED AFTER THE PREVIOUS WORD
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 FILENAME.
5 -12	3	ITEM HAS "LABEL OMITTED" CLAUSE.
5 -13	3	EXPECTED WORD WAS "LABEL"
5 -14	3	ITEM IS NOT REPORT SECTION DATA-NAME
5 -15	3	ALPHABET-NAME IS UNKNOWN
5 -16	2	THIS FEATURE IS NOT IMPLEMENTED YET. IT HAS BEEN SCANNED OFF
5 -17	3	"END COBOL" IN WRONG PLACE.
5 -18	1	THIS OPTION IS NOT MEANINGFUL IN LEVEL-64. IT HAS BEEN SCANNED OFF.
5 -19	3	SUBSCRIPT VALUE IS OUT OF RANGE
5 -20	5	ITEM IS NOT PARAGRAPH OR SECTION DECLARATION
5 -21	3	ITEM IS NOT IDENTIFIER
5 -22	3	RECEIVING FIELD FOR THIS ITEM IS ALPHABETIC
5 -23	3	EXPECTED WORD WAS "SYSIN", "CONSOLE", "DATE", DAY", "DAY-OF-WEEK" OR MNEMONIC-NAME
5 -24	3	ITEM IS NOT ELEMENTARY NUMERIC
5 -25	3	ITEM IS NOT ELEMENTARY NUMERIC OR IS NOT "TO" OR "GIVING".
5 -26	3	ITEM IS NOT ELEMENTARY NUMERIC OR EDITED ELEMENTARY NUMERIC
5 -27	3	ITEM IS NOT ALTERABLE PROCEDURE NAME
5 -28	3	ITEM IS NOT "TO"
5 -29	3	ITEM IS NOT NON-NUMERIC LITERAL OR IDENTIFIER
5 -30	3	ITEM IS NOT 01 OR 77 ITEM IN FILE, WS, COMMUNICATION, OR LINKAGE SECTIONS
5 -31	3	ITEM IS NOT NON-SORT FILENAME
5 -32	3	ITEM IS NOT "REWIND".
5 -33	3	ITEM IS NOT "FROM", "=", OR "EQUALS".
5 -34	3	ITEM IS NOT "INPUT" OR "OUTPUT".
5 -35	3	ITEM IS NOT AN OUTPUT CDNAME
5 -36	3	ITEM IS NOT AN INPUT CDNAME
5 -37	3	ITEM IS NOT "KEY".
5 -38	3	ITEM IS NOT ALPHANUMERIC IDENTIFIER OR LITERAL
5 -39	3	ITEM IS NOT IDENTIFIER OR LITERAL
5 -40	3	ITEM IS NOT PROPER DEVICE
5 -41	3	EXPECTED WORD WAS "INTO" OR "BY".
5 -42	3	"INVALID KEY" SHOULD NOT BE USED FOR THE FILE.
5 -43	3	ITEM CANNOT BE USED IN A "GENERATE" STATEMENT.
5 -44	3	ITEM IS NOT PROCEDURE NAME OR "DEPENDING".
5 -45	3	ITEM IS NOT ELEMENTARY NUMERIC INTEGER

```
5 -46   3   ITEM IS NOT DECLARATIVE SECTION NAME
5 -47   3   ITEM IS NOT REPORT NAME .
5 -48   3   ITEM IS NOT IDENTIFIER WITH "USAGE IS DISPLAY" CLAUSE.
5 -49   3   ITEM IS NOT "TALLYING" OR "REPLACING".
5 -50   3   ITEM IS NOT "FOR".
5 -51   3   ITEM IS NOT "ALL", "LEADING" OR "CHARACTERS".
5 -52   3   ITEM IS NOT NON NUMERIC LITERAL OR ELEMENTARY DATA ITEM
            WITH "USAGE IS DISPLAY" CLAUSE.
5 -53   3   ITEM IS NOT "ALL", "LEADING" OR "FIRST".
5 -54   3   ITEM IS NOT "BY".
5 -55   3   ITEM IS NOT "SEQUENCE".
5 -56   3   ITEM SIZE IS NOT EQUAL TO ITEM REPLACED
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 REFERENCE PROCEDURE NAME
5 -63   3   ITEM IS NOT "TIMES".
5 -64   3   ITEM IS NOT ELEMENTARY NUMERIC ITEM OR 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 CANNOT HAVE VARIABLE SIZE RECORDS
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 NAME
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   5   THIS REFERENCE SHOULD BE A SECTION REFERENCE.
5 -81   3   ITEM IS NOT NON-SUBSCRIPTED AND NON-INDEXED WITH BOTH
            OCCURS AND 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   5   RULES FOR TRANSFER OF CONTROL BETWEEN PROCEDURES
            ARE VIOLATED.
5 -87   3   ITEM IS NOT INDEX NAME IDENTIFIER OR POSITIVE INTEGER
5 -88   3   ITEM IS NOT INDEX DATA NAME OR ELEMENTARY INTEGER
5 -89   3   ITEM IS NOT INTEGER
5 -90   3   ITEM IS NOT SORT FILE.
5 -91   3   ITEM IS NOT "ASCENDING" OR "DESCENDING".
5 -92   3   ITEM IS NOT DATA NAME 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 DOES NOT HAVE "USAGE IS DISPLAY" CLAUSE.
5 -96   3   ITEM IS NOT NON-NUMERIC IDENTIFIER OR LITERAL OR
```

"DELIMITED".

```
5 -97  3  ITEM IS NOT NON-NUMERIC IDENTIFIER OR LITERAL OR "SIZE"
5 -98  3  ITEM IS NOT FIXED LENGTH WITH "USAGE IS DISPLAY" CLAUSE
5 -99  3  ITEM IS NOT IDENTIFIER OR NON NUMERIC LITERAL OR "INTO"
5 -100 3  ITEM SHOULD BE ELEMENTARY WITH NO EDIT AND WITH "USAGE
          IS DISPLAY" CLAUSE.
5 -101 3  ITEM IS NOT ELEMENTARY NUMERIC INTEGER DATA ITEM
5 -102 3  ITEM IS NOT NUMERIC IDENTIFIER OR LITERAL
5 -103 3  ITEM IS NOT NUMERIC IDENTIFIER OR LITERAL OR "FROM".
5 -104 2  PROGRAM SHOULD END WITH A ".".
5 -105 3  ITEM IS NOT ALPHANUMERIC
5 -106 3  ITEM IS NOT "INTO" OR "DELIMITED".
5 -107 3  ITEM CAN NOT BE USED WITHOUT "DELIMITED".
5 -108 3  ITEM IS NOT RECORD NAME IN NON SORT FILE
5 -109 3  ITEM IS NOT IDENTIFIER INTEGER OR MNEMONIC NAME
5 -110 3  "LINAGE" CLAUSE IS MISSING IN ASSOCIATED FILE.
5 -111 3  NO "USE" APPLICABLE, SO "AT END" OR "INVALID" MANDATORY
5 -112 3  EXPECTED WORD WAS "("
5 -113 3  EXPECTED WORD WAS ")"
5 -114 3  ITEM IS NOT A PROPER SUBSCRIPT
5 -115 3  ITEM SHOULD BE INDEX NAME
5 -116 3  ITEM IS NOT AN INDEX NAME OR NOT CORRECT INDEX NAME
5 -117 3  ITEM IS NOT AN UNSIGNED INTEGER
5 -118 5  THE SIZE OF THE COMPOSIT 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 IS 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 5  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 TO".
5 -127 3  ITEM IS NOT "RUN" OR LITERAL
5 -128 3  ITEM IS NOT IMPERATIVE VERB
5 -129 3  EXPECTED WORD WAS "SECTION" COMPILER ERROR
5 -130 3  SYNTAX ERROR. CHECK AGAINST 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  ONLY PARAGRAPH (OR SECTION) NAME OR VERB ALLOWED HERE
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 "ESCAPE".
5 -139 3  EXPECTED WORD WAS "END"
5 -140 3  EXPECTED WORD WAS "COBOL".
5 -141 3  ITEM IS NOT CDNAME, IDENTIFIER, PROCEDURE NAME,
          FILENAME 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 SENTENCE MUST BE IMPERATIVE NOT CONDITIONAL
5 -146 5 THIS VERB MUST BE PRECEDED BY PROCEDURE DEFINITION.
5 -147 2 OVERLAPPING MAY OCCUR BETWEEN THIS RECEIVING ITEM AND SENDING ITEM
5 -148 2 THIS RECEIVING ITEM MAY BE TRUNCATED ON 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 THIS RESULT MAY BE LEFT TRUNCATED.
5 -161 3 ILLEGAL RELATION BETWEEN INDEX AND EXPRESSION.
5 -162 4 THIS FEATURE ¬ FEATURE, NOT INCLUDED IN THE CURRENT COMPILATION LEVEL.
5 -163 3 EXPECTED WORD WAS "INTO" OR "END"
5 -164 1 SYNTAX CHECK DISCONTINUED
5 -165 1 SYNTAX CHECK 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 INDEXED-EXT.
5 -170 3 FILE IS NOT INDEXED
5 -171 3 COMPILER ERROR
5 -172 3 EXPECTED WORD WAS FIGURATIVE CONSTANT OR ALPHANUMERIC LITERAL
5 -173 3 ITEM IS NOT ALTERABLE IDENTIFIER
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 "COMP_" 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 OR IS NOT ELEMENTARY ITEM DESCRIBED AS A NUMERIC INTEGER
5 -180 3 EXPECTED WORD WAS "WHEN", "AT" OR "END".
5 -181 3 ITEM IS NOT ELEMENTARY ALPHABETIC, ALPHANUMERIC, OR NUMERIC EDITED OR A GROUP ITEM
5 -182 3 ITEM IS NOT FIGURATIVE CONSTANT, NONNUMERIC LITERAL OR IDENTIFIER
5 -183 4 COMPILER ERROR SUBROUTINE STACK OVERFLOWED

Code	Level	Message
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 CAN NOT 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 -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	")" MATCHING THIS "(" IS LACKING.
5 -205	3	"(" MATCHING THIS ")" IS LACKING
5 -206	3	NON NUMERIC IDENTIFIER SHOULD PRECEDE THIS OPERATOR
5 -207	3	CD NAME EXPECTED HERE
5 -208	3	THIS OPERAND SHOULD BE NON-ALPHABETIC DISPLAY IDENTIFIER OR GROUP ITEM WITHOUT A SIGNED ELEMENT.
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 ERROR.
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 INPUT DEVICE
5 -224	3	ITEM DOES NOT REFERENCE 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 CLAUSE OR INDEXED AND MUST HAVE SEQUENTIAL OR DYNAMIC ACCESS
5 -228	3	EXPECTED WORD WAS "EQUAL", "GRATER" 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	5	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 FOR ONE OR MORE INDEX NAMES
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
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 BE AN INTEGER RANGING IN VALUE FROM 0 THRU 99
5 -248 3 THIS IDENTIFIER MAY NOT BE A CONSTANT SECTION ITEM.
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 -255 5 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 5 FORBIDDEN USAGE OF ABBREVIATED RELATION.
5 -260 3 THIS FEATURE IS NOT IMPLEMENTED
5 -261 3 ITEM IS NEITHER "PROCEDURES" NOR A DATA-NAME.
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 ITEM IS NEITHER "+", "-", "(", A NUMERIC LITERAL OR A NUMERIC ELEMENTARY DATA ITEM.
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 SEARCH 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 AL_".
5 -279 5 THIS IDENTIFIER DOES NOT COMPLY TO THE RULE ON USAGE 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" IS IMPLIED FOR THE CORRESPONDING FILE.
5 -283 1 COMPARISON BETWEEN NUMERIC AND NONNUMERIC ITEMS.
5 -284 1 MOVING NONNUMERIC 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 COMP-1 ARGUMENTS).
5 -288 5 OLD TEMPORARY PRINTER CHANNEL SPECIFICATION USED INSTEAD OF MNEMONIC NAME.
5 -289 2 THIS FEATURE IS LEVEL-62 SPECIFIC. THE ITEM IS IGNORED.
5 -290 3 LEVEL 62 SPECIFIC 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 COMP-9 OR COMP-10 ITEM SHOULD NOT APPEAR IN THIS CONTEXT.

6 -1	3	AMBIGUOUS UNQUALIFIED REFERENCE.
6 -2	3	ITEM NOT DECLARED.
6 -3	3	AMBIGIOUS QJALIFIED 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, DATANAME, 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	COMPILER ERROR. NAME-STACK OVERFLOW FOR THIS REFERENCE.
6 -13	3	COMPILER ERROR. SUBSCRIPT-STACK OVERFLOW AT THIS ITEM.
6 -14	3	AN IDENTIFIER MUST NOT APPEAR MORE THAN ONCE IN A USING PHRASE.
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	3	REPLACEMENT ABORT. NAME TABLE BUFFER CONTAINS NO NEW ENTRIES FOR TWO CONSECUTIVE LOADS.
6 -20	3	MUST BE AN UNSIGNED INTEGER
6 -21	2	ANSI FORBIDS REL KEY BELONG TO A RECORD OF THE FILE TO WHICH IT IS A KEY
6 -22	3	MUST BE ALPHANUMERIC & NOT VARIABLE LENGTH
6 -23	3	MUST BELONG TO A RECORD ASSOCIATED TO THE FILE
6 -24	3	STATUS CAN ONLY BE 2 CH ALPHANUMERIC & NOT IN FILE, CONSTANT, LINKAGE SECTIONS
6 -25	2	STATUS KEY 3 ITEM IS NOT CONFORMED TO ANSI STANDARD.
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 77, 88, OR 01 LEVEL ENTRY
6 -28	3	RENAME OBJECT1 AND OBJECT2 AREA RANGE CONFLICT
6 -29	3	KEY FIELD IS TOO SMALL
6 -30	3	KEY LOC VALUE TOO LARGE
6 -31	3	KEY LOC IS OUTSIDE OF THE RECORD AREA
6 -32	3	CLASS NOT ALPHANUMERIC
6 -33	2	ITEM NOT ELEMENTARY
6 -34	3	FIELD TOO SHORT FOR RELATIVE KEY
6 -35	2	FIELD TOO LONG FOR RELATIVE KEY
6 -36	3	THE DATA-NAME REPLACED IN THE DEPENDING ON CLAUSE OF OCCURS MUST NOT TO BE SPECIFIED IN THE RANGE OF THE OCCURS
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 KEYS
6 -39	3	THIS REPORT GROUP HAS ALREADY BEEN SPECIFIED IN PREVIOUS "USE BEFORE REPORTING" CLAUSE.

```
6 -40  3  CONTROL ITEM MUST BE DATA NAME.
6 -41  3  CONTROL ITEM CANNOT BE OF VARIABLE LENGTH.
6 -42  3  CONTROL ITEM CANNOT HAVE AN OCCURS CLAUSE NOR BE
          SURBORDINATED TO A GROUP WHICH CONTAINS AN OCCURS
          CLAUSE.
6 -43  3  REPORT ITEM CAN ONLY BE USED FOR SUM COUNTER
          REFERENCE QUALIFICATION.
6 -44  4  THIS FEATURE IS A ¬ FEATURE NOT INCLUDED IN THE
          CURRENT COMPILATION LEVEL.
6 -45  3  SECONDARY KEY EXCEEDS 30 CHARACTER.
6 -46  3  SECONDARY KEY CANNOT HAVE THE SAME OFFSET AS THAT OF
          REACORD KEY.
6 -47  3  INVLID CONTROL ITEM.
6 -48  3  USE FOR DEBUGGING MUST NOT REFERENCE A USE FOR
          DEBUGGING PROC 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.
```

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 NOT ¬
7 -3	3	EXPECTED WORD "¬"
7 -4	3	DATANAMES AND INDEXNAMES NOT ALLOWED TOGETHER AS SUBSCRIPTS
7 -5	3	ITEM IS NOT ELEMENTARY NUMERIC INTEGER.
7 -6	3	ITEM IS NOT ALTERABLE IDENTIFIER.
7 -7	1	WHEN EXECUTING IN DEBUGGING MODE, THE SUBSCRIPT VALUE IN DEBUG ITEM WILL BE THAT AFTER THE STATEMENT IS EXECUTED.
7 -8	4	THIS FEATURE IS A LEVEL-64 FEATURE NOT INCLUDED IN THE CURRENT COMPILATION LEVEL.
7 -10	3	DATANAME DESCRIPTION CONTAINS "¬"
7 -11	4	COMPILER ERROR: UNEXPECTED TOKEN IN CORRESPONDING OPERAND.
7 -13	3	HIERARCHY ERROR IN CORRESPONDING OPERAND
7 -15	4	COMPILER ERROR: PREMATURE END OF FILE DURING CORR [illegible]TIO INITIALIZE STATEMENT
7 -16	3	OPERAND OF CORRESPONDING MUST BE GROUP NAME
7 -17	3	ITEM IS NOT POSITIVE INTEGER LITERAL.
7 -18	3	RELATIVE INDEXING REQUIRES UNSIGNED INTEGER LITERAL
7 -22	3	OPERAND OF INITIALIZE MAY NOT HAVE OCCURS DEPENDING ON.
7 -23	3	ILLEGAL OPERAND IN THE REPLACING CLAUSE OF INITIALIZE.
7 -25	2	INITIALIZE STATEMENT RESULTS IN NO MATCH.
7 -26	3	INITIALIZE SENDING OPERAND NOT LEGAL CATEGORY
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.

8	-1	1	PRINT PHASE WORKING SPACE EXHASTED, PART OF DIAG MESSEGES WILL NOT BE PROCESSED.
8	-21	3	NOT YET IMPLEMENTED.
8	-22	3	TOKEN AREA OVERFLOW.
8	-23	3	ALTER AREA OVERFLOW.
8	-24	3	PERFORM TABLE OVERFLOW.
8	-25	3	ALLOC-TABLE OVERFLOW.
8	-26	3	EGADG ERROR.
8	-27	3	EGADG1 ERROR.
8	-28	3	EGSTOG ERROR.
8	-29	3	EGBUTA ERROR.
8	-30	3	LITERALS WILL NOT BE COMPARED.
8	-31	3	ALTER ALLOCATION ERROR.
8	-32	3	BAD NUMBER OF PARAMETERS IN H_CBL_UGETG4.
8	-33	3	UNEXPECTED COMPARISON.
8	-34	3	ABBREVIATED CONDITION ERROR.
8	-35	3	EGGDBG ERROR.
8	-36	3	PERFORM ALLOCATION ERROR.
8	-37	3	PERFORM ERROR.
8	-38	3	BR LOCK ERROR.
8	-39	3	ALL REGISTERS LOCKED.
8	-40	3	VARIABLE LENGTH ERROR.
8	-41	3	MOVE ERROR.
8	-42	3	GR ALLOCATION ERROR.
8	-43	2	FLGR ERROR ¬.
8	-44	2	GR LOCK ERROR ¬.
8	-45	3	SEGMENT NUMBER ERROR.
8	-48	2	ERRONEOUS NUMBER OF PARAMETERS.
8	-61	4	WORK SPACE NOT AVAILABLE TO BUILD PCF TABLES.
8	-62	4	TOO MANY DATA/PROCEDURE NAMES TO BUILD PCF TABLES.
8	-63	4	COMPILER ERROR: UNKNOWN COMPILER GENERATED DATA-NAME: ¬.
8	-66	4	SEGMENT NUMBER LIMIT OF 128 HAS BEEN EXCEEDED FOR CODE SEGMENTS. GATHER SECTIONS.
8	-91	4	CULIB 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 128 ISN'S 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 ¬.
8	-98	4	COMPILER ERROR. DUPLICATE DEFINITION FOR TAG NUMBER ¬.
8	-99	4	COMPILER ERROR. INVALID EQUIVALENCE OF TAGS ¬.
8	-100	4	COMPILER ERROR. TAG NUMBER ¬ NOT DEFINED.

```
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 -40   2   ILLEGAL DSEGMAX OPTION : '¬'.
9 -41   2   ILLEGAL PSEGMAX OPTION : '¬'.
9 -42   2   SPECIFIED DSEGMAX OPTION EXCEEDS 4M BYTES.
9 -43   2   SPECIFIED PSEGMAX OPTION EXCEEDS 32K BYTES.
9 -44   2   ILLEGAL RESTRICT OPTION: '¬'.
9 -45   4   UNRECOVERABLE DIFFICULTY DUE TO SYSTEM ERROR.
9 -46   4   IMPOSSIBLE TO OPEN ¬.
9 -47   3   IMPOSSIBLE TO CLOSE ¬.
9 -48   4   UNRECOVERABLE DIFFICULTY DUE TO SYSTEM ERROR.
9 -49   4   COMPILER ERROR: ON SEQUENTIAL ¬. FILE IS OPENED INPUT.
9 -50   4   COMPILER ERROR: ON SEQUENTIAL ¬. FILE IS OPENED OUTPUT.
9 -51   4   COMPILER ERROR: ON SEQUENTIAL ¬. FILE IS CLOSED.
9 -52   4   COMPILER ERROR: ON SEQUENTIAL ¬. FILE IS EXHAUSTED.
9 -53   4   COMPILER ERROR: ON SEQUENTIAL ¬. INVALID FILE POINTER.
9 -54   4   COMPILER ERROR: ON SEQUENTIAL PUT. INVALID LENGTH(¬).
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 COMPILER WORK FILES.
9 -58   4   COMPILER ERROR ON DIRECT ¬. FILE IS OPENED.
9 -59   4   COMPILER ERROR ON DIRECT ¬. FILE IS CLOSED.
9 -60   4   COMPILER ERROR ON DIRECT ¬. FILE IS EXHAUSTED.
```

9 -61 4 COMPILER ERROR ON DIRECT SPUT. INVALID LENGTH(¬).

9 -62 4 COMPILER ERROR ON COMMON FILE. INVALID KEY NUMBER(¬).

9 -63 4 IMPLEMENTATION RESTRICTION. TOO MANY NAME IN AN 01.

9 -64 4 COMPILER ERROR ON DIRECT FILE. 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 FILE. BLOCK NUMBER ¬ ALREADY BLOCKED.

9 -68 4 COMPILER ERROR ON DIRECT UNBLOCK. FILE IS NOT BLOCKED.

9 -69 4 BACKING STORE IS FULL. TOO MANY JOBS RUNNING CONCURRENTLY.

9 -70 4 I/O ERROR ON DIRECT FILES.

9 -71 4 COMPILER ERROR ON DIRECT ¬. INVALID FILE POINTER.

9 -72 4 COMPILER ERROR ON DIRECT SGET. INVALID LENGTH(¬).

9 -73 4 COMPILER ERROR ON DIRECT DPUT. INVALID LENGTH(¬).

9 -74 4 COMPILER ERROR ON DIRECT DGET. INVALID LENGTH(¬).

9 -75 4 COMPILER ERROR ON DIRECT DREAD. INVALID LENGTH(¬).

9 -76 4 INVALID ¬ FILE.

APPENDIX C

$LINKER ERROR MESSAGES

ERROR MESSAGES ISSUED BY THE STATIC LINKER

No.	Message	Remarks
108	ERROR IN LMNAME PAPAMETER: WHEN COMFILE OR COMMAND IS MISSING	When a lmname is *, the lmname's are listed in a command or comfile parameter.
112	ERROR IN COMFILE PARAMETER: COMFILE MAY NOT APPEAR IN THE PRESENT CONTEXT (WHEN COMFILE IS PRESENT)	
112	ERROR IN COMFAC PARAMETER: COMFAC MAY NOT APPEAR IN THE PRESENT CONTEXT (WHEN COMFILE OR COMMAND IS PRESENT)	
112	ERROR IN ENTRY PARAMETER: ENTRY MAY NOT APPEAR IN THE PRESENT CONTEXT (WHEN COMFILE OR COMMAND IS PRESENT)	
112	ERROR IN PRTLIB PARAMETER: PRTLIB MAY NOT APPEAR IN THE PRESENT CONTEXT (WHEN PRTFILE IS PRESENT)	
201	MORE THAN 4 FATAL ERRORS	ONLY THE FIRST 4 FATAL ERRORS ARE DISPLAYED!
202	LINKER TABLE INITIALIZATION FAILURE	RC=00000000 PTR=XXXXXXXX
203	ERROR DURING OPEN	RC=XXXXXXXX (LIBRARY TYPE)
204	BUILD	"
205	OPENS IN	"
206	OPEN OUT	"
207	OPEN UPDATE	"
208	PUTX	"
209	CLOSES	"
210	NOTE	"
211	POINT	"
212	GET	"
213	PUT	"
214	CLOSE	"
215	MORE THAN 64 PROCESSES	
216	SYNTAX ERROR IN LM NAME	
217	ENTRY POINT NOT FOUND	
218	LINKER TABLE OVERFLOW	RC=00000000 PTR=XXXXXXXX
219	SEGMENT TABLE OVERFLOW	
220	ERROR DURING CHANGE NAME	RC=XXXXXXXX (LIBRARY TYPE)
221	CU INCORRECTLY FORMATTED	RC=(REC#)(LIB) UNEXPECTED NB OF RECORDS
222	ENTRY SEGMENT OVERFLOW (IMPLEMENTATION RESTRICTION)	NO MORE ENTRIES ARE AVAILABLE IN THE ENTRY SEGMENT. THE ENTRY SEGMENT IS LIMITED TO 256 ENTRIES
223	NO VALID SM NAME HAS BEEN FOUND	THE SM WHERE THE LKU IS TO BE STORED IS NOT DEFINED
224	NO VALID LKU ENTRY-POINT	
225	CONFLICT BETWEEN LKU AND EXISTING SUBFILE	THE LINKER TRIES TO REPLACE A LKU NOT DESCRIBED IN THE SM SUBFILE

226	OUTPUT LIBRARY OVERFLOW	NOT ENOUGH SPACE IN THE OUTPUT LIBRARY TO STORE THE PRODUCED MODULE.
227	SM DOES NOT EXIST, PLEASE SPECIFY ITS STN AND ESSTE	
228	INPUT LIBRARY NOT A CULIB	AN ASSIGNED INPUT LIBRARY IS NOT A CULIB (TYPE, RECFORM, RECSIZE) (SEE LIBMAINT GUIDE)
229	OUTPUT LIBRARY NOT A LMLIB	THE ASSIGNED OUTPUT LIBRARY IS NOT A LMLIB (TYPE, RECFORM, RECSIZE) (SEE LIBMAINT GUIDE)
230	OUTPUT LIBRARY NOT A SMLIB	THE ASSIGNED OUTPUT LIBRARY IS NOT A SMLIB (TYPE, RECFORM, RECSIZE) (SEE LIBMAINT GUIDE)
231	PRTLIB LIBRARY NOT A SL LIBRARY	SELF EXPLANATORY
232	SM ALREADY EXIST PLEASE DOES NOT SPECIFY ITS STN/ESSTE	SELF EXPLANATORY
233	ACCESS VIOLATION TO SYS.HSMLIB	IT IS FORBIDDEN TO LINK IN SYS.HSMLIB
401	UNKNOWN KEYWORD	UNEXPECTED OPTION OR UNKNOWN KEYWORD... THIS ILLEGAL STATEMENT IS IGNORED.
402	ILLEGAL MULTIPLE PARAMETER	A STATEMENT SUCH AS ENTRY,REALLSEG... HAS APPEARED MOTE THAN ONCE: THE FIRST SPECIFICATION IS USED
403	SYNTAX ERROR	SYNTAX ERROR ON A STATEMENT DURING SYNTAX ANALYSIS:ILLEGAL CHARACTERS FOR A PARAMETER ...THE STATEMENT IS IGNORED.
404	PARAMETER ERROR	ERROR IN A PARAMETER:INTEGER VALUE INSTEAD OF AN IDENTIFIER, ILLEGAL VALUE, IMCOMPATIBILITY WITH A PRECEEDING VALUE... THIS SPECIFICATION IS IGNORED.
405	OPTION ALREADY APPEARED	TWO PARAMETERS CONCERN THE SAME OBJECT IDENTIFICATION (MSEGAT,PLACE...) OR THE SAME FIELD (STACKI...) THE 1ST SPECIF. IS USED
406	CU NOT FOUND IN LIBRARIES	A STATEMENT SUCH AS GATE, MSEGAT... REFERS TO A CU THAT DOES NOT EXIST IN CU LIBRARIES (IMPLICIT OR SPECIFIED VIA $LINKER). THE STATEMENT IS IGNORED.
407	THE SPECIFIED CULIB IS NOT ASSIGNED	A STATEMENT SUCH AS REPLACE OR FETCH OR INCLUDE SPECIFIES A CU LIBRARY THAT IS NOT ASSIGNED.
408	ILLEGAL PARAMETER ACCORDING TO LINKTYPE	THE SPECIFIED PARAMETER CANNOT BE USED WITH THIS TYPE OF LINKAGE (ELM OR LKU) EX: LKUENT CANNOT BE USED WITH LINKTYPE=USER
1001	TOO MANY VACANT ENTRIES REQUESTED	THE USER (THRU DATA MANGMT OR VACSEG) ASKS FOR MORE VACANT ENTRIES THAN THERE ARE AVAILABLE ENTRIES LEFT IN SEG. TABLE. REMEMBER THAT AN ST IS LIMITED TO 256 ENTRIES.

Message	Explanation
1002 INITSIZE IN SOME SEGMENT EXCEEDED MAXSIZE	AT LEAST ONE SEGMENT HAS BEEN DECLARED WITH AN INITSIZE GREATER THAN MAXSIZE. THE MAXSIZE IS ADJUSTED TO INITSIZE
1003 SYMBMAP RECORD > 32K	SYMBOLIC PATCHING IMPOSSIBLE.
1004 NOLINK AND INCLEXT FOR A GLOBAL DATA, INCLEXT IGNORED	
1401 PRIVPECT: NO MATCHING DEF	THE NAME REFERENCED IN PRIVPECT PARAMETER HAS NOT BEEN FOUND IN LIBRARIES.
1402 PRIVPECT: MATCHING DEF IS A SYSDEF	THE NAME REFERENCED IN PRIVPECT PARAMETER IS A SYSTEM NAME.
1403 PRIVPECT: INVALID MATCHING DEF	THE FETCHED SYMDEF HAS NO STN,STE,D VALUE ASSIGNED BECAUSE ERROR OCCURRED WHEN PROCESSING IT.
1404 PRIVPECT: IMPROPER MATCHING DEF	NAME REFERRED IN PRIVPECT STATEMENT IS NEITHER A PROCEDURE NAME NOR A SEMAPHORE NAME.
1405 EXCEPTION: NO MATCHING DEF	THE NAME REFERENCED IN EXCEPTION PARAMETER HAS NOT BEEN FOUND IN LIBRARIES.
1406 EXCEPTION: MATCHING DEF IS A SYSDEF	THE NAME REFERENCED IN EXCEPTION PARAMETER IS A SYSTEM NAME.
1407 EXCEPTION: INVALID MATCHING DEF	THE FETCHED SYMDEF FOR EXCEPTION EITHER HAS NO STN,STE,D VALUE ASSIGNED OR IS NOT A PROCEDURE DESCRIPTOR SYMDEF.
1408 INCLUDE: NO MATCHING DEF	THE NAME REFERENCED IN INCLUDE PARAMETER HAS NOT BEEN FOUND IN LIBRARIES.
1601 ENTRY: NO MATCHING DEF	TASK ENTRY POINT HAS BEEN FOUND IN LIBRARIES.
1602 ENTRY: MATCHING DEF IS A SYSDEF	THE NAME DEFINED AS AN ENTRY POINT IS A SYSTEM NAME.
1603 ENTRY: IMPROPER MATCHING DEF	THE SYMDEF FOUND FOR ENTRY POINT IS A DATA SYMDEF.
1605 ENTRY: REALLOC RULES VIOLATION	ENTRY POINT IS IN A PROCESS PRIVATE SEGMENT.
1606 ENTRY: INVALID MATCHING DEF	THE SYMDEF FOUND FOR ENTRY POINT HAS NO STN,STE,D VALUE ASSIGNED BECAUSE AN ERROR OCCURRED WHILE PROCESSING THE SYMDEF OR JCL PARAMETERS.
1607 ENTRY ALREADY USED IN ENTRY SEGMENT	THE ENTRY SPECIFIED IN ESINDEX IS ALREADY USED.
1801 LARGE SEGMENT	
1802 SHARE LEVEL INCONSISTENT WITH ASSIGNMENT	
1803 SHRLEVEL=3 FOR NOT ASSIGNED INCLUDED SEGMENT	

Code	Message	Explanation
1804	ATTEMPT TO ASSIGN PROC.PRIV ATE SEG. AMONG INCLUDED SEGS	
1805	ALREADY USED ENTRY IN ASSIGNMENT	
2001	SOME INIT.VALUE RECORDS NOT USED	THE CU CONTAINS MORE INITIALIZATION VALUES THAN THE NEEDS EXPRESSED BY THE SYMDEFS.
2201	SEG REFERRED TO THRU ISN AND NAME IN JCL	THE USER HAS DEFINED TWO SETS OF ATTRIBUTES FOR A SEGMENT IN JCL. IN THE 1ST, THE SEGMENT HAS BEEN REFERENCED THRU ISN, IN THE SECOND, IT HAS BEEN REFERENCED THRU NAME. WHEN THE SAME ATTRIBUTE APPEARS IN BOTH DEFINITIONS, THE LAST DEFINED VALUE IS USED.
2202	INVALID SHARE LEVEL	THE SHARABILITY LEVEL FOR A SEGMENT IS EQUAL TO 0 OR 1.
2203	SIZE INCONSISTENT WITH A PREVIOUS DEFINITION	THE SEGMENT DEFINITION IN THE CURRENT CU SPECIFIES AN (INITIAL) SIZE OR A MAXIMUM SIZE, BUT THE CURRENT SIZE OF THE SEGMENT (SUM OF THE SIZES OF DATA ALREADY ALLOCATED IN THE SEGMENT) IS GREATER THAN THIS SPECIFIED SIZE.
2204	ATTRIBUTES INCONSISTENT WITH A PREVIOUS DEFINITION	A GLOBAL SEGMENT HAS BEEN DEFINED IN TWO CU'S WITH DIFFERENT ATTRIBUTES.
2205	JCL GATE FOR NON GATEABLE CU	A JCL "GATE" COMMAND EXISTS FOR A CU WHOSE 1ST SEGMENT HAS NO GATE DOUBLE WORD PREFIX.
2206	SHARE LEVEL CONFLICT (AFTER PREV. USE OF PLACE)	AN ANTICIPATED PLACEMENT OCCURRED FIRST WITH SEG(DEFAULT) ATTRIBUTES CONFLICTING WITH THE CURRENT DESCRIPTION.
2207	SIZE SPECIFICATION (MSEGAT) EXCEEDED	THE SIZE SPECIFICATION FOR A HARDWARE PROTYPE IS LESS THAN ITS SIZE VALUE IN THE DESCRIPTION.
2401	EXISTS AS SYSDEF	THE DATA DECLARED IN THE CURRENT CU IS ALSO A SYSDEF. THE SYSDEF IS USED IN SYSLINK ENVIRONMENT OR IF THE 1ST REFERENCE DID NOT HAVE INITIALIZATIONS; ELSE THE SYMDEF IS USED.
2402	REALLOC RULES VIOLATION	
2403	A PREVIOUS REF FOR THIS DATA STATED: DEF CANNOT EXIST	AN INITIALIZATION APPEARS FOR A DATA WHEREAS ANOTHER DECLARATION FOR THE SAME DATA HAS SAID: DATA CANNOT BE INITIALIZED.
2404	ATTRIBUTES INCONSISTENT WITH A PREVIOUS DEFINITION	THE DATA DECLARED IN THE CURRENT CU HAS ATTRIBUTES DIFFERENT FROM THE ONE SPECIFIED IN ANOTHER CU FOR THE SAME DATA.
2405	ALL ENTRY POINTS MSUT BE SIMULTANEOUSLY DECLARED NOLINK	A CU CONTAINS SEVERAL ENTRY POINTS, SOME OF THEM ARE SPECIFIED IN A NOLINK PARAMETER,OTHERS ARE NOT. THUS SOME REFERENCES TO THIS PROCEDURE STAY UNRESOLVED.

2407	SUBITEM CANNOT BE LINKED IF ITEM IS NOLINK	A DATA SPECIFIED IN A NOLINK PARAMETER CONTAINS SUBITEM EXTERNALLY KNOWN: REFERENCES TO THIS SUBITEM STAY UNRESOLVED.
2409	NAME ALSO USED FOR A DIFFERENT ENTITY	
2410	THIS CATALOGED ENTITY ALREADY EXISTS	
2411	INVALID SHARE LEVEL	TYPE 3 SLFICB SEGMENT.
2602	MULTIPLE INITIALIZATION FOR PTR	AN ENTRY VARIABLE IN AN EXTERNAL DATA HAS BEEN INITIALIZED IN DIFFERENT CUS WITH DIFFERENT VALUES.
2603	REALLOC RULES VIOLATION	
2604	UNRESOLVED REFERENCE	THE REFERENCE MAY BE DYNAMICALLY RESOLVED IN A SM.
2802	REALLOC RULES VIOLATION	
2803	CATALOGED MATCHING DEF	USED FOR WANT OF A GLOBAL DATA.
2804	IMPROPER MATCHING DEF	A REFERENCE TO DATA LEADS TO A PROCEDURE DEFINITION.
2805	ILLEGAL MATCHING NO SYMDEF SHOULD EXIST	THE CURRENT REFERENCE STATES: SYNDEF CANNOT EXIST, BUT A SYMDEF HAS BEEN ALREADY FOUND FOR THIS DATA.
2806	INVALID MATCHING DEF	THERE ALREADY EXISTS A DEFINITION FOR THIS DATA BUT NO STN,STE,D VALUE IS ASSOCIATED TO IT BECAUSE ERROR OCCURRED WHILE PROCESSING THE 1ST DEFINITION.
2807	CONFLICT BETWEEN REF-DEF ATTRIBUTES	A PREVIOUS DECLARATION OF THIS DATA SPECIFIED A DIFFERENT VALUE FOR DATA LENGTH.
2808	CONFLICT BETWEEN REF-DEF ATTRIBUTES	A PREVIOUS DECLARATION FOR THIS DATA SPEFIED A DIFFERENT VALUE FOR DATA ATTRIBUTES.
2809	CONFLICT BETWEEN REF-DEF ATTRIBUTES	A PREVIOUS DECLARATION FOR THIS DATA SPECIFIED A DIFFERENT CONTAINING SEGMENT.
2810	ILLEGAL MATCHING	CONTRADICTION BETWEEN THE CURRENT REFERENCE SAYING: SYMDEF MIGHT EXIST, AND A PREVIOUS REFERENCE (TO THE SAME DATA) SAYING: SYMDEF CANNOT EXIST.
2811	A DATA SYMDEF MATCHES A NON DATA SYMDEF OR A SYSDEF OR A SUBITEM SYMDEF	FOR INSTANCE,A FORTRAN LABELED COMMON SYMDEF MATCHES A SYMDEF THAT DOES NOT CORRESPOND TO A BLOCK DATA.
2812	NAME ALSO USED FOR A DIFFERENT ENTITY	
3001	NO MATCHING DEF	A REFERENCE IS MADE TO A DEFINITION WHICH DOES NOT EXIST IN LIBRARIES.
3002	REALLOC RULES VIOLATION	

3005 CONFLICT BETWEEN REF-DEF ATTRIBUTES	REFERENCE TO A PROCEDURE WITH A NUMBER OF ARGUMENTS DIFFERENT FROM THE NUMBER OF PARAMETERS DEFINED IN THE PROCEDURE.
3006 INVALID MATCHING DEF	THE SYMDEF FETCHED HAS NO STN,STE,D VALUE ASSIGNED BECAUSE ERROR OCCURRED WHILE PROCESSING IT.
3007 CONFLICT BETWEEN REF-DEF ATTRIBUTES	THE DATA LENGTH (FOR DATA) OR ARGUMENTS SIZES (FOR PROCEDURES) SPECIFIED IN THE REFERENCE ARE DIFFERENT FROM THE ONES SPECIFIED IN THE DEFINITION.
3008 CONFLICT BETWEEN REF-DEF ATTRIBUTES	THE ATTRIBUTES SECIFIED FOR THE DATA OR THE ARGUMENTS IN THE REFERENCE ARE DIFFERENT FROM THE ONES IN THE DEFINITION.
3009 CONFLICT BETWEEN REF=DEF ATTRIBUTES	
3010 IMPROPER MATCHING DEF	A REFERENCE TO DATA LEADS TO A PROCEDURE OR A NON CATALOGUED DATA.OR A REFERENCE TO PROCEDURE LEADS TO A DATA DEFINITION.
3401 GLOBAL DATA SEGMENT OVERFLOW	THE DATA CURRENTLY PROCESSED CANNOT BE ENTIRELY STORED IN THE GLOBAL SEGMENT EITHER BECAUSE THE DATA SIZE IS GREATER THAN THE MAXIMUM SIZE FOR MULTIPLE SEGMENTS, OR BECAUSE THE SUM OF SIZES OF DATA ALLOCATED IN SEGMENT BECOMES GREATER THAN THE LIMIT SIZE FOR NON-MULTIPLE SEGMENTS.
3402 INCORRECT CIS NUMBER	A GLOBAL DATA IS SAID TO BE CONTAINED IN A SEGMENT WHICH IS NOT A GLOBAL SEGMENT (SEGMENT "TO BE INVENTED" BY THE LINKER).
3403 SIZE SPECIFICATION(MSEGAT) EXCEEDED	THE CURRENT SIZE BECOMES GREATER THAN THE SIZE SPECIFICATION FOR THIS GLOBAL SEGMENT.
3404 REALLOC RULES VIOLATION	
3405 SIZE INCONSISTENT WITH A PREVIOUS DEFINITION	A MAXIMUM SIZE HAS BEEN SPECIFIED WHICH IS ALREADY EXCEEDED BY THE CUMULATED SIZES OF DATA ALLOCATED IN THE SEGMENT.
3801 CROSS REFERENCE LIST HAS BEEN ABORTED (OVERFLOW)	WARNING: THE CROSS REFERENCE LIST CAUSED A LINKER TABLE OVERFLOW.
3802 CU REFERENCED IN JCL HAS NOT BEEN LINKER	A CU HAS BEEN REFERENCED IN A JCL STATEMENT BUT THIS CU NEVER APPEARED DURING THE LINKAGE PROCESSING (NO REFERENCE TO THE PROCEDURE HAS BEEN MADE).
3803 REFERENCE NEVER OCCURRED	THE JCL ASKED FOR THE REPLACEMENT OF A REFERENCE TO A BY A REFERENCE TO B, BUT NO REFERENCE TO A APPEARED IN THE SCOPE OF REPLACE.
3804 THIS STATEMENT HAS NOT BEEN USED	THE ENTITY INVOLVED IN A PLACE OR MSEGAT OR FETCH COMMAND NEVER APPEARED DURING THE LINKAGE.

3805 TASK DEFINITION NEVER OCCURRED	THE JCL TOLD ABOUT A TASK BUT HAS NEVER DEFINED IT.
4401 SEG DEFINED ONLY BY PLACE STAT.	THE JCL ASKED FOR A PLACEMENT IN A SEGMENT FOR WHICH NO DESCRIPTION WAS FOUND IN ANY CU.
4402 ZERO-LENGTH SEGMENT CONTAINS DATA	
5201 REFERENCE NOT FOUND IN CU	THE JCL ASKED FOR THE REPLACEMENT OF A REFERENCE TO A BY A REFERENCE TO B IN A GIVEN CU, BUT NO REFERENCE TO A APPEARED IN THE CU.
5202 MSEGAT/CU. SEG NOT USED	A SEGMENT OF A GIVEN CU REFERENCED IN JCL PARAMETERS THRU SEGMENT NAME OR INTERNAL SEGMENT NUMBER DOES NOT EXIST IN CU; OR PARAMETERS THRU GLOBLSEG WERE PREFERRED TO.
5601 MORE THAN 9 ERRORS FOR THAT ENTITY	THE LINKER ONLY DISPLAYS THE FIRST NINE ERRORS DISCOVERED WHEN PROCESSING AN ENTITY.

INDEX

NOTES: Main references are underlined. Entries beginning with nonalphabetic characters are classified according to the first alphabetic character of each entry.

A

Abnormal compiler termination 2-47
ACEPT statement 11-11
Alphabet 12-06
Alter facility 2-16.2
Alter listing 2-28
ALTERNATE KEYS 9-14
American Standards Assoc. Format 10-01
ANSI 74 2-10
APPLY NO-RESIDENT-INDEX clause 9-14
APPLY NO-SORTED-INDEX clause 9-14
ASA 10-01
$ASSIGN statement 9-01
Asterisk convention 2-06

B

Backing store 2-13
Banner page 3-08
BFAS 9-13
BOTTOM 11-06

C

CALL IDENTIFIER statement 6-05
CALL statement 4-06, <u>6-01</u>, 6-03
Called program 6-01
Calling program 6-01
CANCEL statement 6-01, <u>6-05</u>
Card identifier 2-06
Card punching 11-09
Card reading 11-07
CARDID parameter 1-05, <u>2-06</u>
CASEQ parameter 2-08
Cassettes 11-17

Checkpoint/restart 12-05
CKSEQ parameter 2-08
CLOSE REEL/UNIT statements 9-09, 9-10
CLOSE WITH LOCK statement 9-08
$COBOL statement 2-01
COBOL file-name 6-08, 9-01
COBOL segment number 3-01, 7-02
CODAPND parameter 2-08, 7-06, 7-07
CODE SET clause 12-07
COLLATING SEQUENCE phrase 12-07
COMFILE parameter 2-04, 2-17, 3-04
Command file 2-16.2
COMMAND parameter 3-04
Communications 12-29
Compilation 2-01
COMPILE command 2-17, 2-19
CONTCHAR parameter 1-07
Control record 10-02
COPY statement 2-10, 2-16, 2-29
Cross-reference listing 2-09, 2-37, 4-12
CULIB parameter 2-08

D

Data map and proc.def.listing 2-11, 2-37, 4-12
Data types 5-01
DCARDID parameter 2-06
DCLXREF parameter 2-09
DDEBUGMD parameter 2-09
DEBUG parameter 2-09, 4-02, 4-04
DEBUG-ITEM 4-01
Debugging code 4-01
DEBUGMD parameter 2-09, 4-02
DECLARATIVE 4-01
Device oriented format 10-01
DIAGIN parameter 2-10, 2-11
Diagnostic 2-34, 3-14
DISPLAY data items 5-02
DISPLAY statement 11-11, 11-13
DOF 10-01
DSEGMAX parameter 2-10, 7-06
DUMMY parameter 9-06
Dump analysis 4-04
DUMP parameter 4-04

E

Edited sysout format 11-02, 11-09, 12-18
Editor request 2-16.2
Efficiency 8-01
ENTRY command 3-05
ENTRY parameter 3-03, 6-02
Epilogue 2-24
Error message 2-34, 3-14, 4-16

EXAMINE statement 12-30
Exception 4-06
Exception message 4-16
Execution 4-01
EXIT statement 6-01
Expanded source listing 2-28
EXPLIST parameter 2-10, 2-29
External line number 2-33
EXTERNAL phrase 6-01, 6-04, 6-08
External-file-name 9-01

F

File concatenation 9-12
File names 9-01
File organization 9-04
FILE SECTION 12-07
FILE STATUS 9-15
Fixed-point binary 5-05
Floating-point binary 5-05
FOOTING 11-06
Form control 11-05
FORTRAN programs 6-06
FSN parameter 9-11

G

GENERATE statement 12-12
Group information listing 3-08

H

H-2000 9-14
HFAS 9-13
HIGH-VALUE 12-09

I

I-O-CONTROL paragraph 12-05
INCLUDE command 3-05
Included compile units listing 3-08
INDEX data item 5-06
Indicator area 1-06
INFILE parameter 1-10, 2-04
INITIATE statement 12-12
INLIB parameter 2-04
INLIBn parameter 2-04, 2-16.1
Input enclosure 1-01
INSPECT statement 12-30
Instruction counter 4-10
Interactive operation facility 1-03

Interactive terminal line format 1-10
Intermediate results 12-26
Internal line number 2-29, 2-33
Internal segment number 3-01, 3-11, 4-11, 4-12, 7-10
Internal-file-name 6-08, 9-01
IOF 1-03

JK

JCL STATUS 12-03
Job occurrence report 4-15
Job occurrence report summary 2-46
Journalization 12-05
$JUMP 2-15, 12-03, 12-05

L

Language type 1-05, 2-07
$LET 12-05
LEVEL parameter 2-10, 2-21
LEVEL 62 2-22
LEVEL 64 9-14
$LIB 2-04, 2-16, 3-02, 3-06
$LIBALLOC 1-02, 3-02
$LIBMAINT 1-02, 2-04, 10-02, 10-06
Library 2-16
Library member text format 1-07
Limits 2-21
LINAGE clause 11-06
LINE-COUNTER 12-11
Linkage report 3-08
LINKAGE SECTION 6-01, 6-03
$LINKER statement 3-01
$LINKER segment number 3-01, 4-12
Linking 3-01
LIST parameter 2-11
Listings ($COBOL) 2-24
Listings ($LINKER) 3-08
Load-module-name parameter 3-02
LOW-VALUE 12-09

M

Main program 6-02
Map listing 2-37
MAP parameter 2-11
Maximum data segment size 2-10, 7-06
Maximum procedure segment size 2-10, 7-06
$MERGE 12-01
MERGE statement 12-01, 12-07
Multi logical unit files 9-09
Multiple file tape volumes 9-11
Multivolume files 9-09

N

Naming convention 1-01, 3-02
NCARDID parameter 1-05, 2-06
NCASEQ parameter 2-08
NCKSEQ parameter 2-08
NCLIST parameter 2-11
NCODAPND parameter 2-08
NDCLXREF parameter 2-09
NDEBUG parameter 2-09
NDEBUGMD parameter 2-09, 4-02
NDIAGIN parameter 2-10
NEXPLIST parameter 2-10
NLIST parameter 2-11
NMAP parameter 2-11
NOBJ parameter 2-11
NOBSERV parameter 2-11
NOPT parameter 2-13
NRESIDX parameter 9-15
NWARN parameter 2-13
NXREF parameter 2-15

O

OBJ parameter 2-11
Object code 2-24
OBJECT-COMPUTER paragraph 12-07
OBSERV parameter 2-11
ON SIZE ERROR 12-29
Optional files 9-04
OPTIONAL parameter 9-06
OPTIONS parameter 12-03, 12-10
ORGANIZATION clause 9-14
OUTLIB parameter 3-03
Output writer 10-07, 11-01
Overriding rules 9-02

PQ

Packed decimal 5-03
PAGE-COUNTER 12-11

PCF 4-03
Perform/alter bucket listing 2-11, 2-37
Performance 8-01
PICTURE clause 5-01
$POOL statement 9-08, 9-10, 9-11, 9-13
Printed output ($LINKER) 3-08
Printed output ($COBOL) 2-24
Printing 11-01
Procedure map listing 2-11, 2-37, 4-11
Process control block 4-06
Process control structure 4-05
Process group control structure 4-05

Program checkout facility 2-09, 4-01, 4-03
PROGRAM COLLATING SEQ.clause 12-07
Prologue 2-24
Protection ring 4-06
PRTFILE parameter 2-12
PRTLIB parameter 2-12
PSEGMAX parameter 2-10, 7-06
Punched card format 1-05

R

Record length 9-03
Reference format 1-04
REPEAT parameter 12-05
REPLACE statement 2-10, 2-29
Report writer 6-08, 12-11
Representation of data 5-01
RERUN clause 12-05
RESIDX parameter 9-15
Return code 9-15
Run-time package 12-29

S

SARF 10-01
Search path 2-04, 3-02, 3-06
SEARCH statement 12-23
Segment 2-08
Segment list 2-46, 3-11, 4-11
Segment number 3-01
Segment table entry 4-05
Segment table number 4-05
SEGMENT-LIMIT 7-03
Segmentation 7-01
Sequence number 1-06, 10-02
Serial compilation 2-19
Serial linkage 3-06
SET statement 12-20
Severity value 2-15
$SORT 12-01
SORT statement 12-01, 12-07
$SORTIDX 9-14
$SORTWORK 12-02
Source library 1-02
Source listing 2-28
SOURCE parameter 2-04, 2-19
SPECIAL-NAMES paragraph 11-11, 11-14, 12-04, 12-06
Sra 4-11
SSF 10-01
Stack 4-06
Stack frame 4-10
Standard access record format 10-01
STATUS 2-15
Ste 4-05, 4-11

STEPOPT parameter 2-13, 3-06
Stn 4-05, 4-11
STOP LITERAL statement 11-11, 11-16
STOP RUN statement 6-02
Stream reader 1-06, 10-02, 10-06, 11-07
Structured programming 7-03
SUBOPT parameter 2-13, 8-02
Subscripts 12-20
Summary page 2-45
SUPPRESS statement 12-12
Swapping 7-01
Switches 12-04
SYMREF 3-08.1
SYSIN 1-06, 10-02, 10-06, 11-07
SYSOUT 2-12, 11-01, 11-09
$SYSOUT 10-07, 11-04, 11-09
System standard format 10-01

T

Table handling 12-20
Task listing 3-08
TDS 2-08
TEMP 2-08
TERMINATE statement 12-13
TOP 11-06
Transaction processing routine 2-08
TYPE parameter 1-06, 1-08, 10-02

U

UFAS 9-13
UFF 9-14
Unit record files 11-01
Unpacked decimal 5-02
Updating source program 1-04
$URINIT 11-05
USAGE clause 5-01
USE AFTER ERROR PROCEDURE SECTION 9-15
USE FOR DEBUGGING statement 4-01
USE statement 12-12
USING phrase 6-01, 6-03

V

VACSEG command 3-05
Vertical format unit 11-05
$VOLPREP 11-20
VFU 11-05
Virtual memory management 7-01

W REQUEST 1-08
WARN parameter 2-13
WITH CODE clause 12-18
WITH DEBUGGING MODE clause 2-09, 4-02
WITH SARF phrase 10-04
WITH SSF phrase 10-04
WORKN parameter 2-13
$WRITER 10-07, 11-04, 11-09

HONEYWELL INFORMATION SYSTEMS
Technical Publications Remarks Form

TITLE	SERIES 60 (LEVEL 64) GCOS COBOL User Guide Addendum A	ORDER NO.	AQ63-01A
		DATED	JUNE 1979

ERRORS IN PUBLICATION

SUGGESTIONS FOR IMPROVEMENT TO PUBLICATION

Your comments will be promptly investigated by appropriate technical personnel and action will be taken as required. If you require a written reply, check here and furnish complete mailing address below. ☐

FROM: NAME ______________________ DATE ____________

TITLE ______________________

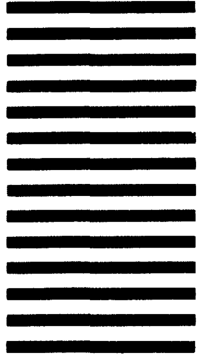
COMPANY ______________________

ADDRESS ______________________

____

PLEASE FOLD AND TAPE—
NOTE: U. S. Postal Service will not deliver stapled forms

NO POSTAGE
NECESSARY
IF MAILED
IN THE
UNITED STATES



BUSINESS REPLY MAIL

FIRST CLASS PERMIT NO. 39531 WALTHAM, MA02154

POSTAGE WILL BE PAID BY ADDRESSEE

HONEYWELL INFORMATION SYSTEMS
200 SMITH STREET
WALTHAM, MA 02154

ATTN: PUBLICATIONS, MS486

Honeywell