DX10

COBOL Programmer's Guide

Part No. 2270521-9701 *C
March 1985

TEXAS INSTRUMENTS

© 1980, 1982, 1983, 1985, Texas Instruments Incorporated. All Rights Reserved.

Printed in U.S.A.

No part of this publication may be reproduced, stored in a retrieval system, or transmitted, in any form or by any means, electronic, mechanical, photocopying, recording, or otherwise, without the prior written permission of Texas Instruments Incorporated.

MANUAL REVISION HISTORY

DX10 COBOL Programmer's Guide (2270521-9701)

Original Issue . . . . . December 1980
Revision . . . . . May 1982
Change 1 . . . . . October 1983
Revision . . . . . March 1985

The total number of pages in this publication is 298.

The computers, as well as the programs that TI has created to use with them, are tools that can help people better manage the information used in their business; but tools—including TI computers—cannot replace sound judgment nor make the manager's business decisions.

Consequently, TI cannot warrant that its systems are suitable for any specific customer application. The manager must rely on judgment of what is best for his or her business.

Preface

This manual contains information about the Texas Instruments version of COBOL (COmmon Business Oriented Language), which is designed to operate on Texas Instruments computers. This information supports the experienced programmer in developing COBOL programs intended for execution under the DX10 Operating System. For additional descriptions of COBOL, refer to the *COBOL Reference Manual*.

This manual contains the following sections and appendices:

Section

1 Introduction — Describes DX10 as it relates to COBOL and the operating system environment. This introduction also includes an overview of the processes necessary to create and execute a COBOL program (task) and includes notations that are used to describe commands in this manual.

2 Operating System Concepts — Describes features related to program development. Includes description of interactive tasks and batch execution, the System Command Interpreter (SCI), directory and file structure, pathnames, access names, and synonyms.

3 Building a COBOL Source Program Module — Discusses how to build a COBOL program source module, beginning with directory and file development, and how to use the Text Editor utility.

4 Compilation — Explains how a COBOL source program module is compiled and discusses compiler completion codes and error messages.

5 Link Edit — Explains the link editing process, COBOL segmentation, overlays, and installation of COBOL task and procedure segments. Includes information on memory mapping and the COBOL run-time interpreter.

6 Execution — Discusses execution of COBOL object modules, linked object modules, and program images. Provides necessary SCI commands, completion codes, and error messages.

7 Debugging — Discusses COBOL debugging for COBOL routines and the operating system debugging for assembly language object modules that are linked to a COBOL object module.

8 Calling Subroutines — Describes the process for calling COBOL and assembly language modules.

9 Interfacing to Productivity Tools — Introduces the productivity tools that can interface with COBOL and explains how these tools can be linked with COBOL object modules.

10 Using SCI Command Procedures to Execute COBOL Tasks — Describes how to design a system to interact with application environment processors and SCI.

11 COBOL Device-Dependent Attributes — Describes the ACCEPT/DISPLAY command option that allows access to function keys, low volume Input/Output (I/O), and graphic I/O.

12 Error Processing — Describes the COBOL file status data item and error processing under program control.

13 Optimizing Run-Time Performance — Discusses various ways to optimize COBOL code.

Appendix

A Keycap Cross-Reference — This appendix contains specific keyboard information to help the user identify individual keys on any supported terminal.

B COBOL Compiler Error Messages — Lists COBOL user and system compiler error messages.

C COBOL Run-Time Error Messages — Lists COBOL user and system run-time error messages.

D COBOL Subroutine Library Package — Describes COBOL subroutine library modules.

E COBOL Compiler Listing Format — Gives example of the results from using the M, O, and X options on the COBOL compiler.

In addition to the software manuals shown on the frontispiece, the following documents contain information related to this manual:

Title	Part Number
COBOL System Design Document	2250953-9901
SCI: A Self-Study Approach to Writing Command Procedures and Batch Streams	2267649-0001

Contents

Paragraph	Title	Page
	1 — Introduction	
1.1	COBOL	1-1
1.2	A COBOL Program Development Overview	1-1
1.3	SCI Command Prompt Format and Notation	1-8
1.3.1	Command Name	1-8
1.3.2	Command Prompts Returned	1-9
1.3.3	Type of Response Expected	1-9
1.3.3.1	Initial Values	1-9
1.3.3.2	Default Values	1-9
	2 — Operating System Concepts	
2.1	Introduction	2-1
2.2	Task Structure	2-1
2.2.1	Interactive Tasks	2-1
2.2.2	Batch Streams	2-1
2.3	Using SCI	2-2
2.3.1	SCI Description	2-2
2.3.2	Entry of SCI Commands in VDT Mode	2-2
2.3.3	Examples of Using SCI	2-2
2.3.3.1	The Show Background Status (SBS) Command	2-2
2.3.3.2	The List Directory (LD) Command	2-3
2.3.4	Batch Use of SCI	2-3
2.3.4.1	Batch Stream Format	2-4
2.3.4.2	Batch Command Format	2-4
2.3.4.3	Interactive Execution of Batch Streams	2-5
2.3.4.4	Entering Programs From Sequential Devices	2-5
2.4	Directory and File Structure	2-6
2.4.1	Establishing Volume Names	2-6
2.4.2	Establishing Directories	2-6
2.4.3	Establishing Files	2-7
2.5	Pathnames and Access Names	2-8
2.6	Synonyms	2-9
2.7	File Types	2-9
2.7.1	Sequential Files	2-9
2.7.1.1	Sequential File Attributes	2-10
2.7.1.2	Creating Sequential Files	2-10
2.7.2	Relative Record Files	2-14
2.7.2.1	Relative Record Attributes	2-14

Paragraph	Title	Page
2.7.2.2	Creating Relative Record Files	2-15
2.7.2.3	Special Types of Relative Record Files	2-20
2.7.3	Key Indexed Files (KIF)	2-20

3 — Building a COBOL Source Program Module

3.1	General	3-1
3.2	Directory and File Preparation	3-1
3.3	Alternate Directory Structures	3-2
3.3.1	Organization by Programs	3-2
3.3.2	Organization by File Type	3-2
3.4	Creating Directories and Files	3-2
3.5	Building the Program Module Via the Text Editor	3-3

4 — Compilation

4.1	General	4-1
4.2	Compiler Execution	4-1
4.2.1	Execute COBOL Compiler in Foreground (XCCF)	4-1
4.2.2	Execute COBOL Compiler in Background (XCC)	4-7
4.3	Compiler Output	4-8
4.4	Compiler Completion Codes	4-8
4.5	Compiler Error Messages	4-8
4.6	Compiler Limitations	4-8

5 — Link Edit

5.1	General	5-1
5.2	Object Modules	5-2
5.2.1	Differences in the Treatment of Shareable Vs. Reentrant Modules	5-2
5.2.2	COBOL Object Modules	5-2
5.3	Program Mapping	5-4
5.4	Program Files	5-5
5.4.1	Segments	5-5
5.4.1.1	Task Segments	5-6
5.4.1.2	Procedure Segments	5-6
5.4.2	Overlays	5-10
5.4.3	COBOL Module Segmentation	5-10
5.5	Creating Linked Object Modules	5-12
5.6	Creating Program Images	5-13
5.6.1	COBOL Run Time	5-14
5.6.2	Linking a Single Procedure Segment With a Single Task Segment	5-14
5.6.3	Linking a Single Procedure Segment With Multiple Task Segments	5-15
5.6.4	Linking Two Procedure Segments With a Single Task Segment	5-15
5.6.5	Linking Two Procedure Segments With Multiple Task Segments	5-19
5.6.6	Overlay Structures	5-22

Paragraph	Title	Page
5.6.7	Sharing Main Program Module	5-24
5.6.8	Linking a Single Procedure One Segment and Multiple Procedure Two Segments	5-24
5.6.9	Linking a Single Procedure Segment With a Single Task Segment on a User Program File	5-26
5.6.10	Installing Program Images From a Relative File	5-26
5.7	Linking Libraries	5-28
5.8	Linking Limitations	5-29

6 — Execution

6.1	General	6-1
6.1.1	Use of a Synonym in the COBOL Select Clause	6-1
6.2	Object Modules Execution	6-1
6.2.1	Execute COBOL Program in Foreground (XCPF)	6-2
6.2.2	Execute COBOL Program in Background (XCP)	6-4
6.3	Execution Completion Codes and Run-Time Error Messages	6-4
6.4	Program Image Execution	6-5
6.4.1	Execute COBOL Task in Foreground (XCTF)	6-5
6.4.2	Execute COBOL Task in Background (XCT)	6-7
6.5	Execution Completion Codes and Run-Time Error Messages	6-7
6.6	Program Termination Messages	6-8

7 — Debugging

7.1	Debug Mode	7-1
7.2	Debugging a COBOL Module	7-1
7.2.1	Activating the Debugger	7-1
7.2.2	COBOL Debug Commands	7-3
7.2.2.1	Assign Address Stop Command (A)	7-7
7.2.2.2	Dump Data Item Command (D)	7-8
7.2.2.3	Exit Debug Mode Command (E)	7-9
7.2.2.4	Change Program Location Command (L)	7-10
7.2.2.5	Modify Data Item Command (M)	7-11
7.2.2.6	Quit Execution Command (Q)	7-14
7.2.2.7	Resume Program Execution Command (R)	7-14
7.2.2.8	Execute Next Single Statement Command (S)	7-14
7.2.2.9	Undo Address Stop Command (U)	7-15
7.2.2.10	Write Screen to Message File Command (W)	7-15
7.3	Debugging of Assembly Language Subroutines Linked to COBOL Programs	7-16

8 — Calling Subroutines

8.1	General	8-1
8.2	COBOL Subroutine Library Package	8-1
8.3	Assembly Language Subroutines	8-3

Paragraph	Title	Page
	9 — Interfacing to Productivity Tools	
9.1	General	9-1
9.2	TIFORM	9-1
9.3	Sort/Merge	9-7
9.4	Database Management System	9-15
9.4.1	DBMS-990 Features	9-16
9.4.2	DBMS-990 User Interface	9-16
9.4.3	Linking DBMS-990 and COBOL Modules	9-16
9.5	Query-990	9-31
9.6	Communications	9-35
9.7	Communication Equipment	9-35
9.8	3780 Emulator Communications Software	9-35
	10 — Using SCI Command Procedures to Execute COBOL Tasks	
10.1	General	10-1
10.2	SCI Command Procedure Elements	10-1
10.3	Example Command Procedures	10-2
10.3.1	Example 1	10-2
10.3.2	Example 2	10-3
10.3.3	Example 3	10-6
	11 — COBOL Device-Dependent Attributes	
11.1	Function Keys	11-1
11.2	Low Volume Input/Output (I/O)	11-2
11.3	Graphic Input/Output	11-8
	12 — Error Processing	
12.1	General	12-1
12.2	File I/O Status	12-1
12.3	File I/O Status Values	12-3
12.4	Use of Declaratives	12-8
	13 — Optimizing Run-Time Performance	
13.1	General	13-1
13.2	Object Size Considerations	13-1
13.3	Arithmetic Operations	13-2
13.4	Control Operations	13-3
13.5	Move Operations	13-6
13.6	I/O Operations	13-8

Appendixes

Appendix	Title	Page
A	Keycap Cross-Reference	A-1
B	COBOL Compiler Error Messages	B-1
C	COBOL Run-Time Error Messages	C-1
D	COBOL Subroutine Library Package	D-1
E	COBOL Compiler Listing Format	E-1

Index

Illustrations

Figure	Title	Page
1-1	Program Source Module — MANUAL.PG.SRC.FIG0101	1-3
1-2	Compiler Listing — MANUAL.PG.LST.FIG0102	1-4
2-1	Directory and File Structure	2-7
2-2	Sequential File Description and Creation	2-11
2-3	Sequential Files: Physical Record Size < Sector Size < ADU Size	2-12
2-4	Sequential Files: Physical Record Size = Sector Size < ADU Size	2-12
2-5	Sequential Files: Sector Size < Physical Record Size < ADU Size	2-13
2-6	Sequential Files: Sector Size < Physical Record Size = ADU Size	2-13
2-7	Sequential Files: Physical Record Size > ADU Size ≥ Sector Size	2-14
2-8	Relative Record File Description and Creation	2-16
2-9	Relative Record Files: Physical Record Size < Sector Size < ADU Size	2-17
2-10	Relative Record Files: Physical Record Size = Sector Size < ADU Size	2-18
2-11	Relative Record Files: Sector Size < Physical Record < ADU Size	2-18
2-12	Relative Record Files: Sector Size < Physical Record Size = ADU Size	2-19
2-13	Relative Record Files: Physical Record Size > ADU Size ≥ Sector Size	2-19
2-14	KIF Description, CFKEY Creation, and MKF Listing	2-22
3-1	Organization of Files in Directory	3-2
3-2	Sample COBOL Program Source Module — VOLUME.SOURCE.EXAMPLE2	3-4
4-1	Sample COBOL Compiler Listing	4-4
5-1	Determining Link Edit Requirements for COBOL Programs	5-3
5-2	Memory Mapping	5-4
5-3	Contents of a Program File	5-5
5-4	Multiple Tasks Sharing Same P1 and P2	5-7
5-5	Multiple Tasks Sharing Same P1 but Different P2s	5-8
5-6	Multiple Tasks on Separate Program Files	5-9
5-7	Comparison of Memory Requirements	5-10
5-8	COBOL Segmentation Within Overlay Phase Modules	5-11

Figure	Title	Page
5-9	Linking a Single Procedure Segment With a Single Task Segment	5-16
5-10	Linking a Single Procedure Segment With Multiple Task Segments	5-17
5-11	Linking Two Procedure Segments With a Single Task Segment	5-18
5-12	Linking Two Procedure Segments With Multiple Task Segments	5-20
5-13	Linking Two Procedure Segments With Multiple Task Segments (ALLOCATE)	5-21
5-14	An Overlay Structure With the Accompanying Link Control File	5-23
5-15	Sharing the Main Program Module With P2	5-24
5-16	Linking a P1 With Different P2s	5-25
5-17	Linking a Single Procedure Segment With a Single Task	5-27
5-18	Random Library Structure	5-28
6-1	SPECIAL-NAMES Paragraph Example	6-3
7-1	Compiler Output Listing	7-4
7-2	Interactive Debugging Example	7-17
7-3	COBOL Program Calling Assembly Language Modules	7-18
7-4	Assembly Language Module ADDRES	7-20
7-5	Assembly Language Module IOCALL	7-21
8-1	Example of COBOL Routine Calling Assembler Subroutine	8-5
8-2	Example of Assembler Subroutine Called by COBOL	8-9
9-1	COBOL Module Interfacing With TIFORM	9-2
9-2	TIFORM VDT Screen Description	9-6
9-3	COBOL Routine Calling Sort/Merge	9-8
9-4	COBOL Interfacing With DBMS-990	9-16
9-5	Data Definition Language (DDL) File	9-30
9-6	COBOL Module Linked to Query	9-32
10-1	Simple SCI Procedure	10-2
10-2	Tailored SCI Procedure	10-4
10-3	COBOL Procedure	10-7
10-4	COBOL Program Module Retrieving Additional SCI Parameters	10-8
11-1	Use of ACCEPT and DISPLAY Statements	11-5
11-2	Contents of VDT Screen at Program Completion	11-8
11-3	Graphics	11-9
11-4	Graphic Characters	11-11
12-1	Checking Error-Handling Capabilities Through DECLARATIVES	12-9

Tables

Table	Title	Page
1-1	Command Prompt Notation	1-10
3-1	Files Required for Program Development	3-1
4-1	COBOL Compiler Options	4-2
5-1	Valid Link Editor Commands With COBOL Object	5-12
7-1	Debug Commands	7-3
8-1	COBOL Subroutines Library	8-2
8-2	Format Codes for Calling Module	8-10
9-1	COBOL Entry Points to the Applications Interface Routines	9-2
11-1	Function Key Mapping	11-2
12-1	File Status Table	12-2
12-2	Operating System Errors and COBOL File Status Errors	12-3
12-3	COBOL I/O Operation Validity Table	12-7
12-4	Device Correspondence Table	12-8

1

Introduction

1.1 COBOL

The COBOL compiler conforms to the American National Standards Institute (ANSI) COBOL subset as defined in ANSI document X3.23-1974. The COBOL compiler incorporates extensions to this subset to provide added capabilities. The compiler package employs the following ANSI 74 standard COBOL modules at the level indicated:

Level 1 Features	Level 1 + Features*
Interprogram communications	Nucleus
Library	Table handling
Segmentation	Sequential I/O
	Relative I/O
	Indexed I/O

* Selected features from level 2

COBOL debug support and ACCEPT and DISPLAY statements are nonstandard and are designed for interactive use on video display terminals (VDTs).

1.2 A COBOL PROGRAM DEVELOPMENT OVERVIEW

The operating system provides developmental and operational support for program modules written in COBOL. The information presented in this section is an overview of the following:

- Building program source modules via the text editor
- Compiling program source modules to produce object program modules
- Linking program object modules to produce program images on a program file
- Executing program images on a program file
- Executing a program object module or a linked object module

Refer to the appropriate sections in this manual for specific details about developmental and operational support for program modules written in COBOL. The details of the language are discussed in the *COBOL Reference Manual*.

During the preparation of this manual, some assumptions have been made for the sake of a clear presentation. You are assumed to have a DX10 system with SCI, a terminal operating in VDT mode, a valid user ID, and a passcode.

The following definitions are provided to assist you when reading this manual:

Module — A set of computer program instructions treated as a unit by an assembler, compiler, link editor, or other similar processor.

Object File — A file (usually created by the compiler) containing one or more object program modules.

Program — A collection of object instructions that directs the activities of a computer; can consist of task segments, procedure segments, and overlays.

Task — A program that executes under control of the operating system.

Source File — A file (usually created by using the text editor) containing one or more program source modules (source code or statements).

Linked Object File — A file (created by the link editor) containing one or more program object modules that have been linked together to produce linked object modules.

Program File — A file (created by you or by the link editor) containing executable program components in memory image form.

Link Control File — A file (created by you) containing instructions for the link editor.

Subroutine — A sequenced set of statements that may be used in one or more programs and at one or more points in a program.

Logical Unit Number (LUNO) — A number that represents a file or device and is specified in an I/O operation.

Synonym — A text string that functions as an alternative for another string.

Normally, you write COBOL program source modules from a VDT under the control of the text editor. The text editor allows you to create or modify an existing program source module. This file is used as input to the COBOL compiler. A pathname is assigned to the source file at its creation. Pathnames are discussed in Section 2. Figure 1-1 shows a sample COBOL program source module.

When SCI commands are invoked during compilation or execution, a command heading and information concerning the software release level are displayed. The software release information appears as follows:

```
VERSION <L.R.V YYDDD>
```

where:

L is the software level.

R is the software release of level L.

V is the software version of release R (operating system).

YY is the year the software was released.

DDD is the day of the year when the software was released.

```
IDENTIFICATION DIVISION.
PROGRAM-ID. LRV.
ENVIRONMENT DIVISION.
CONFIGURATION SECTION.
SOURCE-COMPUTER. TI-990.
OBJECT-COMPUTER. TI-990.
INPUT-OUTPUT SECTION.
FILE-CONTROL.
    SELECT LISTFILE ASSIGN TO RANDOM "LST".
DATA DIVISION.
FILE SECTION.
FD   LISTFILE LABEL RECORDS STANDARD.
01   LISTING.
     02 CC          PIC X(3).
     02 DNCBL       PIC X(15).
     02 L-R-V       PIC X(7).
     02 YY-DDD      PIC X(7).
     02 COMPILED    PIC X(9).
     02 MM-DD-YY    PIC X(9).
     02 HH-MM-SS    PIC X(9).
     02 FILLER      PIC X(10).
     02 PAG         PIC X(4).
     02 FILLER      PIC X(7).
WORKING-STORAGE SECTION.
01   ACTION  PIC X.
01   EOF PIC X VALUE " ".
PROCEDURE DIVISION.
MAIN-PROG.
     OPEN I-O LISTFILE.
     PERFORM READ-WRITE UNTIL EOF > " ".
     CLOSE LISTFILE.
     STOP RUN.
READ-WRITE.
     READ LISTFILE AT END MOVE 1 TO EOF.
     IF DNCBL = "DNCBL"
        IF COMPILED = "COMPILED:"
           IF PAG = "PAGE"
              MOVE "L.R.V" TO L-R-V
              MOVE "YY.DDD" TO YY-DDD
              MOVE "MM/DD/YY" TO MM-DD-YY
              MOVE "HH:MM:SS" TO HH-MM-SS
              REWRITE LISTING.
```

Figure 1-1. Program Source Module — MANUAL.PG.SRC.FIG0101

To compile a COBOL program source module, enter one of the Execute COBOL Compiler (XCC or XCCF) commands. The command prompts for the XCCF command (with sample responses included) are as follows:

```
EXECUTE COBOL COMPILER FOREGROUND <VERSION: L.R.V. YYDDD>
    SOURCE ACCESS NAME: MANUAL.PG.SRC.FIG0102
    OBJECT ACCESS NAME: MANUAL.PG.OBJ.FIG0102
   LISTING ACCESS NAME: MANUAL.PG.LST.FIG0102
               OPTIONS: M
           PRINT WIDTH: 80
             PAGE SIZE: 55
   PROGRAM SIZE(LINES): 1000
```

After responding to the prompts, press the Return key to activate the compiler. When the compilation completes, a completion message appears on the video display terminal (VDT) screen. If an error occurs, check the error message in the appropriate appendix, correct the error, and recompile the program source module. Section 4 has complete instructions for compiling COBOL source program modules.

Figure 1-2 shows an example of a compiler listing. Notice that the number of errors and warnings as a result of the compilation are included near the end of the listing.

```
 DXCBL          L.R.V YY.DDD COMPILED:MM/DD/YY HH:MM:SS  OPT=M       PAGE    1

SOURCE ACCESS NAME:    MANUAL.PG.SRC.FIG0102
OBJECT ACCESS NAME:    MANUAL.PG.OBJ.FIG0102
LISTING ACCESS NAME:   MANUAL.PG.LST.FIG0102
OPTIONS:               M
PRINT WIDTH:           80
PAGE SIZE:             55
PROGRAM SIZE (LINES):  1000
```

Figure 1-2. Compiler Listing — MANUAL.PG.LST.FIG0102 (Sheet 1 of 3)

```
 DXCBL        L.R.V. YY.DDD COMPILED:MM/DD/YY HH:MM:SS  OPT=M      PAGE    2
LINE  DEBUG  PG/LN  A....B.............................................
   1                IDENTIFICATION DIVISION.
   2                PROGRAM-ID. LRV.
   3                ENVIRONMENT DIVISION.
   4                CONFIGURATION SECTION.
   5                SOURCE-COMPUTER. TI-990.
   6                OBJECT-COMPUTER. TI-990.
   7                INPUT-OUTPUT SECTION.
   8                FILE-CONTROL.
   9                    SELECT LISTFILE ASSIGN TO RANDOM "LST".
  10                DATA DIVISION.
  11                FILE SECTION.
  12                FD   LISTFILE LABEL RECORDS STANDARD.
  13                01   LISTING.
  14                     02 CC          PIC X(3).
  15                     02 DNCBL       PIC X(15).
  16                     02 L-R-V       PIC X(7).
  17                     02 YY-DDD      PIC X(7).
  18                     02 COMPILED    PIC X(9).
  19                     02 MM-DD-YY    PIC X(9).
  20                     02 HH-MM-SS    PIC X(9).
  21                     02 FILLER      PIC X(10).
  22                     02 PAG         PIC X(4).
  23                     02 FILLER      PIC X(7).
  24                WORKING-STORAGE SECTION.
  25                01   ACTION  PIC X.
  26                01   EOF PIC X VALUE " ".
  27                PROCEDURE DIVISION.
  28   >0000        MAIN-PROG.
  29   >0000             OPEN I-O LISTFILE.
  30   >0006             PERFORM READ-WRITE UNTIL EOF > " ".
  31   >0010             CLOSE LISTFILE.
  32   >0016             STOP RUN.
  33   >0018        READ-WRITE.
  34   >0018             READ LISTFILE AT END MOVE 1 TO EOF.
  35   >0022             IF DNCBL = "DNCBL"
  36                        IF COMPILED = "COMPILED:"

  37                            IF PAG = "PAGE"
  38                               MOVE "L.R.V" TO L-R-V
  39                               MOVE "YY.DDD" TO YY-DDD
  40                               MOVE "MM/DD/YY" TO MM-DD-YY
  41                               MOVE "HH:MM:SS" TO HH-MM-SS
  42                               REWRITE LISTING.
  43        ZZZZZZ END PROGRAM.                              *** END OF FILE
```

Figure 1-2. Compiler Listing — MANUAL.PG.LST.FIG0102 (Sheet 2 of 3)

```
DXCBL          L.R.V. YY.DDD COMPILED:MM/DD/YY HH:MM:SS  OPT=M      PAGE    3
ADDRESS  SIZE  DEBUG  ORDER TYPE                    NAME

            0                FILE                   LISTFILE
 >0026     80   GRP      0   GROUP                   LISTING
 >0026      3   ANS      0   ALPHANUMERIC              CC
 >0029     15   ANS      0   ALPHANUMERIC              DNCBL
 >0038      7   ANS      0   ALPHANUMERIC              L-R-V
 >003F      7   ANS      0   ALPHANUMERIC              YY-DDD
 >0046      9   ANS      0   ALPHANUMERIC              COMPILED
 >004F      9   ANS      0   ALPHANUMERIC              MM-DD-YY
 >0058      9   ANS      0   ALPHANUMERIC              HH-MM-SS
 >006B      4   ANS      0   ALPHANUMERIC              PAG

 >007A      1   ANS      0   ALPHANUMERIC            ACTION

 >007C      1   ANS      0   ALPHANUMERIC            EOF


READ ONLY BYTE SIZE =         >012A

READ/WRITE BYTE SIZE =        >00CE

OVERLAY SEGMENT BYTE SIZE =   >0000

TOTAL BYTE SIZE =             >01F8

    0 ERRORS

    0 WARNINGS
```

Figure 1-2. Compiler Listing — MANUAL.PG.LST.FIG0102 (Sheet 3 of 3)

After compilation, the compiled object module is either executed, linked to create a linked object module, or linked to create a program image on a program file. Refer to Section 6 for details and restrictions regarding execution of a compiled object module. Section 5 contains details and restrictions for linking.

Before MANUAL.PG.OBJ.FIG0101 is executed, external file assignments must be resolved if synonyms are specified in the source module. In Figure 1-1, the synonym LST must be assigned to the pathname of the compiler listing file. To assign the synonym LST, enter the Assign Synonym (AS) SCI command. The command prompts are as follows (with sample responses included):

```
ASSIGN SYNONYM VALUE
          SYNONYM: LST
            VALUE: MANUAL.PG.LST.FIG0101
```

To execute COBOL object modules, use the Execute COBOL Program (XCP or XCPF) SCI command. When the XCPF command is activated, enter the COBOL object file access name or linked object file access name defined when the COBOL program module was compiled or linked. The SCI commands associated with execution of a COBOL program are described in detail in Section 6. The command prompts are as follows (with sample responses included):

```
EXECUTE COBOL PROGRAM FOREGROUND <VERSION: L.R.V. YYDDD>
     OBJECT ACCESS NAME: MANUAL.PG.OBJ.FIG0101
             DEBUG MODE: NO
    MESSAGE ACCESS NAME:
               SWITCHES: 00000000
          FUNCTION KEYS: NO
```

To create a linked object module, the Link Editor utility and a link control file are required. If a link control file is not available, you must create one. An example link control file is as follows:

```
TASK LRV
INCLUDE MANUAL.PG.OBJ.FIG0101
END
```

You also need a link control file to link an object module for producing a program image using the Link Editor utility. An example link control file is as follows:

```
FORMAT IMAGE,REPLACE
PROCEDURE RCOBOL
DUMMY
INCLUDE .S$SYSLIB.RCBPRC
TASK LRV
INCLUDE .S$SYSLIB.RCBTSK
INCLUDE .S$SYSLIB.RCBMPD
INCLUDE MANUAL.PG.OBJ.FIG0101
END
```

In this link control file, named MANUAL.PG.CONTROL.EXAMPLE1, the IMAGE in the FORMAT statement ensures that the object file output from the link editor is written directly to a program file in memory image form. The word REPLACE ensures that any task segment in the program file with the name LRV is deleted before this task segment is written to the program file. The DUMMY command prevents the shared procedure segment (RCOBOL) from being replaced in the program file.

To initiate the link editor, enter the Execute Link Editor (XLE) SCI command. Respond to the prompts to link and install the LRV task on a program file named MANUAL.PG.PROGRAM. The command prompts are as follows (with sample responses included):

```
EXECUTE LINK EDITOR
       CONTROL ACCESS NAME: MANUAL.PG.CONTROL.EXAMPLE1
 LINKED OUTPUT ACCESS NAME: MANUAL.PG.PROGRAM
       LISTING ACCESS NAME: MANUAL.PG.LINKLIST.EXAMPLE1
       PRINT WIDTH (CHARS): 80
               PAGE LENGTH: 59
```

To execute linked object modules, use the Execute COBOL Program (XCP or XCPF) commands. You can execute the object module as a program image on a program file by using the Execute COBOL Task (XCT or XCTF) commands.

Now, to execute the installed program image, use the XCT or XCTF commands. Once the call has been issued, the COBOL task executes under control of the run-time interpreter; the interpreter is included as part of the task at link edit time.

To execute the task LRV on program file MANUAL.PG.PROGRAM, enter the XCTF command and respond to the command prompts. Section 6 describes the SCI commands associated with execution of a COBOL task. The command prompts are as follows (with sample responses included):

```
EXECUTE COBOL TASK FOREGROUND <VERSION: L.R.V. YYDDD>
        PROGRAM FILE LUNO: >7
          TASK ID OR NAME: LRV
               DEBUG MODE: NO
      MESSAGE ACCESS NAME:
                 SWITCHES: 00000000
           FUNCTION KEYS: NO
```

After responding to the prompts, the program executes. If an error occurs: 1) check the error message in the appropriate appendix; 2) correct the error; and 3) compile, link edit, and execute the task again.

COBOL debug mode is available only with the XCPF and XCTF commands. The debug mode provides for controlled execution of a program or task. When running in debug mode, a program or task can be halted and resumed. The debug mode allows you to specify address stops, single COBOL statement execution, or data item dumps. Also, it is possible to exit from debug mode or quit execution of a task. For further information about debugging, refer to Section 7.

1.3 SCI COMMAND PROMPT FORMAT AND NOTATION

When SCI command prompts are described in this manual, a standard format and notation is used. The notation is described in the following paragraphs.

1.3.1 Command Name

The characters of a command represent the full command name. For example, the characters of the Show Date and Time command are SDT. To enter a command, type the characters of the command and signal when finished by pressing the Return key.

When you enter SDT and press the Return key

```
[] SDT  <RETURN>
```

the system responds as follows:

```
13:48:30 WEDNESDAY, MAY 14, 1980.
```

Since the Show Date and Time command includes no command prompts, the command executes without further user interaction.

1.3.2 Command Prompts Returned

Upon entry of a command, the system displays the full name of the command and any associated command prompts. Command prompts provide you with information and request parameters to complete execution of the command. In the Show File example that follows, the cursor appears after the "FILE PATHNAME:" prompt. The system waits for you to enter a file pathname. (A pathname is a character string that indicates a path to a resource such as a file, channel, or device.)

1.3.3 Type of Response Expected

For each command prompt, a response of a given type is expected. In the remainder of this manual, the expected response type is given after each command prompt. In the Show File example that follows, the expected response type is a pathname. To enter a response, proceed as follows:

1. Type the desired response. The response must be of the type expected. To show the contents of a file named .MYFILE, type .MYFILE in response to the FILE PATHNAME: prompt of the Show File (SF) SCI command.

2. Press the Return key to signal that the entry is complete.

The following example illustrates the description of the SF command:

```
[ ] SF
SHOW FILE
      FILE PATHNAME: pathname@
```

Following the response to the first prompt, the cursor is positioned after the next prompt and waits for your response. After entry of the response to the last prompting message, the command executes. You can press the Command key prior to entering the last prompt to prevent execution of a command.

To help you respond to the prompts, the system sometimes displays an initial value after a prompt or has a default value available for a response. The following paragraphs describe initial values and default values.

1.3.3.1 Initial Values. An *initial value* is a value that the system automatically displays as a response to some prompting messages. Users can accept an initial value by pressing the Return key. They can erase the initial value by pressing the Erase Field or Skip key. Finally, they can reject the initial value by entering a different value.

The initial values for some prompts are fixed; therefore, the same initial value always appears for that prompt. In other cases, the system saves a value entered with a command and displays it as an initial value for a later entry of the same command or for the entry of a related command. Some variable initial values are also saved from one terminal session to another.

1.3.3.2 Default Values. A *default value* is a value that the system automatically supplies as the response to a prompt when you do not enter a value. The system often provides default values to speed up the entry of responses to prompts. This is especially true for optional user responses. To enter the default value for a prompt (where a default value exists), press the Return key without entering any other data. Such an entry is called a *null entry*.

Notation symbols (Table 1-1) enclose some prompt responses in the command descriptions to help explain how the responses are entered.

Table 1-1. Command Prompt Notation

Notation	Meaning
Uppercase	Enter the response as listed.
Lowercase	Enter a response of this type.
No marks	The response is required.
[ ]	The response is optional.
{ }	The response must be exactly one of the enclosed items or must be a type of one of the enclosed items. (Choices separated by a slash.)
item. . .item	More than one item of this type may be entered in reponse to the prompt. Items should be separated by commas.
@	Synonyms or logical names are allowed (as responses).
( )	The item enclosed in parentheses represents the initial value. If (*) is shown, the value may be supplied from a synonym set by a previously used command procedure. If a list is supplied in a form other than interactive (batch mode or a procedure calling a command procedure), the list must be enclosed in parentheses.

2

Operating System Concepts

2.1 INTRODUCTION

This section provides an overview and describes some important system capabilities. For more information, refer to the operating system manuals listed on the frontispiece of this manual.

2.2 TASK STRUCTURE

A task is a specific activation of a program. DX10 is a multitasking operating system designed to share concurrently the memory, machine execution time, and peripheral resources of the system among several tasks. While one task is active (executing), others are suspended awaiting reactivation.

At each terminal, it is possible to have one foreground task and one background task concurrently active.

2.2.1 Interactive Tasks

All interactive tasks operate in either foreground, background, or batch. A foreground task can accept data from the terminal as it is executing. In background mode, SCI does not expect interaction with terminals. You can start a task (for example, updating a database) in background mode and perform other activities (such as data collection) in foreground mode while the background task is active. When complete, the background task returns a message to the terminal, indicating completion.

Commands entered from interactive terminals are entered in foreground mode. The operating system responds by displaying the appropriate command prompts. Enter the required information; the task now begins execution. While the task executes in foreground, SCI is suspended to avoid interference. User interaction now occurs directly with the foreground task. The *DX10 Operations Guide* describes the commands that initiate tasks in all modes.

2.2.2 Batch Streams

Batch streams use SCI in background mode to process batch commands. In batch mode, SCI accepts commands from any sequentially oriented device but not from a terminal. When you enter commands in a batch command stream, include all parameters required for the operation. Also, be sure that the commands included are suitable for execution in background mode. Commands that initiate operations requiring user interaction (for example, text editing and debugging commands) are not permitted.

2.3 USING SCI

The following paragraphs discuss the use of SCI. Section 10 gives information for designing your own command procedures. The *DX10 Applications Programming Guide* contains complete descriptions of SCI commands, plus procedures for creating new commands and menus.

2.3.1 SCI Description

SCI is the interface between you and the operating system, system utilities, the software development programs, and application programs. Application programs can interface with you through user-defined SCI commands and menus.

You can use SCI to activate programs and to pass parameters to the programs during execution. SCI also allows you to build and maintain tables of variables, called *synonyms*, and their values. SCI allows application programs to access these variables for use in the programs.

To execute an application program via SCI, you can use predefined execution commands such as Execute Task (XT), Execute FORTRAN Task (XFT), Execute Pascal Task (XPT), and Execute COBOL Task (XCT), or you can write your own SCI command to initiate a program. You can add user-defined commands to the system library, or you can group them in a separate command library. The .USE primitive allows you to specify which command library SCI should use.

You can enter SCI commands from interactive terminals or in batch command streams. In response to commands entered interactively, SCI displays command prompts associated with the command.

When all required prompts have been properly answered, SCI interprets the responses and initiates the requested operation.

2.3.2 Entry of SCI Commands in VDT Mode

To enter an SCI command in VDT mode, type the characters (in uppercase letters) of the command and press the Return key. If you set the lowercase option with the .OPTION primitive, you can use either upper or lowercase characters. Upon entry of a command, SCI displays the full name of the command entered and all the field prompts associated with the command. Field prompts provide information and request parameters to complete command execution. For example, the following field prompt requests that you identify an output pathname:

```
OUTPUT PATHNAME:
```

2.3.3 Examples of Using SCI

The following paragraphs contain examples of specific uses of SCI commands.

2.3.3.1 The Show Background Status (SBS) Command. Use the SBS command to view the status of a program that is currently executing in background mode and that was initiated from your terminal. Since this command has no associated prompts, the command executes immediately after you enter SBS and press the Return key. A message indicating the state of the background activity appears, as follows:

```
[] SBS

SHOW BACKGROUND STATUS

TASK IS ACTIVE
```

2.3.3.2 The List Directory (LD) Command. Use the List Directory command to list the names of all files and subdirectories in a directory. The display for this command is as follows:

```
[] LD

LIST DIRECTORY
                PATHNAME: pathname@
    LISTING ACCESS NAME: [pathname]@
```

In response to the prompt PATHNAME, enter the pathname of the directory whose filenames and subdirectory names will be listed. The @ indicates that you can specify the pathname as a synonym.

In response to LISTING ACCESS NAME, enter the pathname of the device or file to which the listing should be written. The brackets ([ ]) indicate that the response is optional. The default value is the terminal at which the command is entered. A null response (pressing the Return key while the cursor is in a blank field) causes the default value to be accepted. In the following case, the directory SYS2.DP0080 is listed to the terminal from which the command was executed. Synonym D represents the directory pathname.

```
[] LD

LIST DIRECTORY
                PATHNAME: SYS2.DP0080
    LISTING ACCESS NAME:



DIRECTORY LISTING OF:  SYS2.DP0080
 MAX # OF ENTRIES: 101    # OF ENTRIES AVAILABLE: 78

DIRECTORY  ALIAS OF  ENTRIES    LAST UPDATE           CREATION
 ML           *        5     05/30/80  13:44:48  03/17/80 12:51:06
 TIP          *       11     05/07/80  12:02:20  02/11/80 16:44:21


FILE   ALIAS OF  RECORDS    LAST UPDATE     FMT  TYPE   BLK PROTECT
 BATCH    *         24   06/03/80  08:16:56  BS   N SEQ  YES
 COBOL    *       3550   05/30/80  14:06:46  NBS  N SEQ  YES
 DATA     *         17   05/07/80  15:31:57  BS   N SEQ  YES
 .      .          .       .          .      .   .   .    .    .
 .      .          .       .          .      .   .   .    .    .
16:21:50 TUESDAY, JUN 03, 1980.
```

2.3.4 Batch Use of SCI

To use SCI in a batch mode with a batch stream, use the Execute Batch (XB) command. The XB command starts a background task that is associated with your terminal.

The following paragraphs discuss the characteristics of batch SCI and the differences in format between batch commands and commands entered interactively.

2.3.4.1 Batch Stream Format. The first and last commands of a batch stream should be the BATCH and EBATCH commands, respectively. The BATCH command initiates the batch SCI environment. EBATCH indicates that the batch stream contains no more commands to be processed by SCI.

Upon normal completion of the batch stream executing in background mode, the following message appears:

```
BACKGROUND EXECUTION HAS COMPLETED:
```

2.3.4.2 Batch Command Format. When supplying SCI commands in batch stream format, include the following information for each command:

- The characters of the command
- All required prompts associated with the command
- The parameter values (responses) for the command prompts

The following examples demonstrate the Execute Link Editor (XLE) command in both interactive and batch form. (Refer to the *Link Editor Reference Manual* for a complete description of the XLE command.)

When you enter XLE interactively, the command prompts appear:

```
[ ] XLE

EXECUTE LINK EDITOR
            CONTROL ACCESS NAME: pathname@
      LINKED OUTPUT ACCESS NAME: [pathname]@
            LISTING ACCESS NAME: [pathname]@
                    PRINT WIDTH: integer          (80)
                    PAGE LENGTH: integer          (59)
```

To execute the command, respond to the CONTROL ACCESS NAME prompt by specifying the pathname of the file or device from which the control stream is to be read. Then, either specify values or accept the default values for the remaining prompts. If the control stream is contained in the file .M.CONTROL, the linked output is to be written to the file .M.OBJECT, the Link Editor listing is to be written to the file .M.LIST, and an 80-character line with 59 lines per page is acceptable, respond as follows:

```
[] XLE

EXECUTE LINK EDITOR
      CONTROL ACCESS NAME: .M.CONTROL
LINKED OUTPUT ACCESS NAME: .M.OBJECT
      LISTING ACCESS NAME: .M.LIST
      PRINT WIDTH (CHARS): 80
              PAGE LENGTH: 59
```

To execute this command in a batch stream, include the characters of the command, all required and any optional prompts that are specified, and the responses to those prompts. The following batch command is equivalent to the interactive version shown previously:

```
XLE   CONTROL=.M.CONTROL, LINKED OUTPUT=.M.OBJECT, LISTING=.M.LIST
```

Notice that you can accept the default values for the PRINT WIDTH and PAGE LENGTH prompts by omitting them from the batch command. Also, you can use abbreviated versions of the specified command prompts. The abbreviation must be sufficient to uniquely identify the prompt. Often, only the first character of a command prompt need be entered. For example, the following is equivalent to the previous example:

```
XLE   C=.M.CONTROL, LO=.M.OBJECT, LIST=.M.LIST
```

A batch stream consists of one command or a series of commands in this format, preceded by the BATCH command and followed by the EBATCH command. The file containing the batch command stream is the input file for the XB command.

2.3.4.3 Interactive Execution of Batch Streams. Use the XB command to execute batch streams as background activities. After you enter the XB command and the batch stream begins execution, you can continue to execute SCI commands in foreground mode. After the batch stream completes, the completion message appears the next time you press the CMD key. To monitor batch stream execution, enter the Show Background Status (SBS) command from time to time. Also, you can view the listing file for the batch stream during the run.

An example of the XB command is as follows:

```
[]XB

EXECUTE BATCH
         INPUT ACCESS NAME: pathname@
       LISTING ACCESS NAME: pathname@
```

The INPUT ACCESS NAME is the pathname of the device or file that contains the batch stream. The LISTING ACCESS NAME is the pathname of the device or file that is to receive the results of the batch stream execution. This device or file must not be used by any command in the batch stream.

2.3.4.4 Entering Programs From Sequential Devices. You can use any sequential file of program source code for input to the compilers or the assembler. If necessary, copy source code that has been key-punched on a card deck to a sequential disk file. Program source code, entered by the Text Editor or Copy Concatenate (CC) command, can be read from devices. An example of using the CC command to copy the source code from cards to a disk file is as follows:

```
[] CC

COPY/CONCATENATE
    INPUT ACCESS NAME(S): CR01
      OUTPUT ACCESS NAME: .USER.SOURCE
                 REPLACE?: NO
   MAXIMUM RECORD LENGTH:
```

2.4 DIRECTORY AND FILE STRUCTURE

File management allows you to build, organize, and access directories and files. A *file* consists of a named collection of data. The data in the file can be generated by you (for example, source code or documentation) or by the system (for example, object code or listing files). A *directory* is a relative record file that contains the information necessary to locate other files and describes the characteristics of those files. It does not contain user data.

2.4.1 Establishing Volume Names

Volume names are alphanumeric character strings of as many as eight characters that identify the disk on which a file is found. The first character of a volume name must be an alphabetic character. For example, VOL1 could be the volume name of a disk.

The Initialize Disk Surface (IDS) command prepares the disk surface for initialization by the Initialize New Volume (INV) command. The IDS command must be performed prior to the first INV command. It is not necessary to perform another IDS before any further initializations of the disk.

The INV command assigns volume names to disks. Once a volume is initialized by an INV command, all access to files on that volume must include the volume name in the pathname or access name, unless the volume is the system disk or unless a device is specified.

One disk drive on each system is designated to hold the system disk. The system disk contains all required operating system components, including the loader program, system program files, and temporary system files. The system disk is the default volume when no volume name is specified. For example, .PROOF designates a file named .PROOF on the system disk.

2.4.2 Establishing Directories

Each disk volume has a file directory named VCATALOG, to contain the volume table of contents. The files described in VCATALOG are data files or directory files (Figure 2-1).

Directory files contain the names of, and pointers to, other files; they do not contain user data. Typically, related files are contained in a directory. Directories can also contain subdirectories. Both directories and subdirectories are created by the Create Directory File (CFDIR) command. A subdirectory can be created under a directory only after the directory has been created. For example, subdirectory VOL1.SOURCE.PROGRAMA cannot be created unless directory VOL1.SOURCE already exists.

It is convenient to group related files into a single directory. For example, all source files for a program might be in a directory named VOL1.SOURCE.PROGRAMA; all listings generated from assembly or compilation of source modules for this program might be in a directory named VOL1.LISTING.PROGRAMA.

Do not assign file names that might be confused with system file names. Most system file or directory names begin with S$.

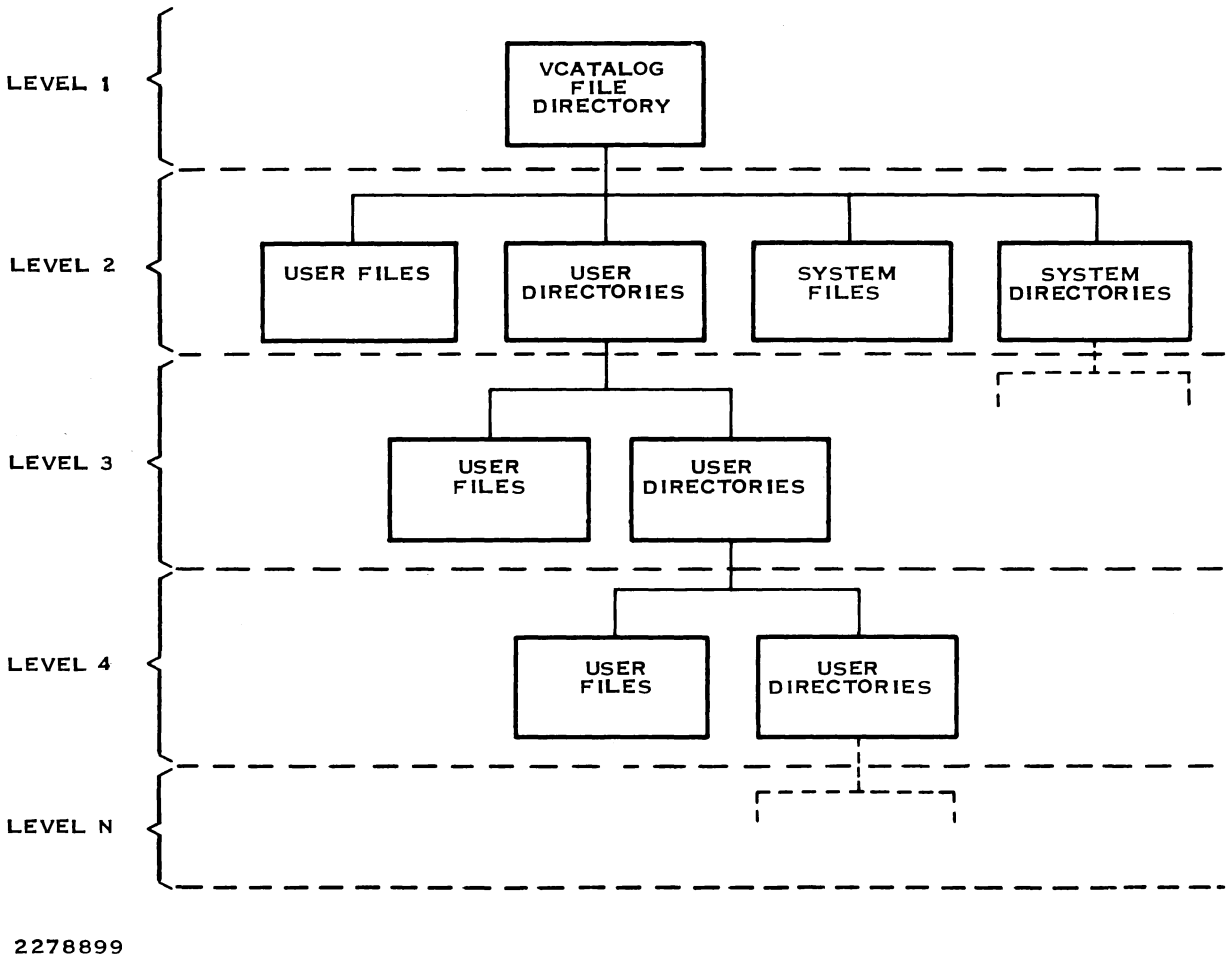


Figure 2-1. Directory and File Structure

2.4.3 Establishing Files

After initializing a disk volume and creating directories and subdirectories, you can create files that are accessible either under the volume or under a directory or subdirectory. The following commands are available to create files:

- Create Key Indexed File (CFKEY)
- Create Relative Record File (CFREL)
- Create Sequential File (CFSEQ)

- Create Program File (CFPRO)
- Create Image File (CFIMG)
- Create File (CF)

The CF command requires the subsequent selection of a file type.

2.5 PATHNAMES AND ACCESS NAMES

A file on a disk volume is referenced by its pathname. A *pathname* is a concatenation of the volume name, names of the directory levels leading to the file (excluding VCATALOG), and the file name itself. Each component of a pathname cannot exceed eight characters in length. A complete pathname must not exceed 48 characters, including the periods used to separate directories, subdirectories, and file names. The components of the pathname are separated by periods, as in the following examples:

```
VOL1.AGENCY.RECORDS

MYDIRECT.MYDIRCTA.MYFILE

VOLTWO.DEB

EMPLOY01.USRA.PAYROLL

EMPLOY01.USRB.CATALOGX.PAYROLL
```

An *access name* can be a device name, volume name, or file pathname. For device names, you must use certain default names (except for special devices). Example device names include ST02 for terminal number 2, LP01 for line printer number 1, and DS03 for disk number 3.

You can reference a volume on which a file resides through either the device name or the volume name. Omitting the volume name and beginning the pathname with a period indicates that the file is on the system disk. Samples of valid names for devices and files are as follows:

File Identifier	Meaning
CR01	Device name
DS02.MYCAT.MYFILE	Device name, directory name, file name
.MYCAT.MYFILE	System disk, directory name, file name
VOLID.MYCAT.MYFILE	Volume name, directory name, file name

2.6 SYNONYMS

Synonyms are abbreviations of one of more characters in length that are commonly used in place of long pathnames or portions of pathnames. These synonyms are always available to foreground tasks. Background tasks receive a copy of the foreground synonyms when the background task is initiated. At terminals requiring log-on, user-defined synonyms are associated with the user's ID and are available whenever that user logs on at any terminal. Use the Assign Synonym (AS) and Modify Synonym (MS) commands to define synonyms and to modify defined synonyms. When you enter a synonym in response to an SCI command prompt, the synonym is replaced by the actual text string.

When an SCI command is executed in foreground mode, you can use a synonym only as the first or only component of a pathname (device name or file name). For example, if A is a synonym for directory VOL1.SOURCE and B is a synonym for PROGRAMA in that directory, A.PROGRAMA is an acceptable file name. However, VOL1.SOURCE.B or A.B is not acceptable.

2.7 FILE TYPES

A file consists of a collection of data groupings called *logical records*. This division into logical records does not necessarily correspond to the physical division of data on disk or other media. Thus, in addition to logical records, files also have *physical records*.

A logical record is the amount of information transferred in one (not multiple) Read or Write I/O request. A physical record is the amount of data actually transferred by the operating system during an I/O operation to the file. The ratio of the physical record size to the logical record size is called the *blocking factor*. The logical record length (LRECL) in a file can be constant or can vary, depending on the file type.

Disk space is assigned in allocatable disk units (ADUs). An ADU is an integral number of disk sectors. The size of an ADU depends on disk capacity; larger disks have larger ADUs. An ADU is always smaller than a track. On some disks, ADUs are as small as one sector.

The following file types are supported: sequential, relative record, and key indexed.

2.7.1 Sequential Files

Sequential files are variable-record-length files whose records are always read, written, and accessed serially (that is, record 0 must be accessed first, record 1 must be accessed next, and so on). Some examples of using sequential files are as follows:

- As an input file for card images. If a logical record length of 80 is specified, the sequential file can be treated as a card reader by the program reading the file.
- As an output file. In this function, the file can resemble the line printer.
- As a location for listing files.

2.7.1.1 Sequential File Attributes. Sequential files have the following attributes:

- Sequential file logical records must be an even number of bytes in length.
- Sequential files can be created expandable. To extend the file, it must be opened in the open extend mode.
- Record-level locking is supported.
- Blank suppression and blank adjustment are allowed on sequential files that are used for input purposes. However, neither is performed on sequential files that are automatically created by COBOL. COBOL does not perform blank suppression or blank adjustment on sequential files so that they can be used in the I/O operation Rewrite. Rewrite verifies that the length of the record read has not changed before the rewrite is attempted.

If the logical record length defined in the program is larger than the actual record read from the file, the characters in the buffer beyond those of the actual record are undefined. For example, if the defined record length is 80 and the file contains variable-length records with the specific record read having a length of 50, the buffer area described in the file record-description-entry contains the 50-character record plus 30 characters undefined. COBOL does not automatically initialize its buffer area prior to a read operation. When reading variable-length records, the program should initialize the buffer area prior to each read operation.

Files assigned to the device name PRINT are created as sequential files with carriage control characters appended. With the appended characters, the logical record length is six characters larger than that specified in the program. The six characters are split, with from one to four characters preceding the record, and from one to four characters following the record, with a maximum of six characters per record.

2.7.1.2 Creating Sequential Files. Consider the following rules when creating sequential files:

- Logical record length must be less than or equal to the physical record length.
- Logical records can span sector boundaries.
- Logical records can span physical records; thus, partial records are created in both physical records.
- Logical records can span ADU boundaries.
- Physical records must begin on sector boundaries.
- Physical records beginning in the middle of an ADU cannot span the ADU boundary.

Figure 2-2 shows both a file description for a sequential file in a COBOL program and the creation of a sequential file using the Create Sequential File (CFSEQ) SCI command.

```
   .
   .
   .

SELECT SEQ-EMPLOYEE
     ASSIGN TO RANDOM, "EMPL"
     ORGANIZATION SEQUENTIAL
     ACCESS SEQUENTIAL
     FILE STATUS SEQ-STATUS.

   .
   .
   .

FD   SEQ-EMPLOYEE LABEL RECORDS STANDARD.
01   SEQ-RECORD.
     02   SOCIAL-SECURITY                    PIC X(9).
     02   EMPLOYEE-NAME.
          03   EMPLOYEE-FIRST-INITIAL        PIC X.
          03   EMPLOYEE-SECOND-INITIAL       PIC X.
          03   EMPLOYEE-LAST-NAME            PIC X(20).
     02   REST-OF-DATA                       PIC X(113).

   .
   .
   .

-----------------------------------------------------


CREATE SEQUENTIAL FILE
                PATHNAME: EMPL
    LOGICAL RECORD LENGTH: 144
   PHYSICAL RECORD LENGTH:
       INITIAL ALLOCATION:
     SECONDARY ALLOCATION:
             EXPANDABLE ?: YES
         BLANK SUPPRESS ?: NO
           FORCED WRITE ?: NO
```

Figure 2-2. Sequential File Description and Creation

To minimize wasted disk space, the physical record size should be an integral multiple or factor both of the ADU size and of the sector size.

The following figures illustrate the relationships between the logical record, physical record, sector, and ADU sizes. In some instances, disk space is wasted; in others, no space is wasted, depending on the physical record size chosen. Each figure defines the relationship between logical record, physical record, sector, and ADU sizes. The boxed information represents a linear description of the logical records on a file. Below the logical record are the physical record, sector, and ADU divisions of the data.

Figure 2-3 indicates the relationship between the physical record, sector, and ADU sizes when the physical record size is less than the sector size and the sector size is less than the ADU size. In this case, logical records are spanning physical records. Space is wasted within each sector because the physical record must begin on the next sector boundary.

Figure 2-4 indicates the relationship between physical record, sector, and ADU sizes when the physical record size is equal to the sector size and the sector size is less than the ADU size. In this case, logical records are spanning physical, sector, and ADU boundaries.

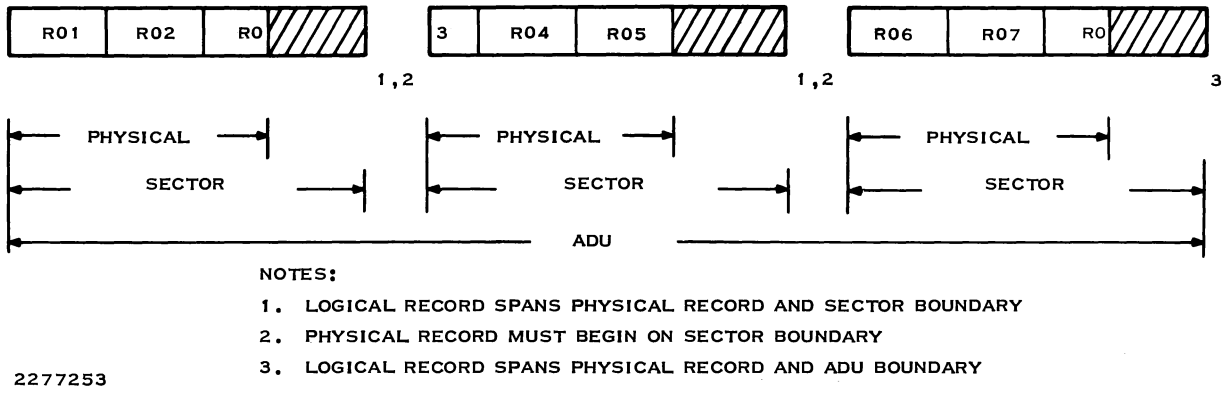


Figure 2-3. Sequential Files: Physical Record Size < Sector Size < ADU Size

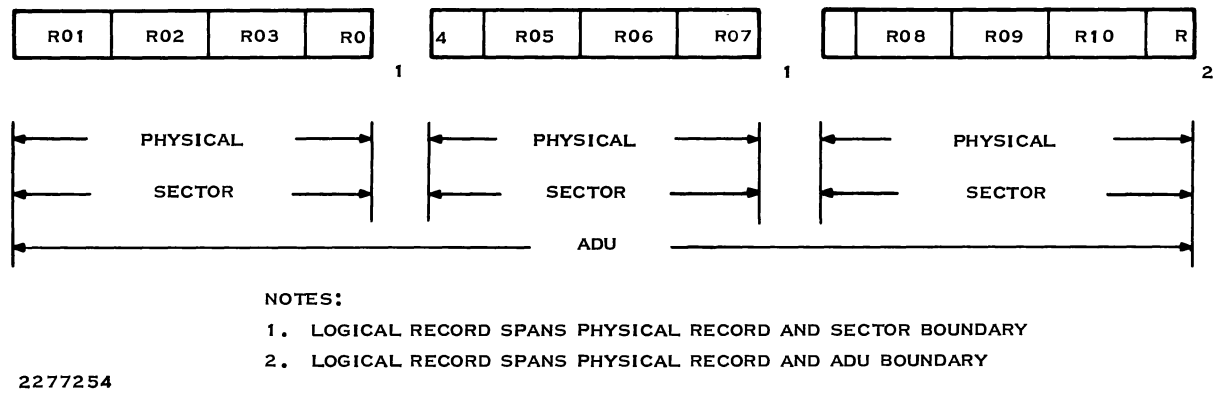


Figure 2-4. Sequential Files: Physical Record Size = Sector Size < ADU Size

Figure 2-5 indicates the relationship between physical record, sector, and ADU sizes when the sector size is less than the physical record size and the physical record size is less than the ADU size. In this case, the physical record is two times the sector size. One sector for every ADU is wasted because there is not enough space in the ADU to hold another physical record.

Figure 2-6 indicates the relationship between physical record, sector, and ADU sizes when the physical record size is equal to the ADU size. When the physical record size is not specified at file creation, the default value used is the defined default of the directory on which the file is created. Logical records span physical records, sectors, and ADU boundaries.

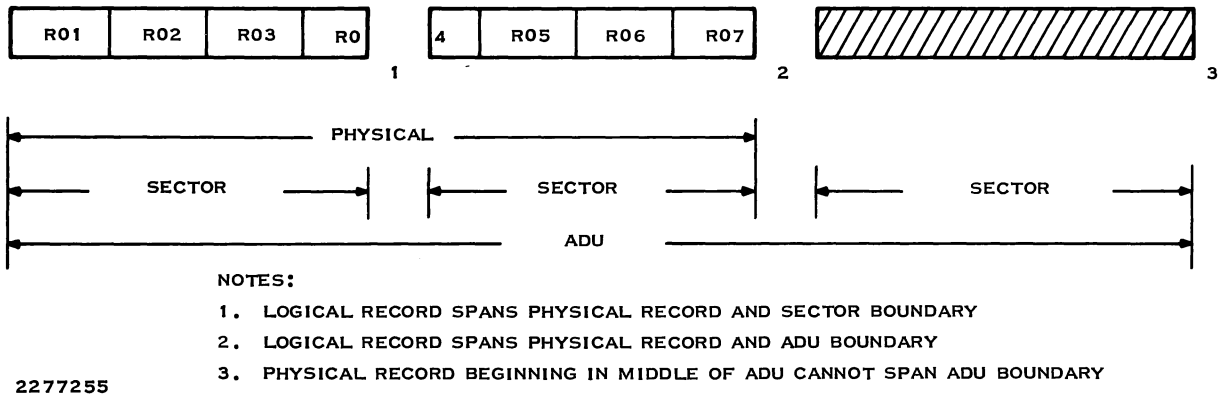


Figure 2-5. Sequential Files: Sector Size < Physical Record Size < ADU Size

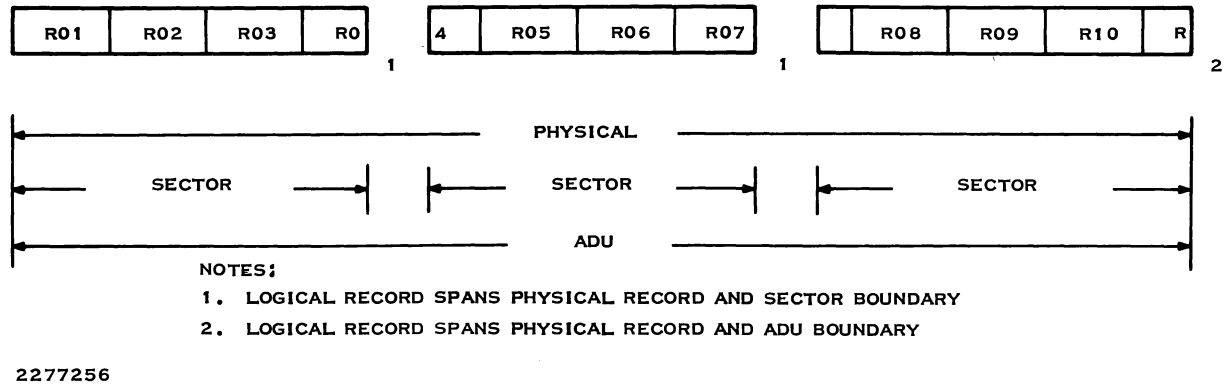


Figure 2-6. Sequential Files: Sector Size < Physical Record Size = ADU Size

Figure 2-7 indicates the relationship between physical record, sector, and ADU sizes when the physical record size is greater than the ADU size and the ADU size is greater than or equal to the sector size. In this case, space is wasted on the disk because the remaining space of the ADU is too small to contain another physical record. Therefore, the next physical record must begin on the next ADU boundary. Note that the logical record spans to the next physical record, which begins on the next ADU.

2.7.2 Relative Record Files

Relative record files are also called random-access files. Unlike sequential files, relative record files can be accessed in any order. Each record has a unique record number, which you specify to access that individual record. The operating system increments the caller's record number after each read or write so that sequential access is permitted. One end-of-file (EOF) record is maintained wherever it was last specified by a program. The range of record numbers is from zero to one less than the number of records in the file. The maximum number of records in a relative record file is 2 to the 24th power. The records are fixed in length, and the length must be specified during file creation.

Relative record files are useful when each record in the file is already associated with a unique value ranging from 0 to n; for example, in an inventory file, the item number can be specified as the record number. Consequently, information about item number 1 can be obtained by accessing record number 1.

2.7.2.1 Relative Record Attributes. Relative record files have the following attributes:

- Relative record files can be accessed sequentially in ascending order.
- Relative record files can be accessed randomly in any order.
- Records of odd or zero length are not allowed.
- All records are fixed in length, and the length must be specified during file creation.

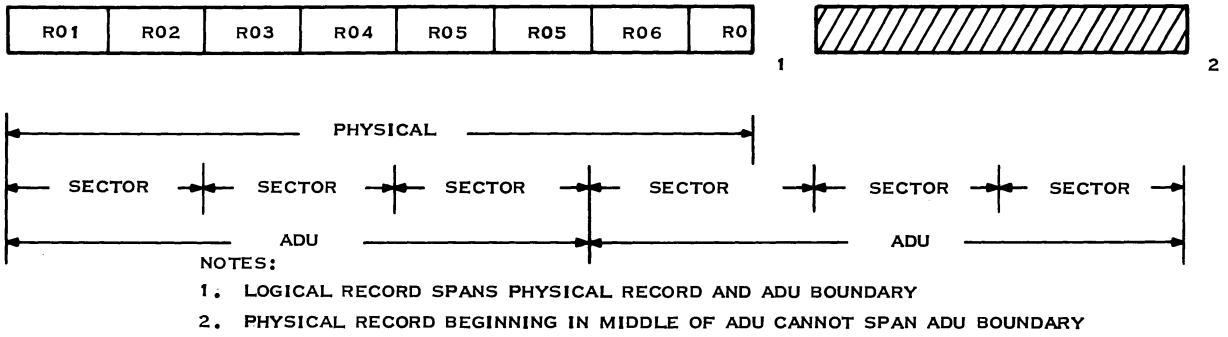


Figure 2-7. Sequential Files: Physical Record Size > ADU Size ≥ Sector Size

- Variable length records are not allowed.
- Blank suppression and blank adjustment are not allowed.
- Deleted records in a relative record file are flagged by COBOL with a hexadecimal FF (>FF) in the first character of the record. These flagged records are ignored by COBOL during sequential read operations. Therefore, data records should not contain binary data in the first character position. The concept of deleted records is not recognized by the file management of the operating system.
- Record-level locking is supported.
- Relative record files can be expanded by adding a record or records whose record number is greater than the highest record number currently in the file. During this operation, any record between the current last record and the new last record is added to the file. Each of the deleted records has >FF in the first character position, flagging the records as being deleted. All records between the lowest and highest record numbers on the file must be present as either data records or deleted records (place holders) in order to locate any given record on a random I/O request.

Each record is uniquely identified by its position. The operating system increments the caller's record number after each read or write to allow sequential access. One EOF record is maintained wherever it was last specified by a program. To access record number n, record number n is requested. The range of record numbers is from 0 to one less than the number of records in the file. The maximum number of records in a relative record file is 2 to the 24th power.

2.7.2.2 Creating Relative Record Files. Consider the following rules when creating relative record files:

- Logical record length must be less than or equal to the physical record length.
- Logical records can span sector boundaries.
- Logical records cannot span physical records.
- Physical records must begin on sector boundaries.
- Physical records beginning in the middle of an ADU cannot span ADU boundaries.
- Physical records should be an integral multiple of sectors.

Figure 2-8 shows both a file description for a relative record file in a COBOL program and the creation of a relative record file using the Create Relative Record File (CFREL) SCI command.

```
    .
    .
    .

SELECT REL-EMPLOYEE
     ASSIGN TO RANDOM, "EMPL"
     ORGANIZATION RELATIVE
     ACCESS RELATIVE
     RELATIVE KEY REL-KEY
     FILE STATUS REL-STATUS.

    .
    .
    .

FD   REL-EMPLOYEE LABEL RECORDS STANDARD.
01   REL-RECORD.
     02   SOCIAL-SECURITY                    PIC X(9).
     02   EMPLOYEE-NAME.
          03   EMPLOYEE-FIRST-INITIAL        PIC X.
          03   EMPLOYEE-SECOND-INITIAL       PIC X.
          03   EMPLOYEE-LAST-NAME            PIC X(20).
     02   REST-OF-DATA                       PIC X(113).

    .
    .
    .


WORKING-STORAGE SECTION.
01   REL-KEY                                 PIC 9(6).

    .
    .
    .


-------------------------------------------------------

CREATE RELATIVE RECORD FILE
                 PATHNAME: EMPL
    LOGICAL RECORD LENGTH: 144
   PHYSICAL RECORD LENGTH:
       INITIAL ALLOCATION:
     SECONDARY ALLOCATION:
             EXPANDABLE ?: YES
           FORCED WRITE ?: NO
```

Figure 2-8. Relative Record File Description and Creation

To minimize wasted disk space, choose the physical record length (PRECL) such that it is one of the following: either it is the largest integral multiple of the logical record size that is less than or equal to the ADU size, or it is an integral multiple of the ADU size.

The following figures illustrate the relationships between the logical record, physical record, sector, and ADU sizes. In all cases, some disk space is wasted; the amount depends on the physical record size chosen. Each figure defines the relationship between logical record, physical record, sector, and ADU sizes. The boxed information represents a linear description of the logical records on a file. Below the logical records are physical record, sector, and ADU divisions of the data.

Figure 2-9 indicates the relationship between physical record, sector, and ADU sizes when the physical record size is less than the sector size, and the sector size is less than the ADU size. Space is wasted within each sector because the physical record must begin on the next sector boundary.

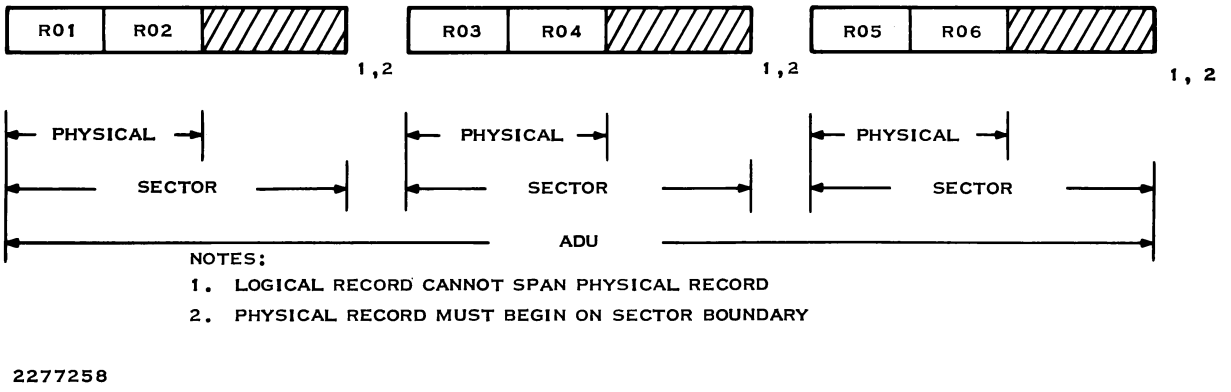


Figure 2-9. Relative Record Files: Physical Record Size < Sector Size < ADU Size

Figure 2-10 indicates the relationship between physical record, sector, and ADU sizes when the physical record size is equal to the sector size and the sector size is less than the ADU size. In this case, if a logical record does not fit into the remaining space of a physical record, the space is unused and the logical record begins in the next physical record.

Figure 2-11 indicates the relationship between physical record, sector, and ADU sizes when the sector size is less than the physical record size and the physical record size is less than the ADU size. In this case, the physical record is two times the sector size. More than one sector for every ADU is wasted because there is not enough space in the ADU to hold another physical record.

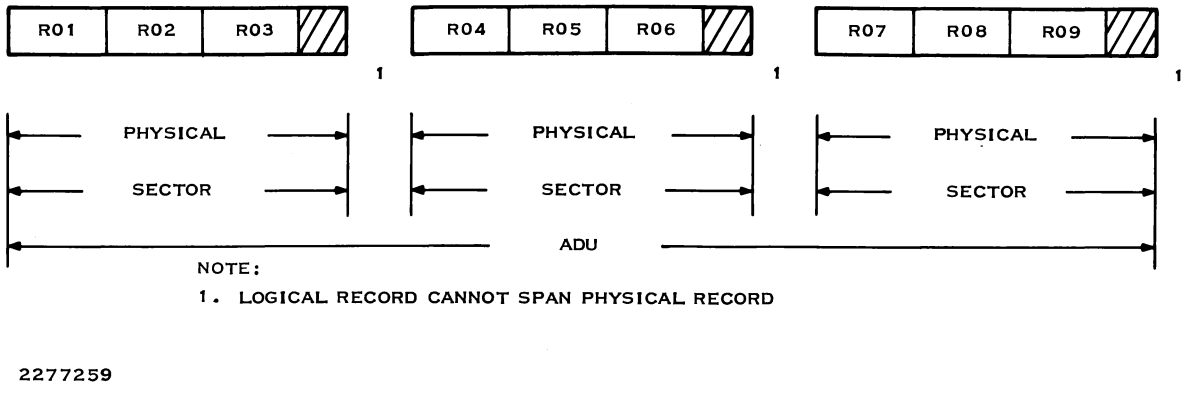


Figure 2-10. Relative Record Files: Physical Record Size = Sector Size < ADU Size

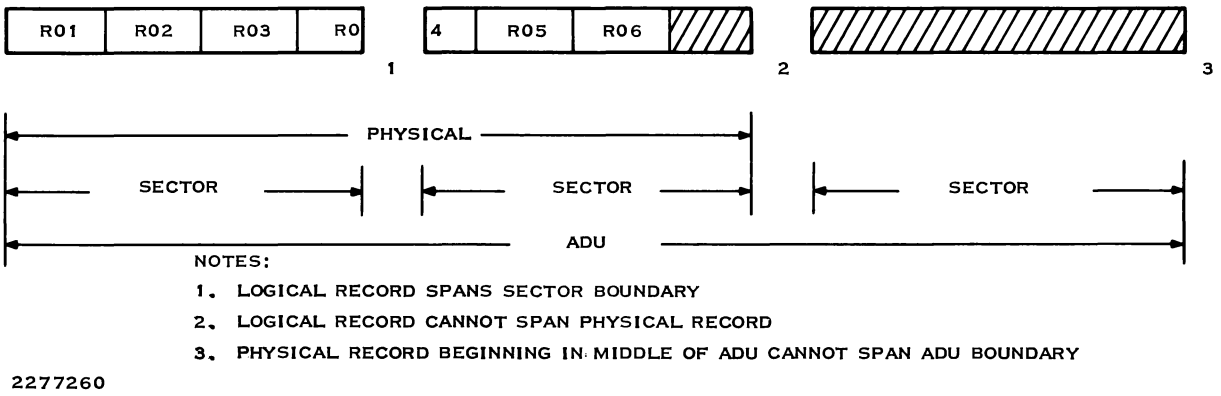


Figure 2-11. Relative Record Files: Sector Size < Physical Record Size < ADU Size

Figure 2-12 indicates the relationship between physical record, sector, and ADU sizes when the sector size is less than the physical record size and the physical record size is equal to the ADU size. When the physical record size is not specified at file creation, the default value used is the defined default of the directory on which the file is created. Logical records can span only sector and ADU boundaries. If a logical record does not fit into the space of a physical record, the space is unused and the logical record begins on the next physical record.

Figure 2-13 indicates the relationship between physical record, sector, and ADU sizes when the physical record size is greater than the ADU size and the ADU size is greater than or equal to the sector size. In this case, space is wasted on the disk because the remaining space of the ADU is too small to contain another physical record. Therefore, the next physical record must begin on the next ADU boundary. Note that the logical record must span to the next physical record, which begins on the next ADU.

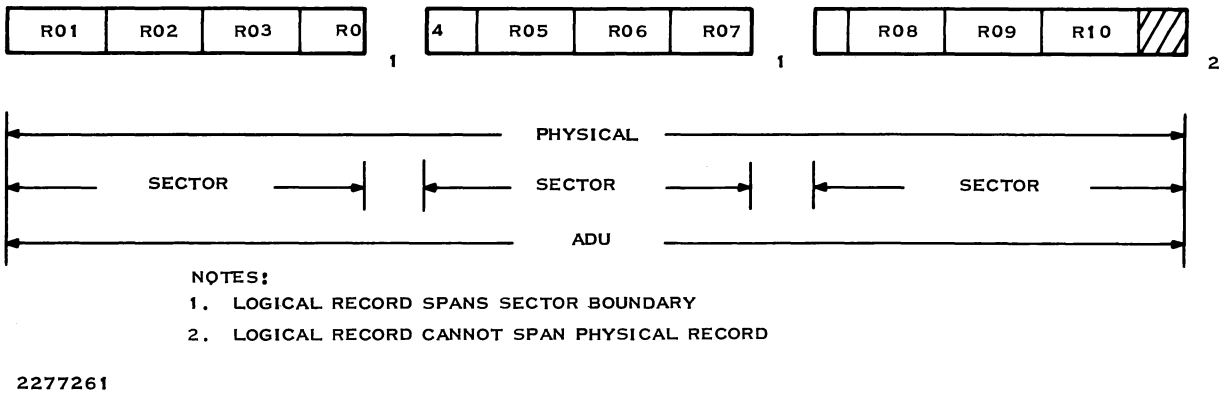


Figure 2-12. Relative Record Files: Sector Size < Physical Record Size = ADU Size

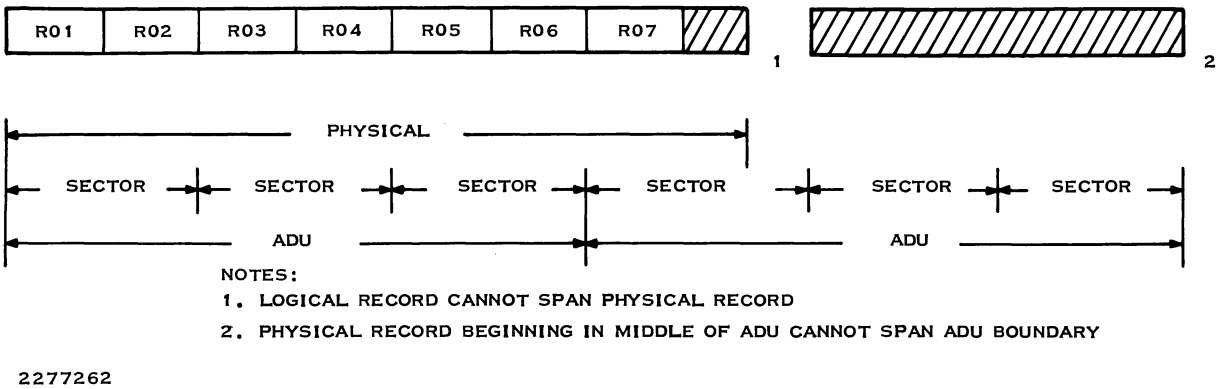


Figure 2-13. Relative Record Files: Physical Record Size > ADU Size ≥ Sector Size

2.7.2.3 Special Types of Relative Record Files. There are three special types of relative record files available: directory, program, and image files. These files provide special interface mechanisms that are used primarily for memory images, memory swapping, and diagnostic dumps.

- Directory files — Contain names of and pointers to other files
- Program files — Contain program images and an internal directory of the images
- Image files — Special-purpose files used primarily by the operating system for memory images, memory swapping, and diagnostic dumps

None of these special types of relative record files can be accessed through COBOL programs.

2.7.3 Key Indexed Files (KIF)

A KIF allows random access to its records via a key. The key is a character string of up to 100 characters, located in a fixed position within each file record. From 1 to 14 individual keys can be specified. For example, the records in an employee file can be accessed by keys that indicate the employee's ID, name, and social security number.

Keys can overlap one another, with certain restrictions, within the record. Although the keys can be structured anywhere within a record, they must appear in the same relative position in all records in the file. One key must be specified as the primary key; the other keys are secondary keys. The primary key must be present in all records, but secondary keys are optional.

In addition to supporting random access, KIFs include the following characteristics:

- Records can be accessed sequentially in the sort order of any key.
- At file creation, any key can be designated as allowing duplicates, which means that two or more records in the file can have the same value for this key.
- At file creation, any key except the primary key can be designed as being modifiable. This means that when a record is being rewritten, the key value may change. Also, a secondary key value that is missing in the record can be added later on a rewrite.
- Alternate keys cannot overlap the primary key.
- Alternate keys cannot overlap the first character position of any other alternate key.
- Records can be of variable length.
- A start is allowed on the first portion of a key.

- Records are automatically blank-suppressed.
- Record-level locking is supported.
- The file is expanded dynamically allocating space when needed.
- File integrity is maintained through pre-image logging of modified blocks. Before a record is modified on disk, it is copied to a backup area in the file overhead area. Consequently, system failures cause the loss of only the last I/O operation.
- Records of odd or zero length are not allowed.

The physical record length must be greater than or equal to 22 plus the logical record length. For maximum efficiency, the physical record length should equal the ADU size of the disk on which the file is to reside or a multiple of the ADU size.

To ensure that a sufficient buffer is allocated at execution time, the COBOL program source module must define the maximum record size in the file description. If the file was created using the average blank-suppressed logical record length, an invalid record length error is returned on an Open request. Under these conditions, the USE procedures of the DECLARATIVES can be specified to intercept and ignore the invalid record length error returned on the OPEN request. (Refer to Section 12 for more details on intercepting and ignoring I/O errors.) The *TI COBOL Reference Manual* contains a detailed explanation of the USE and OPEN statements and the keyword DECLARATIVES.

If a KIF is created with the Create Key Indexed File (CFKEY) command and the KIF is to be used in COBOL programs, the keys must be defined in the following order:

- Primary key
- Alternate key with the lowest displacement
- Alternate key with the next lowest displacement
- Alternate key with the highest displacement

The number of keys must exactly match the number of keys declared in the source program. The key lengths, flags (modifiable and duplicate attributes), and offsets must also match those declared in the program. The primary key cannot have duplicates or be modifiable. Alternate keys must all be modifiable and can have duplicates only when the duplicates are declared as such in the program. Alternate keys can overlap in any character position except the first, thereby preventing any two keys from having the same displacement. Alternate keys must never overlap the primary key in any character position. If any of the preceding conditions fails to match at open time, an invalid open error occurs (status code 94).

Figure 2-14 shows both the file description for a KIF in a COBOL program and the creation of the KIF using the Create Key Indexed File (CFKEY) SCI command. After the KIF is created, use a Map Key Indexed File (MKF) SCI command to view the key attributes.

```
    .
    .
    .
SELECT EMPLOYEE-MASTER
     ASSIGN TO RANDOM, "EMPL"
     ORGANIZATION INDEXED
     ACCESS RANDOM
     RECORD KEY SOCIAL-SECURITY
     ALTERNATE RECORD KEY EMPLOYEE-NAME
     ALTERNATE RECORD KEY EMPLOYEE-LAST-NAME
          WITH DUPLICATES
     FILE STATUS EMPLOYEE-STATUS.

    .
    .
    .

FD   EMPLOYEE-MASTER LABEL RECORDS STANDARD.
01   EMPLOYEE-RECORD.
     02   SOCIAL-SECURITY                    PIC X(9).
     02   EMPLOYEE-NAME.
          03   EMPLOYEE-FIRST-INITIAL        PIC X.
          03   EMPLOYEE-SECOND-INITIAL       PIC X.
          03   EMPLOYEE-LAST-NAME            PIC X(20).
     02   REST-OF-DATA                       PIC X(113).

    .
    .
    .

-----------------------------------------------------

CREATE KEY INDEXED FILE
                PATHNAME: EMPL
    LOGICAL RECORD LENGTH: 144
   PHYSICAL RECORD LENGTH:
       INITIAL ALLOCATION:
     SECONDARY ALLOCATION:
             MAXIMUM SIZE: 1000

KEY DESCRIPTION FOR KEY NUMBER 1
       START POSITION: 1
           KEY LENGTH: 9
          DUPLICATES?: NO
          MODIFIABLE?: NO
       ANY MORE KEYS?: YES
```

Figure 2-14. KIF Description, CFKEY Creation, and MKF Listing (Sheet 1 of 2)

```
KEY DESCRIPTION FOR KEY NUMBER 2
       START POSITION: 10
           KEY LENGTH: 22
          DUPLICATES?: NO
          MODIFIABLE?: YES
       ANY MORE KEYS?: YES

KEY DESCRIPTION FOR KEY NUMBER 3
       START POSITION: 12
           KEY LENGTH: 20
          DUPLICATES?: YES
          MODIFIABLE?: YES
       ANY MORE KEYS?: NO

-------------------------------------------------

        FILE MAP OF .MASTER
          TODAY IS 09:00:41 FRIDAY, SEPTEMBER 26, 1980

KEYS:

          START                                 DUPLICATES
  KEY     COLUMN      LENGTH      MODIFIABLE    ALLOWED
   1          1           9           N            N
   2         10          22           Y            N
   3         12          20           Y            Y
```

Figure 2-14. KIF Description, CFKEY Creation, and MKF Listing (Sheet 2 of 2)

3

Building a COBOL Source Program Module

3.1 GENERAL

The initial phase of COBOL program development involves building the program source module. This process requires preparing the necessary directories and files and entering the program source code (presumably via the text editor).

3.2 DIRECTORY AND FILE PREPARATION

Table 3-1 lists and describes the files that are typically used when developing and executing COBOL programs. (Optional procedures may require additional files.)

Table 3-1. Files Required for Program Development

File	Description
Source file	Contains program source module code, which is created by using the text editor and input to the COBOL compiler.
Object file	Contains program object module code, which is output from the COBOL compiler and input to the link editor or the Execute COBOL Program (XCP) command. (Refer to Section 6 for details about the XCP command.)
Compiler listing file	Contains the program source module listing with any errors detected by the COBOL compiler. The COBOL compiler produces this listing.
Link control file	Contains instructions for the link editor, such as which object modules, run-time libraries, user libraries, and external routines are to be linked.
Link editor listing	Contains the link map, which is produced by the link editor.
Program file	The user's program file; contains programs in image format.

3.3 ALTERNATE DIRECTORY STRUCTURES

File organization varies according to the requirements of a specific installation. Several methods of organization are possible, including the following:

- Organization according to related programs
- Organization according to file type

3.3.1 Organization by Programs

When files are organized by programs, all necessary files for a given program are located in a single directory; the directory name is associated with the program name. In the following example, all files for PROGRAMA are in directory PROGA, and all files for PROGRAMB are in directory PROGB:

```
VOLUME.PROGA.SRCFILE        VOLUME.PROGB.SRCFILE
VOLUME.PROGA.OBJFILE        VOLUME.PROGB.OBJFILE
VOLUME.PROGA.LSTFILE        VOLUME.PROGB.LSTFILE
VOLUME.PROGA.CTRFILE        VOLUME.PROGB.CTRFILE
VOLUME.PROGA.LINKMAP        VOLUME.PROGB.LINKMAP
VOLUME.PROGA.PRGFILE        VOLUME.PROGB.PRGFILE
```

3.3.2 Organization by File Type

In the diagram in Figure 3-1, files are arranged under a single directory (PROJECT). Subdirectories are created for source, object, listing, link control, and link map files. This type of file organization allows for a network of programs where the same module may be linked into different programs.

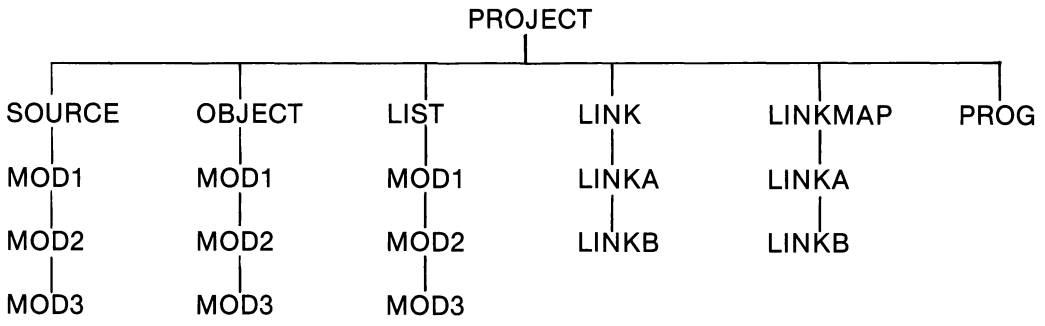


Figure 3-1. Organization of Files in Directory

3.4 CREATING DIRECTORIES AND FILES

To create a directory or subdirectory, enter the Create Directory File (CFDIR) SCI command. The following display appears:

```
CREATE DIRECTORY FILE
                    PATHNAME: pathname@ (*)
                 MAX ENTRIES: integer
 DEFAULT PHYSICAL RECORD SIZE: [integer]
```

Assume that the pathname has a volume name of VOLUME and a directory name of SOURCE. SOURCE will contain all source files for programs. Respond to the prompt PATHNAME by entering VOLUME.SOURCE. Respond to the prompt MAX ENTRIES by entering the maximum number of entries (files and subdirectories) that the directory may contain.

Files that are output from utilities (such as the text editor or the compiler) need not be created prior to executing the utility; the utility automatically creates the files if they do not already exist. However, pathnames must be specified before termination of the utility. Pathnames must be unique unless the information in a file is being replaced. Directories are not automatically created. The compiler automatically creates the compiler listing file and the object file if they do not already exist. Since the link control file is a utility input file, it must be created (usually via the text editor) prior to executing the link editor.

3.5 BUILDING THE PROGRAM MODULE VIA THE TEXT EDITOR

COBOL source program modules are generated on a VDT using SCI. Editing on the VDT occurs on a page basis; each page can have any consecutive 24 lines displayed on the screen. You can edit any record displayed on the screen by positioning the cursor anywhere within the line that contains the record. You can insert records between any lines, and you can insert or delete them in any order. Also, you can insert, delete, or modify characters within a line. Use the Show Line (SL) SCI command and the F2 (Roll Up Function), F1 (Roll Down Function), Previous Line and Next Line control keys to access specific lines, records, or characters.

To enter a source program module via the text editor (assuming a directory has been created previously), enter the Initiate Text Editor (XE) SCI command, and press the Return key. The following display appears:

```
EXECUTE TEXT EDITOR
          FILE ACCESS NAME:
           EXCLUSIVE EDIT?: YES
               LINE LENGTH: 80
```

Press the Return key to indicate that no file exists. The Text Editor clears the VDT screen and displays the following in the first four columns of row 1 with the cursor in column 1, row 1:

```
*EOF
```

This display indicates that the end-of-file (EOF) record is the only record in the file. To begin entering data, press the Return key. Notice that a blank line appears before the *EOF notation. Press the Command key and enter the Modify Tabs (MT) SCI command to adjust the tabs for coding. Set the tabs at 1, 8, 12, 24, and 73 (standard tabs for a COBOL coding sheet), and press the Return key. Now, begin entering the source code shown in Figure 3-2. Each time you enter a new line and press the Return key, a new blank line appears beneath the previously entered line of information.

```
 IDENTIFICATION DIVISION.
 PROGRAM-ID. FUNCTION.
*     THIS PROGRAM WAS DESIGNED AS A FUNCTIONAL
*     DEMONSTRATION TEST FOR CHECKING FUNCTION KEY
*     ACCESSIBILITY.
*     FUNCTION KEYS MUST HAVE BEEN ACTIVATED VIA THE
*     SCI EXECUTION COMMAND.
 ENVIRONMENT DIVISION.
 CONFIGURATION SECTION.
 SOURCE-COMPUTER. TI-990.
 OBJECT-COMPUTER. TI-990.
 DATA DIVISION.
 WORKING-STORAGE SECTION.
 01   ACTION PIC XX.
 01   FUNC   PIC 99.
 01   X      PIC S99 COMP-1.
 01   XX     PIC S99 COMP-1.
 01   HEADS.
      02 FILLER PIC X(21) VALUE
          "01 - F1".
      02 FILLER PIC X(21) VALUE
          "02 - F2".
      02 FILLER PIC X(21) VALUE
          "03 - F3".
      02 FILLER PIC X(21) VALUE
          "04 - F4".
      02 FILLER PIC X(21) VALUE
          "05 - F5".
      02 FILLER PIC X(21) VALUE
          "06 - F6".
      02 FILLER PIC X(21) VALUE
          "07 - F7".
      02 FILLER PIC X(21) VALUE
          "08 - F8".
      02 FILLER PIC X(21) VALUE
          "09 - F9".
      02 FILLER PIC X(21) VALUE
          "10 - F10".
      02 FILLER PIC X(21) VALUE
          "11 - F11".
      02 FILLER PIC X(21) VALUE
          "12 - F12".
      02 FILLER PIC X(21) VALUE
          "13 - F13".
      02 FILLER PIC X(21) VALUE
          "14 - F14".
      02 FILLER PIC X(21) VALUE
          "40 - Command".
```

Figure 3-2. Sample COBOL Program Source Module — VOLUME.SOURCE.EXAMPLE2 (Sheet 1 of 2)

```
           02 FILLER PIC X(21) VALUE
               "49 - Print".
           02 FILLER PIC X(21) VALUE
               "52 - Previous Line".
           02 FILLER PIC X(21) VALUE
               "53 - Next Line".
           02 FILLER PIC X(21) VALUE
               "54 - Home".
           02 FILLER PIC X(21) VALUE
               "55 - Next Field".
           02 FILLER PIC X(21) VALUE
               "56 - Previous Field".
           02 FILLER PIC X(21) VALUE
               "57 - Skip".
           02 FILLER PIC X(21) VALUE
               "58 - Forward Tab".
           02 FILLER PIC X(21) VALUE
               "59 - Initialize Input".
           02 FILLER PIC X(21) VALUE
               "61 - Erase Input".
           02 FILLER PIC X(21) VALUE
               "64 - Enter".
       01  HEADINGS REDEFINES HEADS.
           02 HEAD PIC X(21) OCCURS 26.
       PROCEDURE DIVISION.
       MAIN-PROG.
       RD-INPUT.
           DISPLAY "COBOL FUNCTION KEYS TEST"
              LINE 1 POSITION 20 ERASE.
           PERFORM DSP-13 THRU E-13 VARYING X FROM 1
              BY 1 UNTIL X > 13.
           PERFORM DSP-26 THRU E-26 VARYING X FROM 14
              BY 1 UNTIL X > 26.
           DISPLAY "DEPRESS DESIRED KEY" LINE 20 POSITION 20.
           PERFORM GET-FUNC UNTIL ACTION = "X".
           STOP RUN.
       GET-FUNC.
           ACCEPT ACTION LINE 20 POSITION 40
              ON EXCEPTION FUNC
                 DISPLAY FUNC LINE 20 POSITION 40.
           DISPLAY "HIT 'CR' TO CONTINUE, 'X' TO STOP"
              LINE 22 POSITION 20.
           ACCEPT ACTION LINE 22 POSITION 54.
           DISPLAY "  " LINE 20 POSITION 40.
       DSP-13.
           COMPUTE XX = X + 1.
           DISPLAY HEAD (X) LINE XX POSITION 20.
       E-13. EXIT.
       DSP-26.
           COMPUTE XX = X - 12.
           DISPLAY HEAD (X) LINE XX POSITION 45.
       E-26. EXIT.
       END PROGRAM.
```

Figure 3-2. Sample COBOL Program Source Module — VOLUME.SOURCE.EXAMPLE2 (Sheet 2 of 2)

After entering the program source module, check for errors. To return to the first page of the source code, press the Command key and enter the SL command. The following display appears:

```
SHOW LINE
         LINE: 1
```

Press the Return key to accept the initial value of 1. To review the source code, use the F1 and F2 keys. Each time the F1 key is pressed, the display scrolls forward; each time the F2 key is pressed, the display scrolls backward. To change the number of lines that are scrolled, enter the Modify Roll (MR) SCI command, and press the Return key. The following display appears:

```
MODIFY ROLL
  NUMBER OF LINES TO ROLL: 23
```

A different value may appear as the initial value of this command prompt. In any case, the response to this prompt should be 23. This allows the last line of the display to appear as the first line on the next display when the F1 key is pressed or the first line of the display to appear as the last line on the next display when the F2 key is pressed. Now, press the Return key.

Certain keys can be helpful when verifying the source code. Each of these keys may be used in conjunction with the Repeat key. The keys and their functions are as follows:

- Previous Line — Moves the cursor up one line from the current line. If the cursor is on the top line, the screen scrolls backward one line.
- Next Line — Moves the cursor down one line from the current line. If the cursor is on the bottom line, the screen scrolls forward one line.
- Previous Character — Moves the cursor to the left one character from the current position of the cursor.
- Next Character — Moves the cursor to the right one character from the current position of the cursor.

If no errors are found, press the Command key again and enter the Quit Edit (QE) SCI command. The following display appears:

```
QUIT EDIT
       ABORT?: NO
```

A YES response to the prompt ABORT? terminates the text editor without any modification to the input file; if no input file was specified in the XE command, no new file is created. Any modifications made or data entered are lost when the response to the ABORT? prompt is YES. Accept the initial value (NO) and press the Return key. The following display appears:

```
QUIT EDIT
   OUTPUT FILE ACCESS NAME: VOLUME.SOURCE.EXAMPLE2
                  REPLACE?: YES
      MOD LIST ACCESS NAME:
```

Enter a valid pathname such as VOLUME.SOURCE.EXAMPLE2 for the output file access name, and press the Return key. The response to the prompt REPLACE? determines whether the designated output file is to be replaced by the edited file. If the response is NO and the output file exists, the edited file does not replace the existing file. If the response is NO and no file exists by that name, a new file is created. If the response is YES, the edited file replaces the specified file; if no file exists by that name, a new file is created. Press the Return key in response to the prompt MOD LIST ACCESS NAME. The program is now entered and has a file name of VOLUME.SOURCE.EXAMPLE2.

When you are editing a source file, the functions of various keys can be helpful. For instance, the F4 key duplicates information on a previous line to a preset tab when the cursor is placed beneath the line to be copied. The F5 key acts as a tab key and clears the line to the preset tab positions, and the F6 key displays or suppresses line numbers. When line numbers are displayed, only 74 characters of each record are displayed. When line numbers are suppressed, a full 80 characters are displayed. Other keys of importance include the following:

- Initialize Input key — Inserts a blank line above the line containing the cursor
- Insert Character key — Inserts characters at the current cursor position and moves all characters that are to the right of the cursor one position to the right (truncates characters if line is full)
- Delete Character key — Deletes characters at the current cursor position and moves all characters that are to the right of the cursor one position to the left
- Home key — Positions the cursor in row 1, column 1 of the display
- Erase Field key — Replaces all characters in a line with blanks
- Erase Input key — Deletes the line on which the cursor is positioned and rolls up all lines beneath it

Certain SCI commands can also be helpful when editing a file. These commands include the following:

- FS (Find String) — Locates a predefined string in the source file for a specified number of occurrences
- DL (Delete Lines) — Deletes certain lines specified by the user
- ML (Move Lines) — Moves specified lines in a file and inserts them after a specified line number
- CL (Copy Lines) — Duplicates the specified lines and inserts them after a specified line number
- IF (Insert File) — Inserts an existing file into the file that is being edited, after a specified line number

4

Compilation

4.1 GENERAL

Compilation is the process of translating a COBOL program source module into a series of instructions (interpretive object code) comprehensible to the computer. The interpretive object code is interpreted by the COBOL run-time interpreter at execution time. (Refer to Section 5 for a description of the COBOL run-time interpreter.)

4.2 COMPILER EXECUTION

To execute the COBOL compiler, enter the Execute COBOL Compiler in Background (XCC) command for background compiles or the Execute COBOL Compiler in Foreground (XCCF) command for foreground compiles. The XCC command allows the terminal to be used for foreground purposes during the background compilation.

4.2.1 Execute COBOL Compiler in Foreground (XCCF)

For the XCCF command, the following prompts appear with the indicated initial values:

```
EXECUTE COBOL COMPILER FOREGROUND <VERSION: L.R.V YYDDD>
      SOURCE ACCESS NAME: pathname@
      OBJECT ACCESS NAME: pathname@
     LISTING ACCESS NAME: pathname@
                 OPTIONS: [{D/I/M/O/X}]
             PRINT WIDTH: integer          (80)
               PAGE SIZE: integer          (55)
    PROGRAM SIZE (LINES): integer        (1000)
```

Press the Return key after each entry.

SOURCE ACCESS NAME — Enter the input device name, pathname, or synonym for the file that contains the source module to be compiled.

OBJECT ACCESS NAME — Enter the pathname or synonym of the output object file. The compiler places the generated object code in the object file. The pathname must refer to a mass storage file with relative record organization. If the file does not exist, the compiler automatically creates a relative record file for the object file. If the file exists but is not a relative record file, the compiler terminates and an error is generated. (Refer to Appendix C for a listing of the compiler error messages.) If DUMY is specified for the object access name, the output object file is not generated.

LISTING ACCESS NAME — Enter the listing device name, pathname or synonym. The name entered is the name of the device or sequential file to which the compiler outputs the requested listings. If a file is specified and does not exist, the compiler automatically creates a sequential file for the listing file. Enter ME to have the listing displayed on the screen as it is generated.

OPTIONS — To request options, enter (without intervening commas) one or more of the characters listed in Table 4-1.

Table 4-1. COBOL Compiler Options

Character	Option
D	Debug
I	Information Message
M	Data Maps
O	List Object
X	Cross-Reference Listing

Entering the M option causes a listing similar to Figure 4-1.

The order in which the options are listed is not important. However, invalid options generate warnings and then are ignored. Descriptions of the options are as follows:

- Debug Option (D) — Causes the compiler to compile source statements that have a D in character position seven, along with rest of the statements in the program source module. Otherwise, the source statements with D in position seven are treated as comments.

- Information Message Option (I) — Causes the compiler to list any informative messages. These messages are not errors or warnings. See Table B-3 in Appendix B for the list of informative messages.

- Data Maps Option (M) — Causes the data map to be listed as part of the compiler listing (listing access name). Otherwise, no data map is listed. Refer to Appendix E for a COBOL object listing example including data maps.

- List Object Option (O) — Causes the compiler to include the object code in the listing file, following the listing of the corresponding source statement. Refer to Appendix E for a COBOL object listing example including object code.

- Cross-Reference Listing Option (X) — Causes the compiler to produce a cross-reference listing following the source listing or data maps if requested. Data names, index names, condition names, file names, section names, and paragraph names (contained in the Procedure Division of the program) are listed in the cross-reference. The line numbers of all appearances of a name are printed to the right of the name. When a line number is enclosed in slashes (/nnnn/), the statement on that line defines the item. When a line number is enclosed in asterisks (*nnnn*), the statement on that line may alter the contents of the item. When a line number is enclosed in blanks (nnnn), the statement on that line references the item.

PRINT WIDTH — Enter the appropriate print width to specify the number of characters to be formatted on a line of the listing. The compiler truncates the compiler listing lines if the print width is less than the compiler-generated line length. The initial value print width is 80 positions.

PAGE SIZE — Enter the maximum number of print lines per page for the compiler listing file. The initial value page size is 55 lines per page.

PROGRAM SIZE (LINES) — Enter an estimate of the number of program source module lines contained in the program source module. This estimate determines the amount of initial memory used in the compilation. If more memory is requested, compilation is faster provided memory is available. The initial value program size is 1000 lines.

After the program module is compiled, if an error occurs, correct the error and recompile the source module before attempting to link edit or execute the code. When the compilation completes successfully, the following message appears:

```
COBOL COMPILER COMPLETED, 0 ERRORS, 0 WARNINGS
```

```
 DXCBL          L.R.V YY.DDD COMPILED:MM/DD/YY HH:MM:SS  OPT=M     PAGE    1

SOURCE ACCESS NAME:    MANUAL.PG.SRC.FIG0401
OBJECT ACCESS NAME:    DUMY
LISTING ACCESS NAME:   MANUAL.PG.LST.FIG0401
OPTIONS:               M
PRINT WIDTH:           80
PAGE SIZE:             55
PROGRAM SIZE (LINES):  1000


 DXCBL          L.R.V YY.DDD COMPILED:MM/DD/YY HH:MM:SS  OPT=M     PAGE    2
LINE  DEBUG PG/LN   A...B.................................................
   1                IDENTIFICATION DIVISION.
   2                PROGRAM-ID. FUNCTION.
   3               *     THIS PROGRAM WAS DESIGNED AS A FUNCTIONAL
   4               *     DEMONSTRATION TEST FOR CHECKING FUNCTION KEY
   5               *     ACCESIBILITY.
   6               *     FUNCTION KEYS MUST HAVE BEEN ACTIVATED VIA THE
   7               *     SCI EXECUTION COMMAND.
   8                ENVIRONMENT DIVISION.
   9                CONFIGURATION SECTION.
  10                SOURCE-COMPUTER. TI-990.
  11                OBJECT-COMPUTER. TI-990.
  12                DATA DIVISION.



 DXCBL          L.R.V YY.DDD COMPILED:MM/DD/YY HH:MM:SS  OPT=M     PAGE    3
LINE  DEBUG PG/LN   A...B.................................................
  13               /
  14                WORKING-STORAGE SECTION.
  15                01   ACTION PIC XX.
  16                01   FUNC   PIC 99.
  17                01   X      PIC S99 COMP-1.
  18                01   XX     PIC S99 COMP-1.
  19                01   HEADS.
  20                     02 FILLER PIC X(21) VALUE
  21                        "01 - F1".
  22                     02 FILLER PIC X(21) VALUE
  23                        "02 - F2".
  24                     02 FILLER PIC X(21) VALUE
  25                        "03 - F3".
  26                     02 FILLER PIC X(21) VALUE
  27                        "04 - F4".
  28                     02 FILLER PIC X(21) VALUE
  29                        "05 - F5".
  30                     02 FILLER PIC X(21) VALUE
  31                        "06 - F6".
  32                     02 FILLER PIC X(21) VALUE
  33                        "07 - F7".
```

Figure 4-1. Sample COBOL Compiler Listing (Sheet 1 of 4)

```
 34                     02 FILLER PIC X(21) VALUE
 35                        "08 - F8".
 36                     02 FILLER PIC X(21) VALUE
 37                        "09 - F9".
 38                     02 FILLER PIC X(21) VALUE
 39                        "10 - F10".
 40                     02 FILLER PIC X(21) VALUE
 41                        "11 - F11".
 42                     02 FILLER PIC X(21) VALUE
 43                        "12 - F12".
 44                     02 FILLER PIC X(21) VALUE
 45                        "13 - F13".
 46                     02 FILLER PIC X(21) VALUE
 47                        "14 - F14".
 48                     02 FILLER PIC X(21) VALUE
 49                        "40 - Command".
 50                     02 FILLER PIC X(21) VALUE
 51                        "49 - Print".
 52                     02 FILLER PIC X(21) VALUE
 53                        "52 - Previous Line".
 54                     02 FILLER PIC X(21) VALUE
 55                        "53 - Next Line".
 56                     02 FILLER PIC X(21) VALUE
 57                        "54 - Home".
 58                     02 FILLER PIC X(21) VALUE
 59                        "55 - Next Field".
 60                     02 FILLER PIC X(21) VALUE
 61                        "56 - Previous Field".
 62                     02 FILLER PIC X(21) VALUE
 63                        "57 - Skip".
 64                     02 FILLER PIC X(21) VALUE
 65                        "58 - Forward Tab".
 66                     02 FILLER PIC X(21) VALUE
 67                        "59 - Initialize Input".
 68                     02 FILLER PIC X(21) VALUE


 DXCBL          L.R.V YY.DDD COMPILED:MM/DD/YY HH:MM:SS   OPT=M      PAGE    4
LINE  DEBUG PG/LN   A...B.....................................................
 69                        "61 - Erase Input".
 70                     02 FILLER PIC X(21) VALUE
 71                        "64 - Enter".
 72                01   HEADINGS REDEFINES HEADS.
 73                     02 HEAD PIC X(21) OCCURS 26.
```

Figure 4-1. Sample COBOL Compiler Listing (Sheet 2 of 4)

```
 DXCBL          L.R.V YY.DDD COMPILED:MM/DD/YY HH:MM:SS   OPT=M      PAGE    5
LINE  DEBUG PG/LN   A...B.....................................................
  74               /
  75                PROCEDURE DIVISION.
  76  >0000         MAIN-PROG.
  77  >0002         RD-INPUT.
  78  >0002              DISPLAY "COBOL FUNCTION KEYS TEST"
  79                        LINE 1 POSITION 20 ERASE.
  80  >000C              PERFORM DSP-13 THRU E-13 VARYING X FROM 1
  81                        BY 1 UNTIL X > 13.
  82  >0020              PERFORM DSP-26 THRU E-26 VARYING X FROM 14
  83                        BY 1 UNTIL X > 26.
  84  >0034              DISPLAY "DEPRESS DESIRED KEY" LINE 20 POSITION 20.
  85  >003C              PERFORM GET-FUNC UNTIL ACTION = "X".
  86  >0046              STOP RUN.
  87  >0048         GET-FUNC.
  88  >0048              ACCEPT ACTION LINE 20 POSITION 40
  89                        ON EXCEPTION FUNC
  90                           DISPLAY FUNC LINE 20 POSITION 40.
  91  >005E              DISPLAY "HIT 'CR' TO CONTINUE, 'X' TO STOP"
  92                        LINE 22 POSITION 20.
  93  >0066              ACCEPT ACTION LINE 22 POSITION 54.
  94  >006E              DISPLAY "  " LINE 20 POSITION 40.
  95  >0078         DSP-13.
  96  >0078              COMPUTE XX = X + 1.
  97  >007E              DISPLAY HEAD (X) LINE XX POSITION 20.
  98  >008E         E-13. EXIT.
  99  >0090         DSP-26.
 100  >0090              COMPUTE XX = X - 12.
 101  >0096              DISPLAY HEAD (X) LINE XX POSITION 45.
 102  >00A6         E-26. EXIT.
 103         ZZZZZZ END PROGRAM.                              *** END OF FILE
```

Figure 4-1. Sample COBOL Compiler Listing (Sheet 3 of 4)

```
DXCBL          L.R.V YY.DDD COMPILED:MM/DD/YY HH:MM:SS   OPT=M      PAGE      6
ADDRESS    SIZE DEBUG ORDER TYPE                    NAME

 >002A        2  ANS     0   ALPHANUMERIC          ACTION

 >002C        2  NSU     0   NUMERIC UNSIGNED      FUNC

 >002E        2  NBS     0   BINARY SIGNED         X

 >0030        2  NBS     0   BINARY SIGNED         XX

 >0032      520  GRP     0   GROUP                 HEADS

 >0032      520  GRP     0   GROUP                 HEADINGS
 >0032       20  ANS     1   ALPHANUMERIC           HEAD


READ ONLY BYTE SIZE =       >01C8

READ/WRITE BYTE SIZE =      >0248

OVERLAY SEGMENT BYTE SIZE = >0000

TOTAL BYTE SIZE =           >0410

    0 ERRORS

    0 WARNINGS
```

Figure 4-1. Sample COBOL Compiler Listing (Sheet 4 of 4)

4.2.2 Execute COBOL Compiler in Background (XCC)

For the XCC command, the following prompts appear with the indicated initial values:

```
EXECUTE COBOL COMPILER <VERSION: L.R.V YYDDD>
            SOURCE ACCESS NAME: pathname@
            OBJECT ACCESS NAME: pathname@
           LISTING ACCESS NAME: pathname@
                       OPTIONS: [{D/I/M/O/X}]
                   PRINT WIDTH: integer    (80)
                     PAGE SIZE: integer    (55)
          PROGRAM SIZE (LINES): integer  (1000)
```

The parameters are the same as those for the XCCF command except that ME should not be used as the listing access name.

4.3 COMPILER OUTPUT

The compiler output consists of the object file and the listing file. The object file contains the object modules (interpretive code) generated by the computer. The reentrant code (instructions) is generated as a group named PSEG. The nonreentrant code (data) is generated as a group named DSEG. DSEGs are often referred to as $DATA. The object file may be executed by the run-time interpreter or linked to another object module. The listing file contains the listing of the program source code and lists any error messages detected by the compiler.

4.4 COMPILER COMPLETION CODES

The COBOL compiler returns a system completion code for the most severe diagnostic encountered in the compilation. The completion code is returned in the synonym $$CC. The values and meanings of these codes are as follows:

Value	Meaning
0000	No warnings or errors occurred
4000	Warnings occurred
8000	Errors occurred

The synonym $$CC should be checked in batch streams immediately after compiler execution. $$CC is used by other processors, and its integrity is not guaranteed after completion of the batch stream or execution of another command.

4.5 COMPILER ERROR MESSAGES

The compiler generates user and system error messages. User error messages are included in the compiler listing. Compilation of a program source module proceeds to the end of a program module regardless of the number of errors found.

Errors that prevent proper execution of the COBOL compiler are system errors. When one of these errors occurs, the system displays an error message and terminates the execution of the compiler. Refer to Appendix B for a listing of user and system error messages and their meanings.

4.6 COMPILER LIMITATIONS

Each of the following items is limited to 2047 entries:

- Level-88 condition names
- Nesting of IF statements
- Nesting of PERFORM statements

- Using parameters in CALL statements
- Unique index names
- Unique spellings (identifiers, paragraph/section/internally generated labels)
- Unique literal values
- Unique identifiers (data names)
- Unique paragraph/section/internally generated labels
- Unique references to data items

In practice, because of interactions between different statements and related temporary information during the compilation process, the actual limits may be somewhat less than 2047. However, the limits for all practical purposes should be higher than typical program modules require.

5

Link Edit

5.1 GENERAL

Link editing is the process of preparing object modules for execution. It can also combine two or more separately compiled object modules to form a single linked object module. This process is performed by one of the operating system utilities, the link editor. The process of link editing resolves external definitions and references between object modules.

Object modules do not always require linking before execution. They must be linked as a linked object module if subroutines are present. (Refer to the section entitled Creating Linked Object Modules.) Also, object modules must be linked to a program file when task and procedure segments or overlays are needed. (Refer to the section entitled Creating Program Images.)

The following features are supported with linked object modules:

- Callable subroutines
- COBOL program module segmentation
- Object file compression

The following features are supported on program files:

- Callable subroutines
- Reentrant user modules
- Shared procedure segments
- Overlay phases
- COBOL program module segmentation

The diagram in Figure 5-1 shows the link edit and execution options available with COBOL programs. In the logical flow labeled A, no linking is necessary to execute an object module. (Refer to Object Modules Execution in Section 6 for a description of how to execute object modules.) The logical flow labeled B indicates that object modules must be linked when they contain CALL statements. The linked object modules are then executed using the same SCI commands as used for object modules. The logical flow labeled C shows how to execute a program (task) installed in a program file. (Refer to Program Image Execution in Section 6 for a description of how to execute program images on program files.) The Execute COBOL Program (XCP) and Execute COBOL Program in Foreground (XCPF) commands shown in Figure 5-1 reflect the method of executing compiled object files and linked object files. The Execute COBOL Task (XCT) and Execute COBOL Task in Foreground (XCTF) commands show the method of executing linked program images on program files.

User programs that operate under control of the operating system can include a combination of data, procedures, and overlays as required. Programs are installed and stored on program files in memory image form. When a program is activated, the images of its program segments are loaded into available memory areas. The hardware mapping facility precludes the necessity of relocating program images. Thus, the operating system can swap an active program to various locations in memory several times during execution. This process assists in sharing memory and making CPU execution time available (time-slicing). The hardware mapping capability also allows three separately loaded program segments to be mapped into a single, logically contiguous program address space.

5.2 OBJECT MODULES

The following paragraphs discuss object modules constructed using PSEGS and DSEGS. An object module can contain a PSEG only, a DSEG only, or both a PSEG and a DSEG.

An object module using the PSEG/DSEG structure should contain only the following in the PSEG portion:

- Unmodifiable instructions.
- Constant data.

If the object module contains a DSEG, the DSEG can contain modifiable data.

The Link Editor always positions the PSEG portion of an object module in the segment in which it is included. It always positions the DSEG portion in the task segment.

5.2.1 Differences in the Treatment of Sharable Vs. Reentrant Modules

In a sharable object module, data outside the PSEG can be directly addressed if the ALLOCATE command of the Link Editor is properly used during link edit. In a reentrant object module, all referencing of data outside the PSEG must be by means of indirect addressing.

5.2.2 COBOL Object Modules

An object module generated by the COBOL compiler is constructed using the PSEG/ DSEG structure. PSEGs directly address data in DSEGs; therefore, the ALLOCATE command of the Link Editor must be used in order to share COBOL object modules. COBOL object modules that use segmentation cannot be shared.

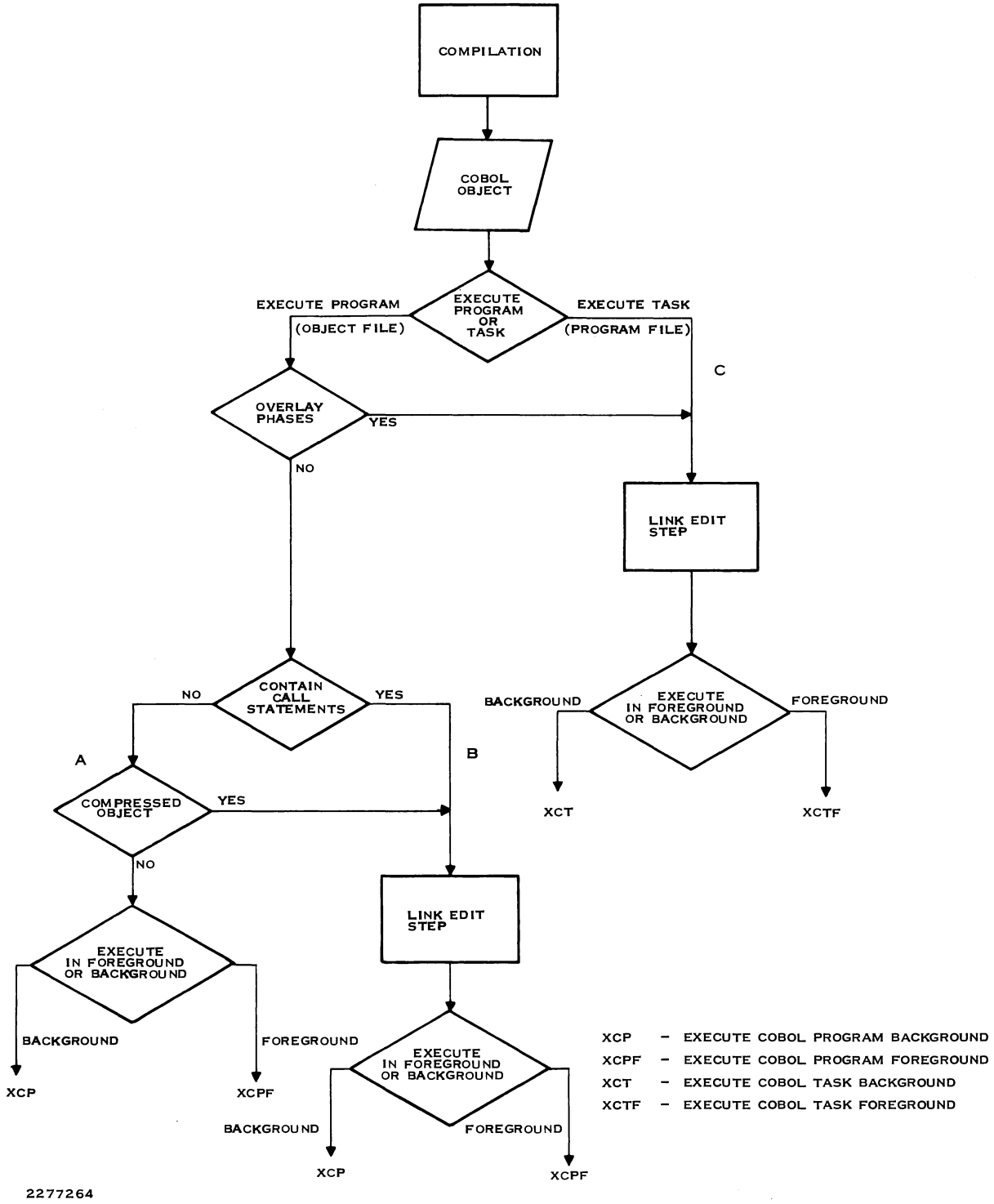


Figure 5-1. Determining Link Edit Requirements for COBOL Programs

5.3 PROGRAM MAPPING

The hardware has a 20-bit memory address bus and can address 2048 bytes of memory. The logical address space available to a task (program) is limited to 64K bytes. This difference is resolved by mapping the task's logical address space into the computer's physical address space. The segments in physical address space need not be contiguous. Since the operating system maintains separate mapping parameters for each task, each task may consist of one, two, or three segments with a total extent of 64K bytes. Furthermore, several tasks may share one or two procedure segments. However, one segment is unique to each instance of a program. This unique segment is called the task segment (T). The sharable segments of a task are called procedure segments (P1) and (P2). Refer to Figure 5-2.

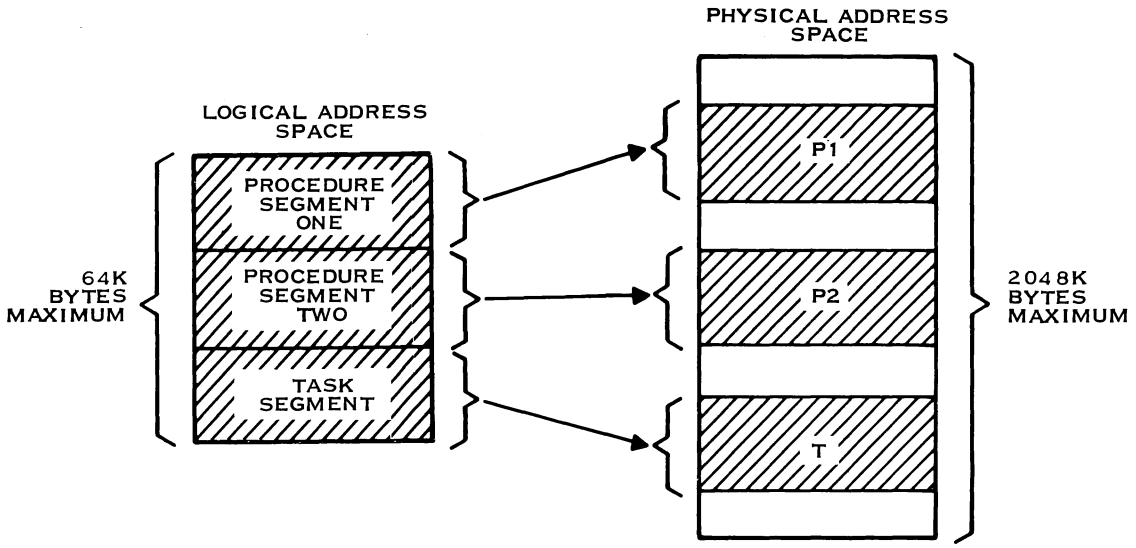


Figure 5-2. Memory Mapping

5.4 PROGRAM FILES

All task and procedure segments and overlays are installed in structures referred to as *program files*. These files are similar to the expandable relative record files and contain program images in blocks corresponding to file records. An internal directory is maintained within the file itself. This internal directory contains pointers to each image on the file as well as relevant information about the images. Figure 5-3 shows a listing of a program file produced by the Map Program File (MPF) command.

```
      FILE MAP OF VOLUME.PROG
        TODAY IS 15:58:24 WEDNESDAY, JUN 04, 1980.

TASK SEGMENTS: MAXIMUM POSSIBLE = 255
 ID    NAME   LENGTH LOAD PRI  S P M R D E O C OVLY     P1/SAME P2/SAME INSTALLED
 01  TSKSEG1   136A  0000  3        R           04                      3/26/80
 02  TSKSEG2   7082  0000  3        R                                   5/ 7/80
 03  TSKSEG3   12E2  4440  4        R                    01/Y           5/17/80
 04  TSKSEG4   AFA4  4060  4        R           06       02/Y           6/10/80

PROCEDURE: MAXIMUM POSSIBLE = 255
 ID    NAME   LENGTH LOAD     M D E W C                                INSTALLED
 01  PRCSEG1   4438  0000                                              5/17/80
 02  PRCSEG2   4050  0000                                              6/10/80

OVERLAYS: MAXIMUM POSSIBLE = 255
 ID    NAME   LENGTH LOAD   MAP  D  OVLY                              INSTALLED
 01  OVLY1     05B6  0006                                             5/ 7/80
 02  OVLY2     13F4  0006           01                                5/ 7/80
 03  OVLY3     1394  0006           02                                5/ 7/80
 04  OVLY4     1148  0006           03                                5/ 7/80
 05  OVLY5     119E  AE9A                                             6/10/80
 06  OVLY6     2E7C  AE9A           05                                6/10/80
```

Figure 5-3. Contents of a Program File

In Figure 5-3, task 1 consists of task segment 1. Task 2 consists of task segment 2 and overlays 1 through 4. Task 3 consists of task segment 3 and procedure segment 1. Task 4 consists of task segment 4, procedure segment 2, and overlays 5 and 6. Various examples of how to create linked program images with one, two, or three segments are provided in the Section 5 paragraph entitled Creating Linked Object Modules.

5.4.1 Segments

Because the operating system maintains separate mapping parameters for each task, each task can consist of one, two, or three segments with a total extent of 64K bytes. Furthermore, several tasks may share one or two segments. One segment, however, is unique to each instance of a program. This unique segment is called the task segment. The sharable segments of a task are called procedure segments.

5.4.1.1 Task Segments. Task segments contain the initial portion of the program such as entry vectors, optional data, and optional program code. The task segment is unique to each separate execution and cannot be shared. A task segment may be uniquely replicated from a single image installed in a program file on disk for each activation. Replication of tasks, therefore, conserves disk space and time by eliminating the need to install a copy of the same task with different IDs for each possible concurrent activation of a program.

5.4.1.2 Procedure Segments. A COBOL task can be linked with two or fewer procedure segments. Code linked in the procedure segments can be shared by more than one task. A procedure is considered sharable if more than one task can share one copy of the module during execution without loss of data. Reentrant (or pure) procedures must contain only unmodifiable code and constant data. Data modified by the reentrant module is usually stored in the task segment and can be located at different addresses in the tasks without loss of data. The COBOL run-time interpreter module is reentrant. All reentrant procedures are sharable.

The procedure portion (PSEG) of the object generated by the COBOL compiler is not reentrant. It can be made sharable through the use of the ALLOCATE command in the link control file. (Refer to paragraph 5.6.5 entitled Linking Two Procedure Segments With Multiple Task Segments for an explanation of how to use the ALLOCATE command.) Procedure segments are linked by use of the PROCEDURE command as referenced in the *Link Editor Reference Manual*. Sharing procedure segments conserves memory by precluding the replication of a task's procedure segment.

Procedure segments installed on the system program file can be shared by tasks in any user program file. Procedure segments installed on a user program file can be shared only by tasks on that program file.

The COBOL run-time interpreter (RCOBOL) is stored in the system program file. To conserve both memory and disk space, it is recommended that COBOL tasks share this procedure.

If task 1 and task 2 reside on the same program file and each share the same procedure(s) (either on the same program file as the task or on the system program file), only one copy of any shared procedure segment is in memory during execution of the tasks.

Conversely, if task 1 and task 2 are on separate program files and each has a copy of the same procedure(s), then two copies of the procedure(s) occur in memory during simultaneous execution of the tasks.

Figure 5-4 shows a construct with multiple task and procedure segments on the same program file. Each task segment is attached to the procedure segment. Therefore, sharing P1 and P2 reduces the amount of memory required to run the application. The task segments may be identical (that is, duplicated and/or executed from two different terminals) or they may be unique task segments. Tasks on separate program files that share the same procedure(s) on the system program file require only one copy of the procedure(s) in memory during concurrent execution of the tasks.

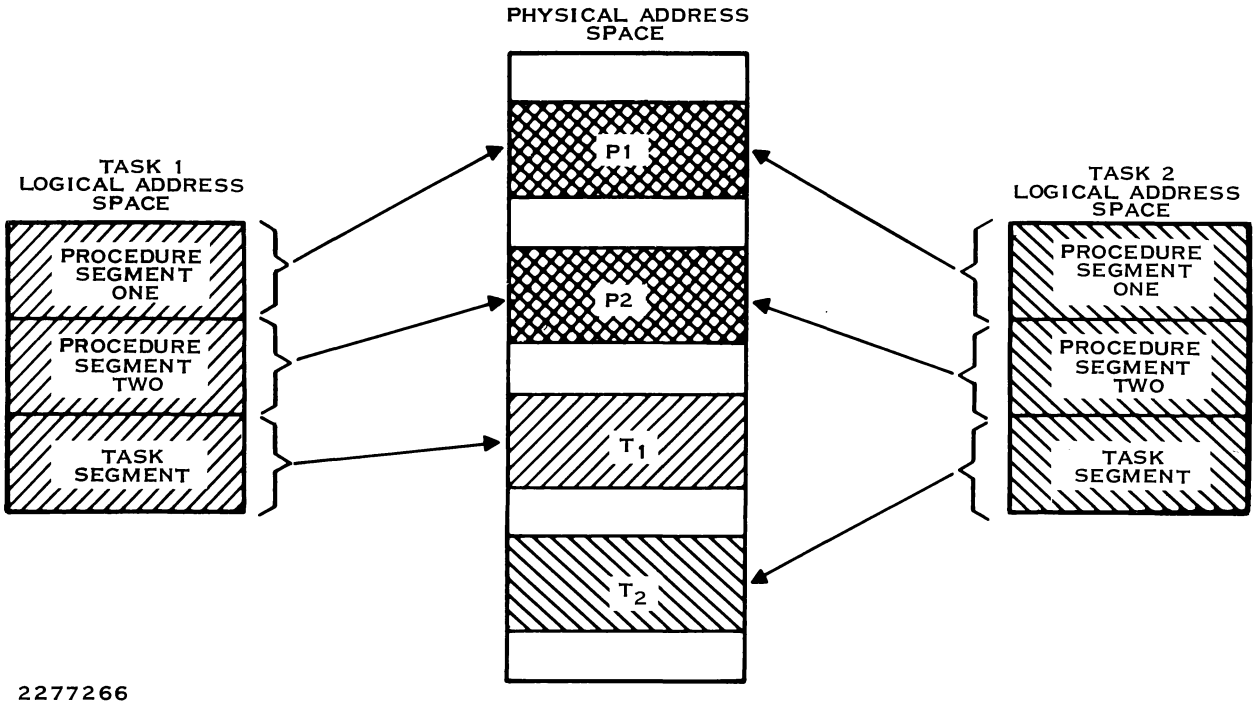


Figure 5-4. Multiple Tasks Sharing Same P1 and P2

Figure 5-5 shows another construct with multiple task and procedure segments on the same program file. Task segments 1 and 2 share the first P2 with P1 while task segments 3 and 4 share the second P2 with P1.

Figure 5-6 shows a construct with task and procedure segments on separate nonsystem program files.

Figure 5-7 illustrates the importance of sharing procedure segments. The total memory required to execute the group of tasks shown in Figure 5-7 is 215K bytes (1K = 1024 bytes) if procedure segments are not shared. If procedure segments are shared, only 130K bytes are required. Nearly half of the memory required to execute this group of tasks has been eliminated. In many cases, such a reduction can mean reduced swapping and, consequently, faster execution time.

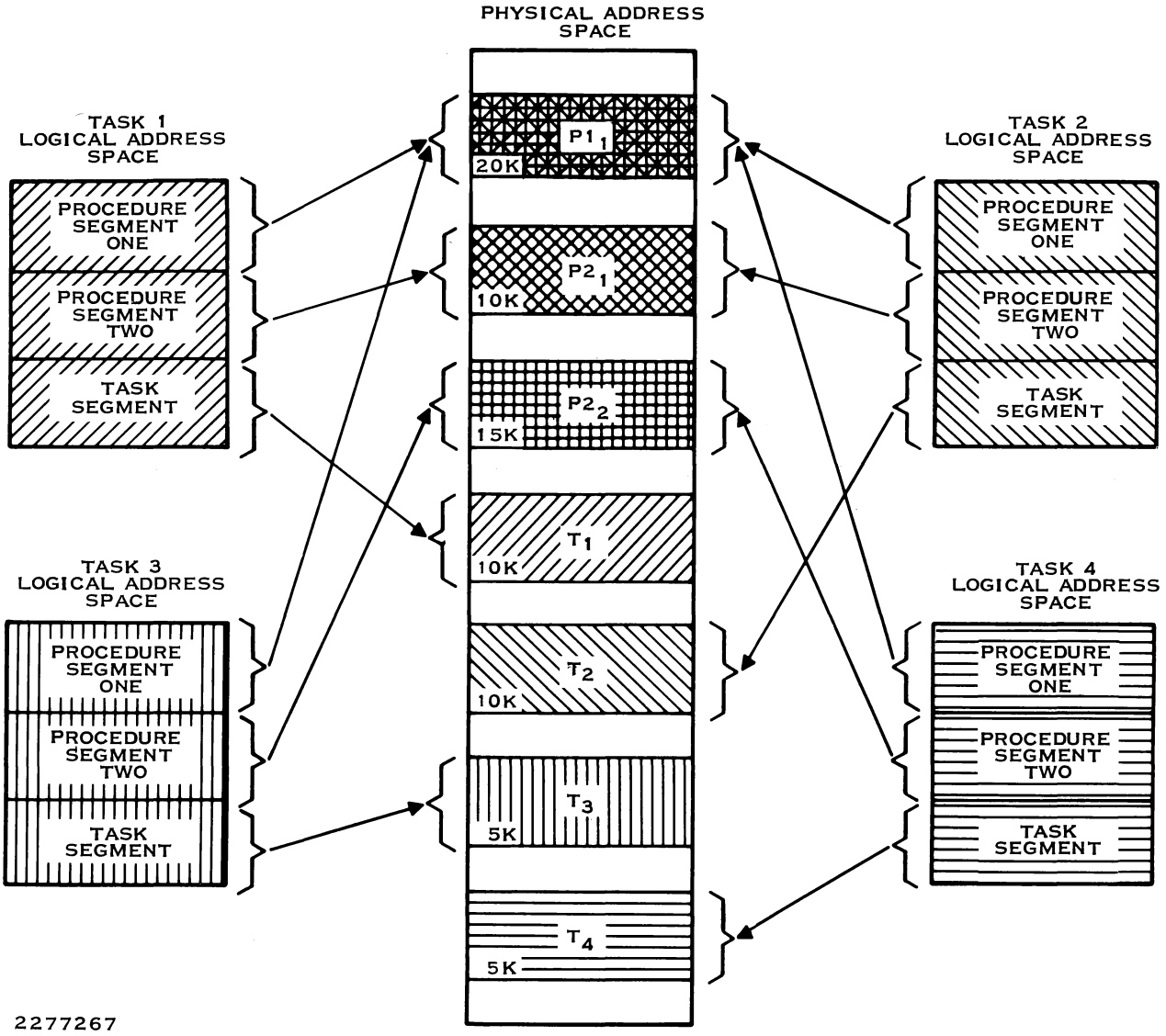


Figure 5-5. Multiple Tasks Sharing Same P1 but Different P2s

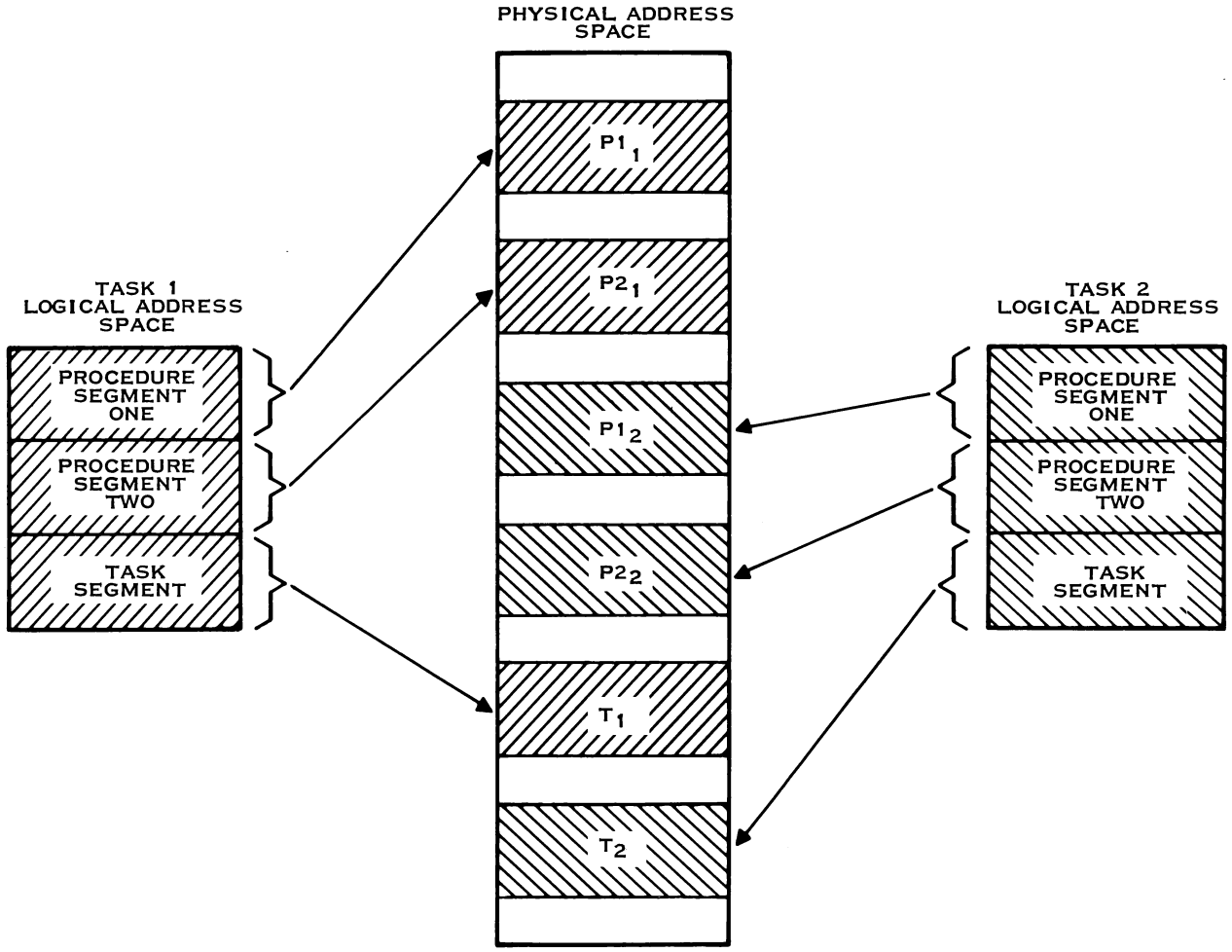


Figure 5-6. Multiple Tasks on Separate Program Files

	REENTRANT PROCEDURES NOT SHARED					REENTRANT PROCEDURES SHARED				
	P1	$P2_1$	$P2_2$	TASK	TOTAL	P1	$P2_1$	$P2_2$	TASK	TOTAL
T_1	20K	10K	-	10K	40K	20K	10K	-	10K	40K
T_2	20K	10K	-	10K	40K	-	-	-	10K	10K
T_3	20K	-	15K	5K	40K	-	-	15K	5K	20K
T_4	20K	-	15K	5K	40K	-	-	-	5K	5K
	TOTAL MEMORY				160K	TOTAL MEMORY				75K

2277269

Figure 5-7. Comparison of Memory Requirements

5.4.2 Overlays

Overlays are parts of a task that reside on disk until explicitly requested by the task. When requested, an overlay is loaded into an area of the task reserved for overlays and replaces any other overlay which may have been present at the time of the request. The use of overlays can reduce the amount of memory required by a task segment.

An *overlay phase* is the smallest functional unit that can be loaded as a logical entity during execution. A phase consists of one or more object modules. The structure of an overlaid program depends on the relationships between the phases in the program. Phases that need not be in memory at the same time can overlay each other. These phases are independent in that they do not reference each other, either directly or indirectly. Independent phases can be assigned the same load address and are loaded into memory only when referenced. The *Link Editor Reference Manual* contains a detailed description of overlays and overlay phases.

5.4.3 COBOL Module Segmentation

COBOL module segmentation is a type of overlay. *COBOL segmentation* provides a means of communicating with the compiler when specifying requirements of the object program module overlay. A task (program) may be structured to include COBOL segment overlays and also may include overlay phases.

Any COBOL module in the task segment, including modules within overlay phases, can contain segments. COBOL module segments are automatically generated in the object module when specified in the source module. All segments are assigned the name COBOVY. Figure 5-8 shows a map program file listing containing overlay phases with embedded COBOL segments. When creating program images on program files, segments are contained in the program file as overlay entries. Refer to Figure 5-8. The module T.SEGMENT is a segmented COBOL module in an overlay phase. T.NONSEG is a nonsegmented COBOL module in an overlay phase. Both overlay phases and the COBOL segments are listed as overlay entries in the map program file listing.

COBOL segmentation deals only with the segmentation of the Procedure Division (PSEGs) of a COBOL program module. Two types of PSEGs are fixed and independent. The fixed portion is the part of the object program that is logically treated as if it were always in memory. An independent segment is the part of the object program that can overlay or be overlaid by another independent segment. The TI *COBOL Reference Manual* contains a detailed description of COBOL segmentation.

```
      FILE MAP OF .DON020.PROG
        TODAY IS 12:57:26 WEDNESDAY, SEP 10, 1980.

TASKS: MAXIMUM POSSIBLE =    1
 ID   NAME   LENGTH LOAD PRI  S P M R D E O C  OVLY P1/SAME P2/SAME  INSTALLED
 01 OVLY     1A5E  3D20   4        R           05  01/Y               9/10/80

PROCEDURES: MAXIMUM POSSIBLE =  1
 ID   NAME   LENGTH LOAD     M D E W C                               INSTALLED
 01 RTCOBOL  3D18  0000                                              9/10/80

OVERLAYS: MAXIMUM POSSIBLE =   5
 ID   NAME   LENGTH LOAD   MAP  D  OVLY                             INSTALLED
 01 SEGMNT   02D0  533C                                             9/10/80
 02 NONSEG   0442  533C            01                               9/10/80
 03 COBOVY   00DA  5530            02                               9/10/80
 04 COBOVY   00DA  5530            03                               9/10/80
 05 COBOVY   00DA  5530            04                               9/10/80
```

Figure 5-8. COBOL Segmentation Within Overlay Phase Modules

5.5 CREATING LINKED OBJECT MODULES

Table 5-1 contains a list of valid link editor commands for COBOL linking object modules.

Table 5-1. Valid Link Editor Commands With COBOL Object

Command (Default Underscored)	Partial Link	Execute (From Object File)	Execute (From Program File)
ADJUST	Y	Y	Y
ALLOCATE	NO	NO	Y
AUTO	[1]	Y	Y
COMMON	NO	NO	NO
DATA	NO	NO	NO
DUMMY	Y	NO	Y
END	Y	Y	Y
ERROR/NO ERROR	Y	Y	Y
FORMAT ASCII	Y	Y	NO
FORMAT COMPRESSED	Y	Y	NO
FORMAT IMAGE	NO	NO	Y
FORMAT IMAGE, REPLACE	NO	NO	Y
GLOBAL/ALL GLOBAL/ NOT GLOBAL	Y	NO	NO
INCLUDE	Y	[2]	[3]
LIBRARY	Y	Y	Y
LOAD/NO LOAD	NO	NO	Y
MAP/NO MAP	Y	Y	Y
NOAUTO	Y	Y	Y
NOSYMT	Y	Y	Y
PAGE/NO PAGE	Y	Y	Y
PARTIAL	Y	NO	NO
PHASE 0	Y	Y	Y
PHASE 1,2,...n	NO	NO	Y
PROCEDURE	NO	NO	Y
PROGRAM	NO	NO	NO
SEARCH	Y	Y	Y
SHARE	NO	NO	NO
SYMT	Y	Y	NO
TASK	Y	Y	Y

Notes:

[1] For a PARTIAL link, the default is NO AUTO and these commands should be omitted.

[2] Main program must be included first.

[3] COBOL run-time procedure, task, and main program designator modules must be included as part of the link.

Overlay phases are not allowed with linked object modules.

A linked object module must be produced in one of the following distinct formats:

- Tagged
- Compressed

Tagged object modules consist of ASCII characters with ASCII tags. Compressed object modules also have tags, but the numeric characters are changed to binary representations.

Compared to the normal tagged object, the compressed object saves approximately 47 percent of disk space.

The following example of a link control file shows how to generate a tagged object module:

```
TASK CBLTSK1
INCLUDE EX.MAINPRG1
INCLUDE EX.SUBPRGM
END
```

The following example of a link control file shows how to generate a compressed object module:

```
FORMAT COMPRESSED
TASK CBLTSK1
INCLUDE EX.MAINPRG1
INCLUDE EX.SUBPRGM
END
```

Note that the only difference between the two sets of link control commands is the FORMAT command. The default format of the linked output is tagged (ASCII). The FORMAT command is not required for tagged format. In both cases, the link editor resolves external addresses or references. Object modules or linked object modules are executed by using the XCP or XCPF commands. Section 6 contains information for executing an object module or a linked object module.

5.6 CREATING PROGRAM IMAGES

For object modules produced by the link editor and installed on program files, the link editor must link the program modules to the run-time interpreter module. Object modules are installed and stored on program files in memory image form. The link editor may install the memory image object directly on a program file. When the necessary program file does not exist, it is automatically created. The link editor creates a program file with only enough room for the task and procedure segments and overlays defined for the program. If a program file is created by the Create Program File (CFPRO) command, the operating system allows a maximum of 255 task segments, 255 procedure segments, and 255 overlays.

Program images are executed by using the XCT or XCTF commands. Section 6 contains information for executing object modules produced by the link editor and installed on program files.

5.6.1 COBOL Run Time

COBOL run time consists of the following prelinked object modules:

- .S$SYSLIB.RCBTSK — This module contains the task entry vector plus the data area portion of COBOL run time needed by the reentrant module RCBPRC. It must be included as the first module in the task segment of the task. It is not reentrant.

- .S$SYSLIB.RCBTSKD — This module includes everything contained in .S$SYSLIB.RCBTSK and the COBOL debugger module needed when performing interactive debugging of COBOL modules.

- .S$SYSLIB.RCBPRC — This is the reentrant module that contains the COBOL run-time interpreter and can be included in a procedure segment of a task when desired.

- .S$SYSLIB.RCBNOIO — This module is similar to .S$SYSLIB.RCBPRC with the exception that any modules comprising the run-time interpreter relating to I/O operations are omitted.

- .S$SYSLIB.RCBMPD — This module must be stored during Link Edit immediately preceding the COBOL object module intended to receive control at execution time. It then designates to the run time where the object module begins. Since it is reentrant, it can be used in either task or procedure segments.

The run-time entry module (.S$SYSLIB.RCBTSK), one of the two reentrant modules (.S$SYSLIB.RCBPRC or .S$SYSLIB.RCBNOIO), and the main program designator module (.S$SYSLIB.RCBMPD) can be specifically included in the appropriate places in the link control file. The reentrant module .S$SYSLIB.RCBNOIO cannot be linked with the run-time entry module .S$SYSLIB.RCBTSKD. The reentrant module .S$SYSLIB.RCBPRC (or .S$SYSLIB.RCBNOIO) can be included anywhere in the link control file except as the first module in the task segment (phase zero). If .S$SYSLIB.RCBPRC is used, it is suggested that it be made P1, so that the shared procedure segment on the system program file can be used. If .S$SYSLIB.RCBPRC is anywhere other than P1, a separate copy is generated in the user program file and in memory when the program is executed. When the first program module to receive control is a COBOL program module, the run-time entry module (.S$SYSLIB.RCBTSK or .S$SYSLIB.RCBTSKD) must be the first module included in the task (phase zero) since it contains the task entry vector. The main program designator (.S$SYSLIB.RCBMPD) module must be included just prior to the COBOL program module that receives control. The following paragraphs demonstrate various techniques for linking these modules with user modules to build tasks.

5.6.2 Linking a Single Procedure Segment With a Single Task Segment

The COBOL reentrant run-time interpreter module is installed by the COBOL installation on the system program file as the reentrant procedure segment RCOBOL. This procedure segment is identical to .S$SYSLIB.RCBPRC and can be shared by all user tasks that have been linked and installed on user-defined program files. Using this procedure segment eliminates the need for a copy of .S$SYSLIB.RCBPRC on each user-defined program file, thus saving disk storage. If you have two user-defined program files and .S$SYSLIB.RCBPRC is installed on each, executing one task from each program file loads two copies of .S$SYSLIB.RCBPRC into memory. If the procedure segment on the system program file is used, only one copy of the reentrant procedure segment is in memory during the execution of the tasks, thus saving memory space and minimizing swapping.

Figure 5-9 shows a simple link edit using the system program file procedure segment RCOBOL.

The presence of the DUMMY command in the link control file prevents the procedure segment from being replaced in the program file.

This procedure segment (RCOBOL) on the system program file must be used only in the link procedure segment one (P1).

The procedure segment two (P2) and the task segments (T) may be structured using any of the techniques mentioned in paragraphs 5.6.3 through 5.6.5. All examples use the shared procedure segment RCOBOL. *The origin addresses and lengths in the following figures do not necessarily reflect the actual origin and lengths of the TI COBOL run time.*

To use RCOBOL on the system program file, the DUMMY command must always be specified, even on the first link edit to a new program file. The procedure segment RCOBOL must not already exist on the user program file. The reentrant procedure segment on the system program file is identical to .S$SYSLIB.RCBPRC.

5.6.3 Linking a Single Procedure Segment With Multiple Task Segments

A single procedure segment may be shared by multiple tasks. The task segments must be linked and installed on the same program file. They will then be attached to this shared procedure segment. Figure 5-10 presents the structure shown in Figure 5-9 with an additional task segment attached to the procedure segment. A link control file is shown on the right side of Figure 5-9. When sharing a single procedure segment, all link control files must be identical within the procedure segment. If any change is required in the procedure segment, all tasks on the program file must be linked again.

5.6.4 Linking Two Procedure Segments With a Single Task Segment

A task segment may be attached to multiple procedure segments. Figure 5-11 shows the structure of Figure 5-9 with an additional procedure segment added. Note that the DSEG or $DATA (nonreentrant object module code in the form of data) from the procedure segment is relocated to the task segment immediately following the task PSEG allocations. All data referenced in procedure segments P1 and P2 must be referenced using indirect or indexed addressing. No direct references can be made to the DSEG. Although the COBOL compiler segregates executable code from data items and the link editor relocates DSEGs by moving them to the task segment, the PSEGs (reentrant object module code in the form of instructions) still reference data items with direct relocatable addresses. Reentrant execution is permitted by locating the DSEG at the same absolute location in each task segment. Assembly language object modules can also be made reentrant through the use of PSEG and DSEG assembler directives.

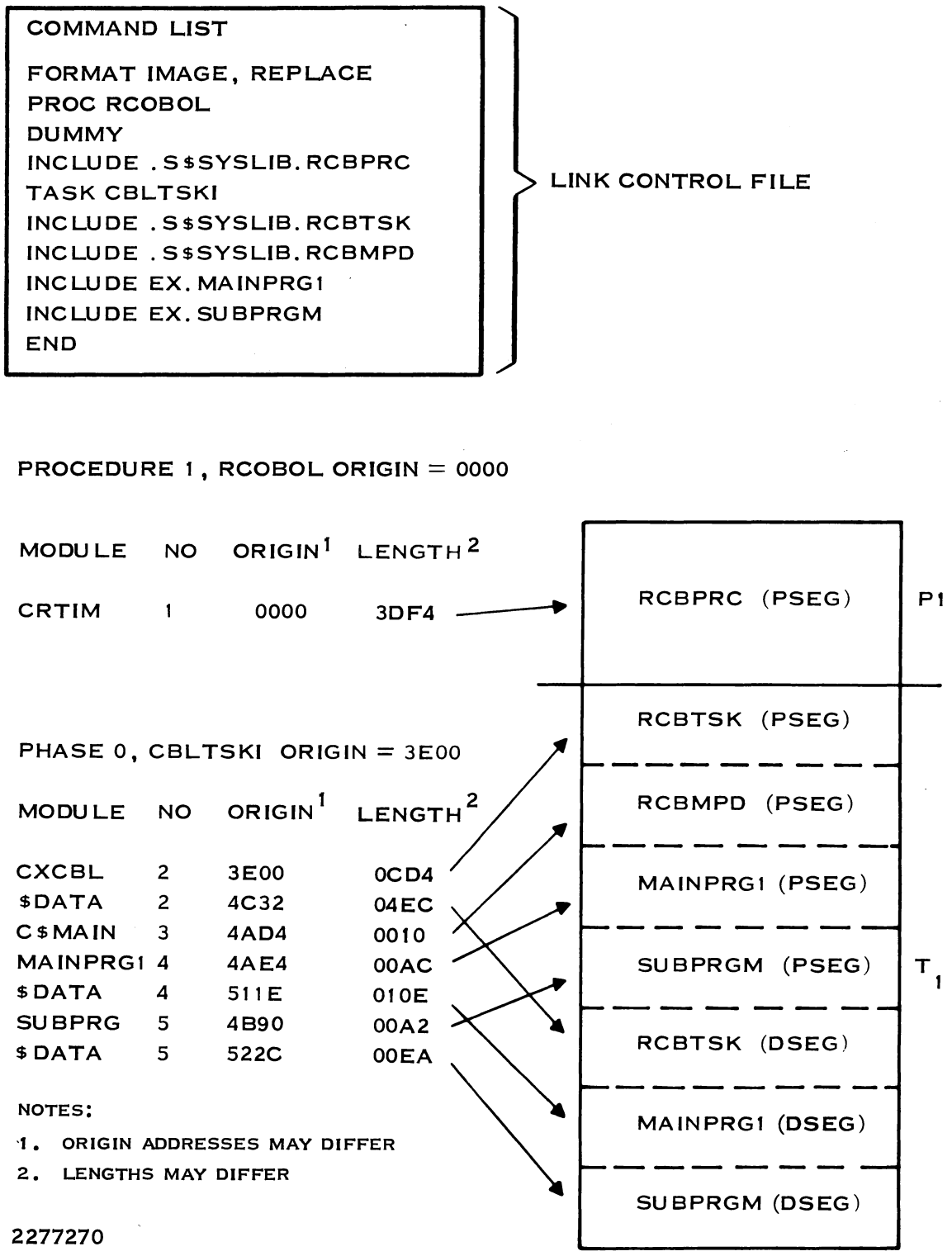


Figure 5-9. Linking a Single Procedure Segment With a Single Task Segment

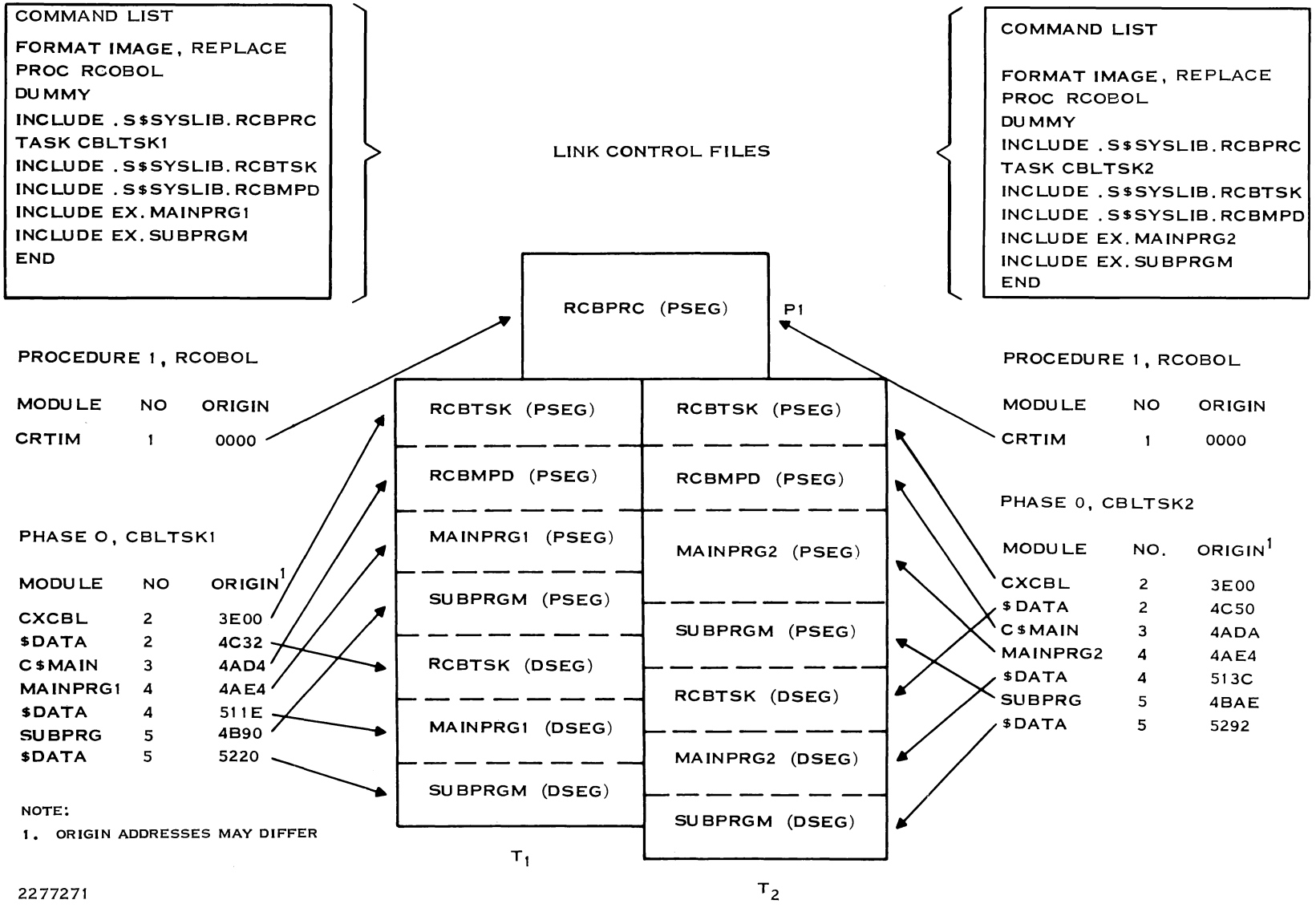


Figure 5-10. Linking a Single Procedure Segment With Multiple Task Segments

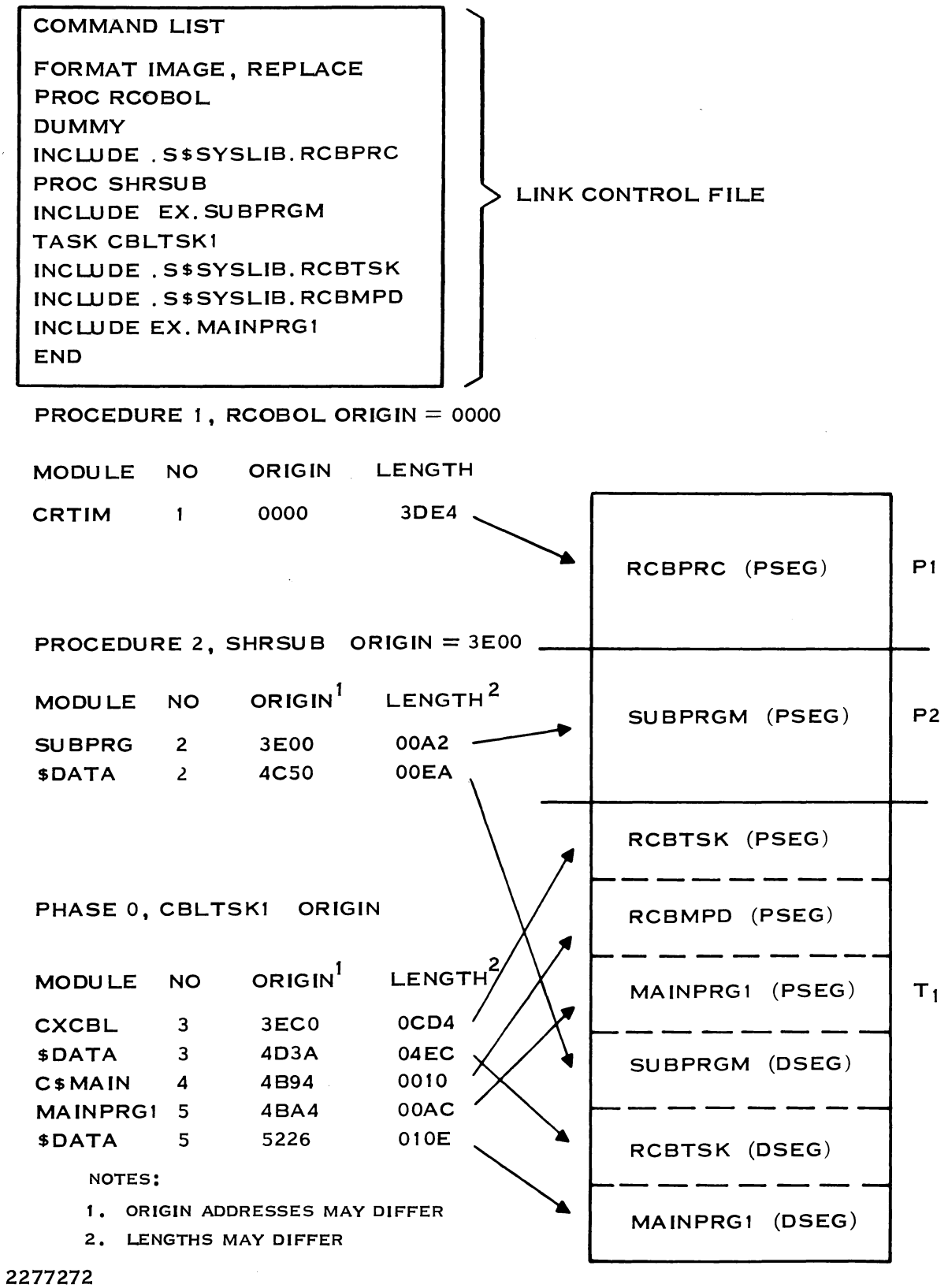


Figure 5-11. Linking Two Procedure Segments With a Single Task Segment

5.6.5 Linking Two Procedure Segments With Multiple Task Segments

Multiple task segments may be attached to multiple procedure segments. Figure 5-12 shows the structure of Figure 5-11 with an additional task segment attached to the procedure segments. Note the allocation addresses shown in Figure 5-12. The origin address for the $DATA (DSEG) associated with SUBPRGM is 4C50 for task T1 and 4C6E for task T2. Since the program SUBPRGM always expects its data to be in the same location, execution of CBLTSK2 will not execute correctly.

This situation is handled by using the ALLOCATE command. The ALLOCATE command allows you to share COBOL program object modules as procedure segments. The ALLOCATE command is always used in the task segment of the link control file. Place the ALLOCATE command after a TASK or PHASE 0 command and before a PHASE 1 or LOAD command, if any are used. The ALLOCATE command should be issued immediately following the INCLUDE.S$SYSLIB.RCBTSK statement and must be placed in the same location in the link control file for all task segments that are sharing COBOL program object modules in P2. The ALLOCATE command causes all DSEGs associated with previously allocated executable PSEGs to be allocated immediately. Space is immediately allocated to all DSEGs associated with PSEGs in either P1 or P2 when the ALLOCATE command occurs in the link control file. Figure 5-13 shows the effects of using the ALLOCATE command when linking two procedure segments with multiple task segments. Note that the origin address for the DSEG for SUBPRGM is 4B94 for both tasks.

If either the link control file statements or a procedure segment in this structure change before the ALLOCATE command is issued, all task segments on the affected program file must be relinked.

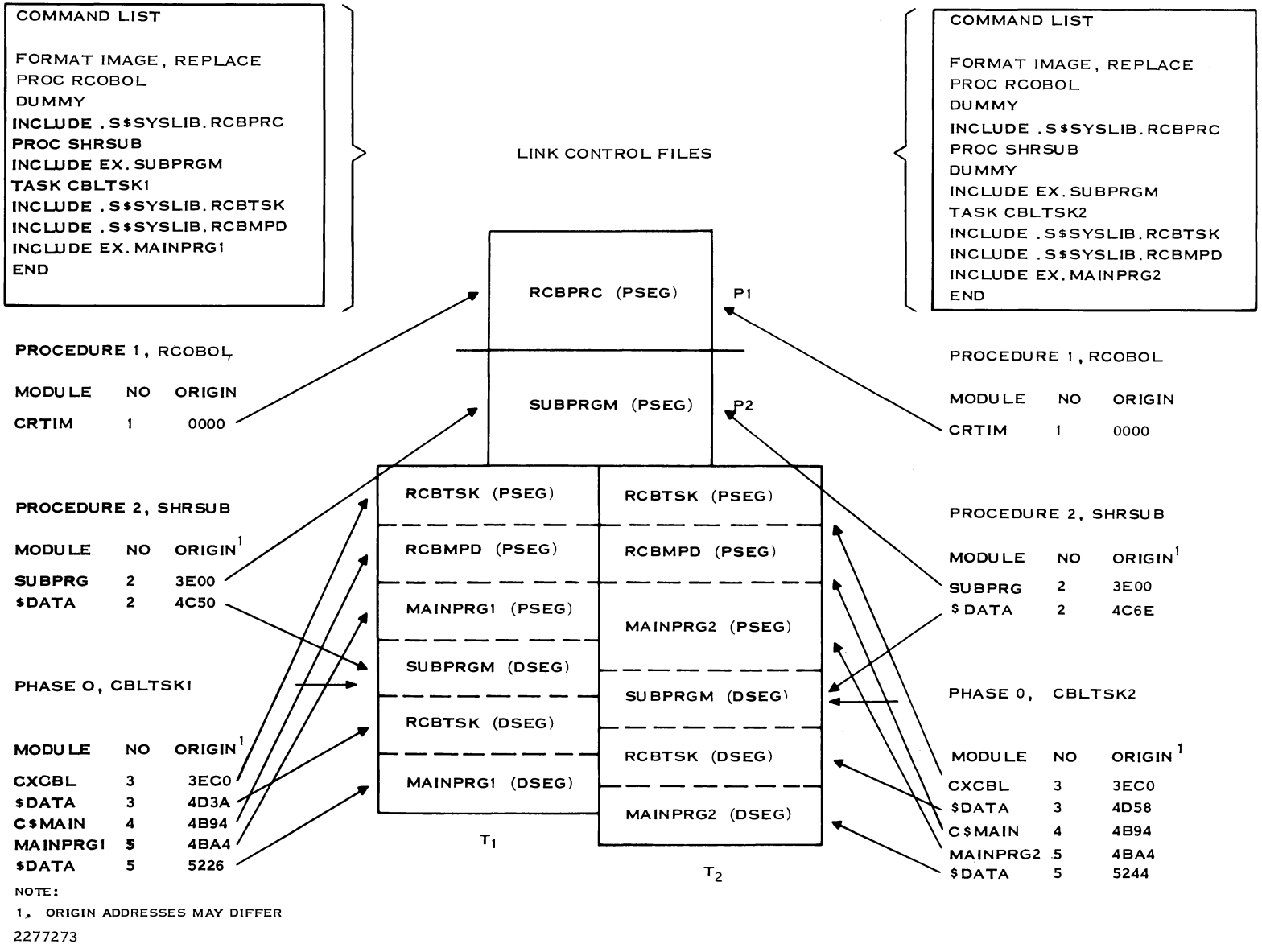


Figure 5-12. Linking Two Procedure Segments With Multiple Task Segments

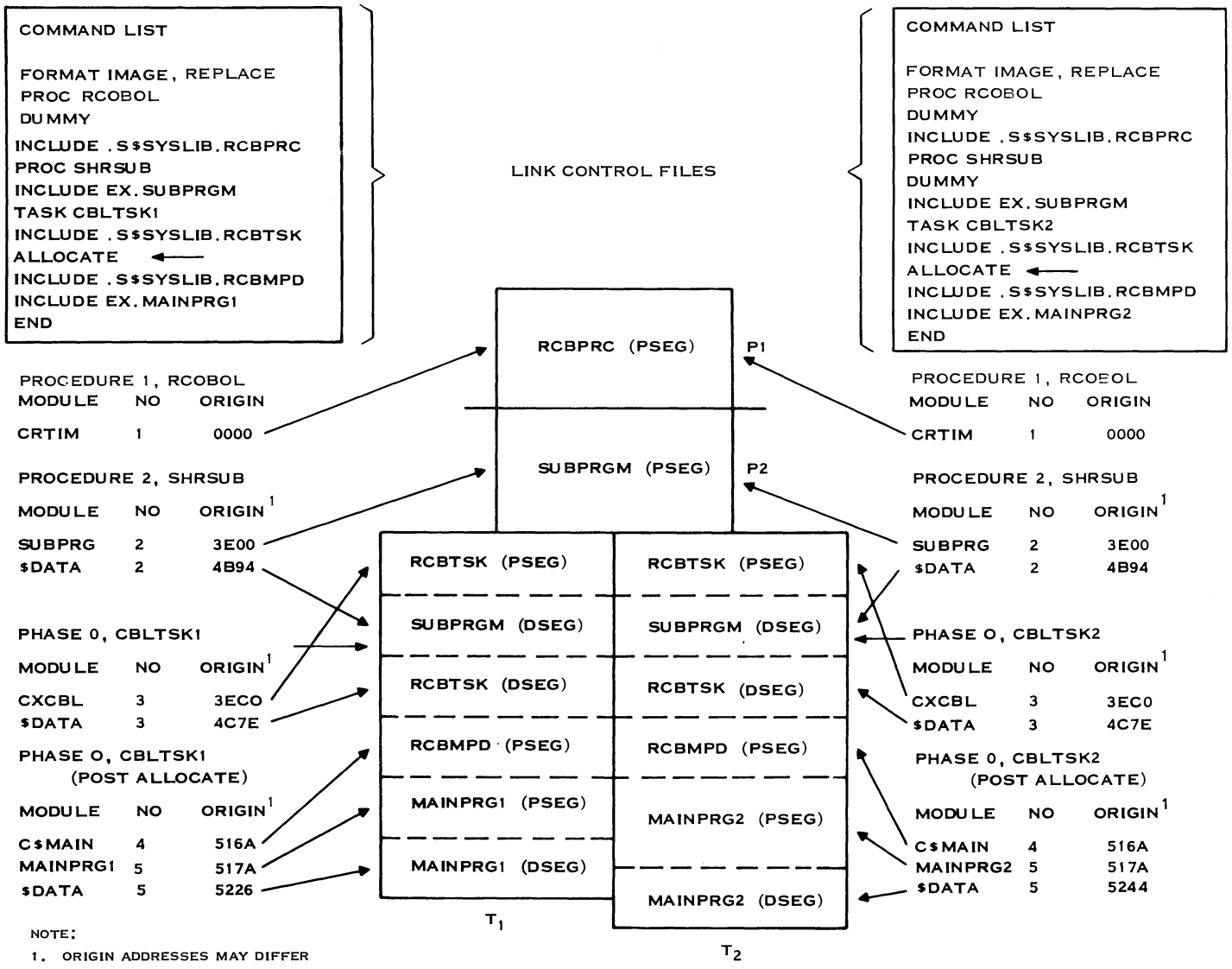


Figure 5-13. Linking Two Procedure Segments With Multiple Task Segments (ALLOCATE)

Object modules that have been separated into PSEGs and DSEGs can be shared successfully if the following conditions are met:

- All modifiable data is contained in the DSEGs. Object modules generated by the COBOL compiler are produced with all modifiable data in the DSEGs.
- If the first procedure segment uses this PSEG/DSEG structure, the second procedure (if used) must be the same length for all tasks that share the first procedure.
- Tasks that share a second procedure must also share the same first procedure.

When using the ALLOCATE command, you can construct a task whose first procedure segment is the reentrant module of the COBOL run-time interpreter (.S$SYSLIB.RCBPRC), and whose second procedure segment is a set of COBOL and/or assembly language program modules. The DSEGs for the routines can be loaded immediately after the run-time interpreter entry module (.S$SYSLIB.RCBTSK) by using the ALLOCATE command. Even though the task segments associated with the two different programs are different, the DSEGs are located in identical locations, allowing direct references in the second procedure segment to be completed successfully.

5.6.6 Overlay Structures

When two or more subroutines are not required to reside in memory simultaneously, an overlay structure can be used to reduce the task's memory requirements. Programs that do not use overlays are loaded into memory and remain in memory until execution completes. Programs that use overlays conserve memory space since each overlay resides in memory only when it is called. The total memory space required by the program is that which is required to hold the root portion of the task segment and the longest overlay path. Overlays are defined by the use of the link control file. Figure 5-14 shows a link control file and tree structure depicting two phase one and two phase two overlays. The location of phase one is after phase zero. The CBLOV10 phase one overlay contains two phase two overlays. The LOAD command allocates the overlay loader module in the appropriate location. (The command LIBRARY .S$SYSLIB must be included in the link control file when using the LOAD command.)

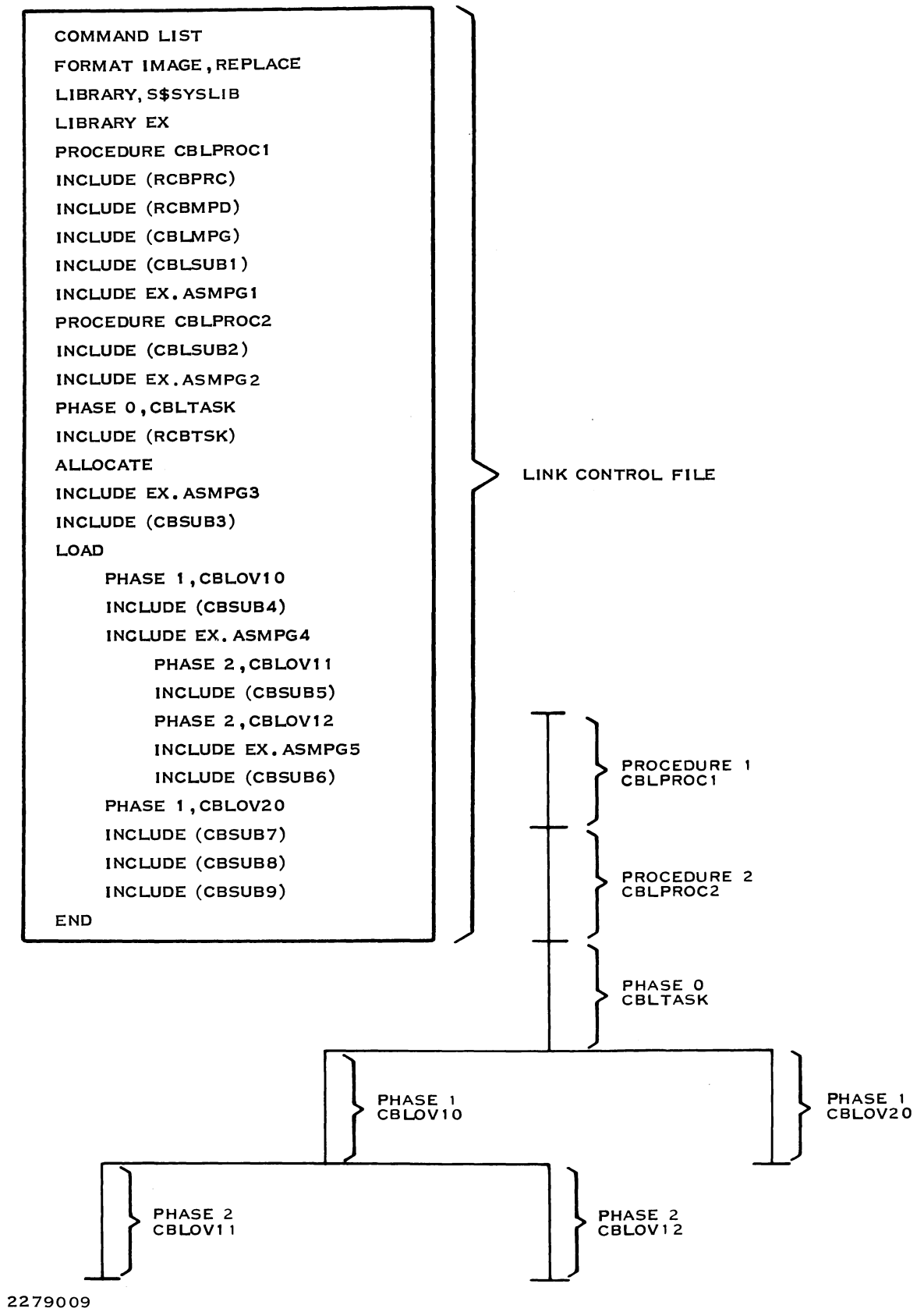


Figure 5-14. An Overlay Structure With the Accompanying Link Control File

The DSEGs for both the CBLPROC1 (P1) and CBLPROC2 (P2) procedure segments float to the end of phase zero (CBLTASK) following the PSEGs of the routines in phase zero. The PSEGs remain in their respective procedure segments. The DSEGs of all phases float to the end of their respective phase immediately following all the PSEGs of the modules in the phases.

It must be noted that if file I/O is performed in an overlay module, the files must be opened on each entry and closed before exiting to release any assigned LUNO. The overlay phase is loaded in its initial state on each entry. However, if consecutive calls are made to the same overlay phase module, the module already resides in memory and is not reinitialized.

5.6.7 Sharing Main Program Module

The main program designator module (.S$SYSLIB.RCBMPD) may be shared with multiple users or terminals. Figure 5-15 shows inclusion of the main program designator module and the user's main COBOL program object module in the P2. The task may be executed from multiple terminals simultaneously, with each task's memory requirements significantly reduced because the main program module is shared among all tasks.

5.6.8 Linking a Single Procedure One Segment and Multiple Procedure Two Segments

Figure 5-16 shows an example of a P1 with different P2s. Applicable to the discussions for this example, which has multiple procedure segments, are the Section 5 paragraphs Linking Two Procedure Segments With a Single Task Segment and Linking Two Procedure Segments With Multiple Task Segments.

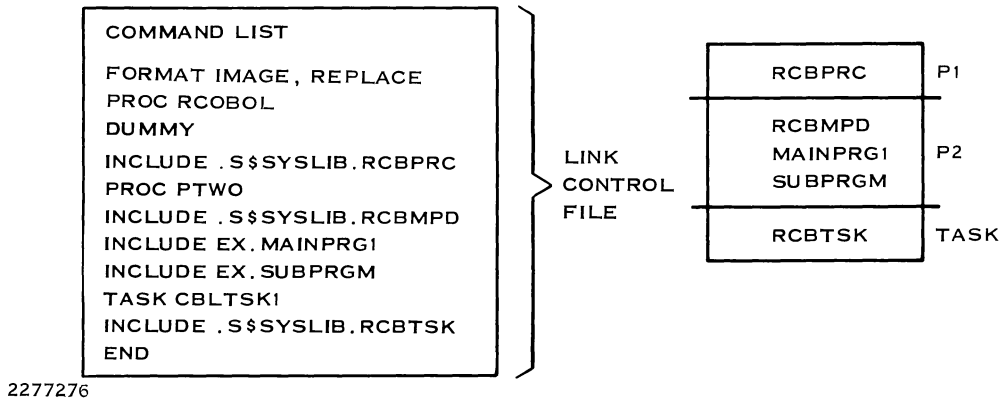


Figure 5-15. Sharing the Main Program Module With P2

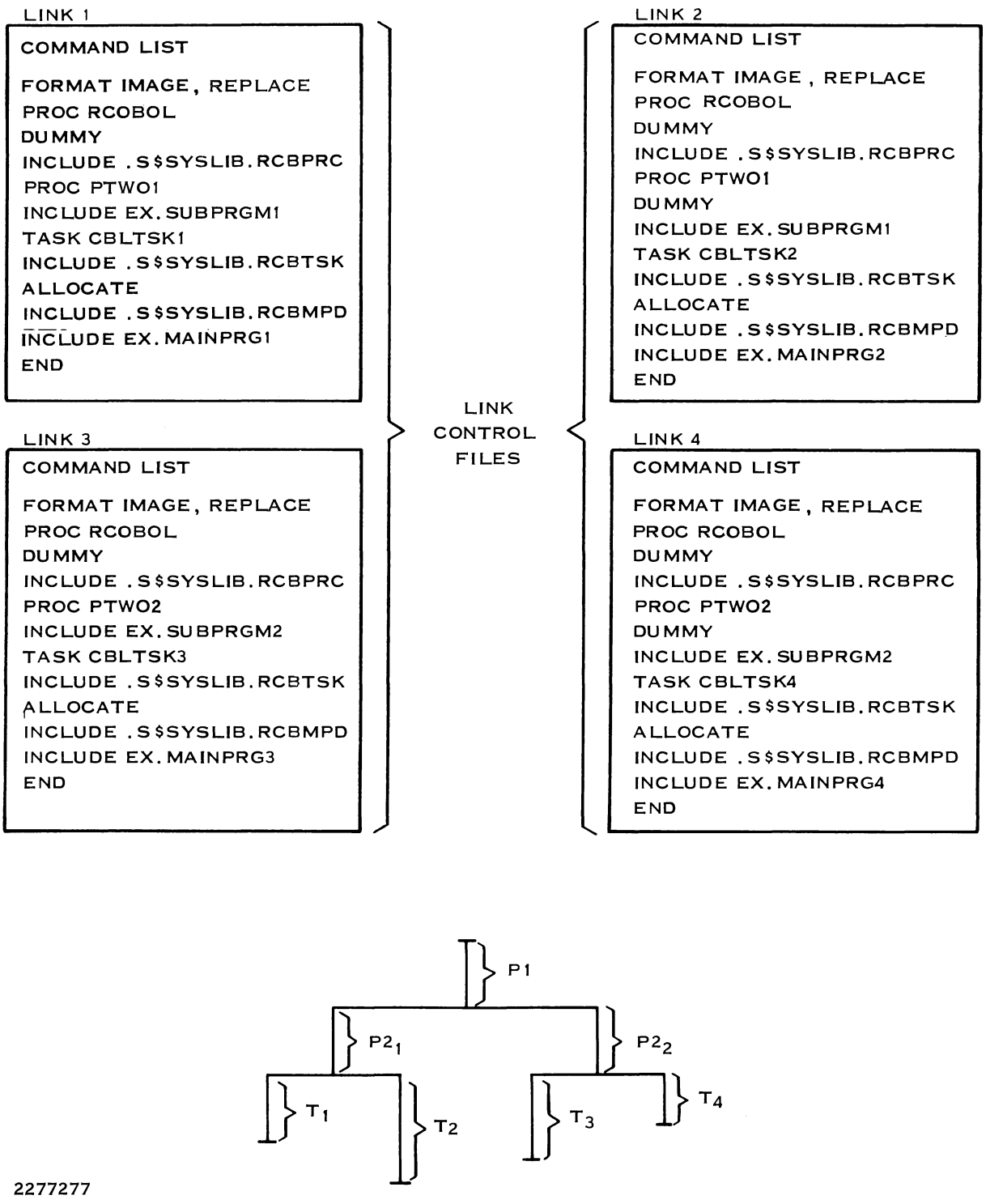


Figure 5-16. Linking a P1 With Different P2s

5.6.9 Linking a Single Procedure Segment With a Single Task

Figure 5-17 shows an example of a single procedure segment linked to a single task segment. Both the procedure segment and the task segment are contained in the user's program file. To include both segments in the user's program file, you can either:

- Specify procedure RCOBOL and omit the DUMMY command in the link control file, or
- Specify a procedure name other than RCOBOL in the link control file.

5.6.10 Installing Program Images From a Relative File

To install the task and procedure segments in a program file, the Install Procedure (IP), Install Task (IT), and Install Overlay (IO) commands are used. A LUNO must be assigned to the relative file and used in the IT, IO, and IP commands. The IP command must be executed before the IT command, which must be executed before the IO command (if applicable), because the link editor outputs the procedure and task segments to a relative file in the order in which they are processed. Relative files are read sequentially by the IP and IT commands; therefore, assigning a LUNO to a relative file prevents the file from being repositioned to the beginning between commands. The following is an example of a link control file linking a procedure segment and task segment, sending output to a relative file.

```
PROCEDURE RCOBOL
INCLUDE .S$SYSLIB.RCBPRC
TASK CBLTSKI
INCLUDE .S$SYSLIB.RCBTSK
INCLUDE .S$SYSLIB.RCBMPD
INCLUDE EX.MAINPRG1
INCLUDE EX.SUBPRGM
END
```

NOTE

A procedure segment and task segment cannot be linked to create a linked object file if any COBOL object modules contain segmentation. COBOL programs with segmentation must be installed automatically by the Link Editor (through the use of the FORMAT IMAGE statement).

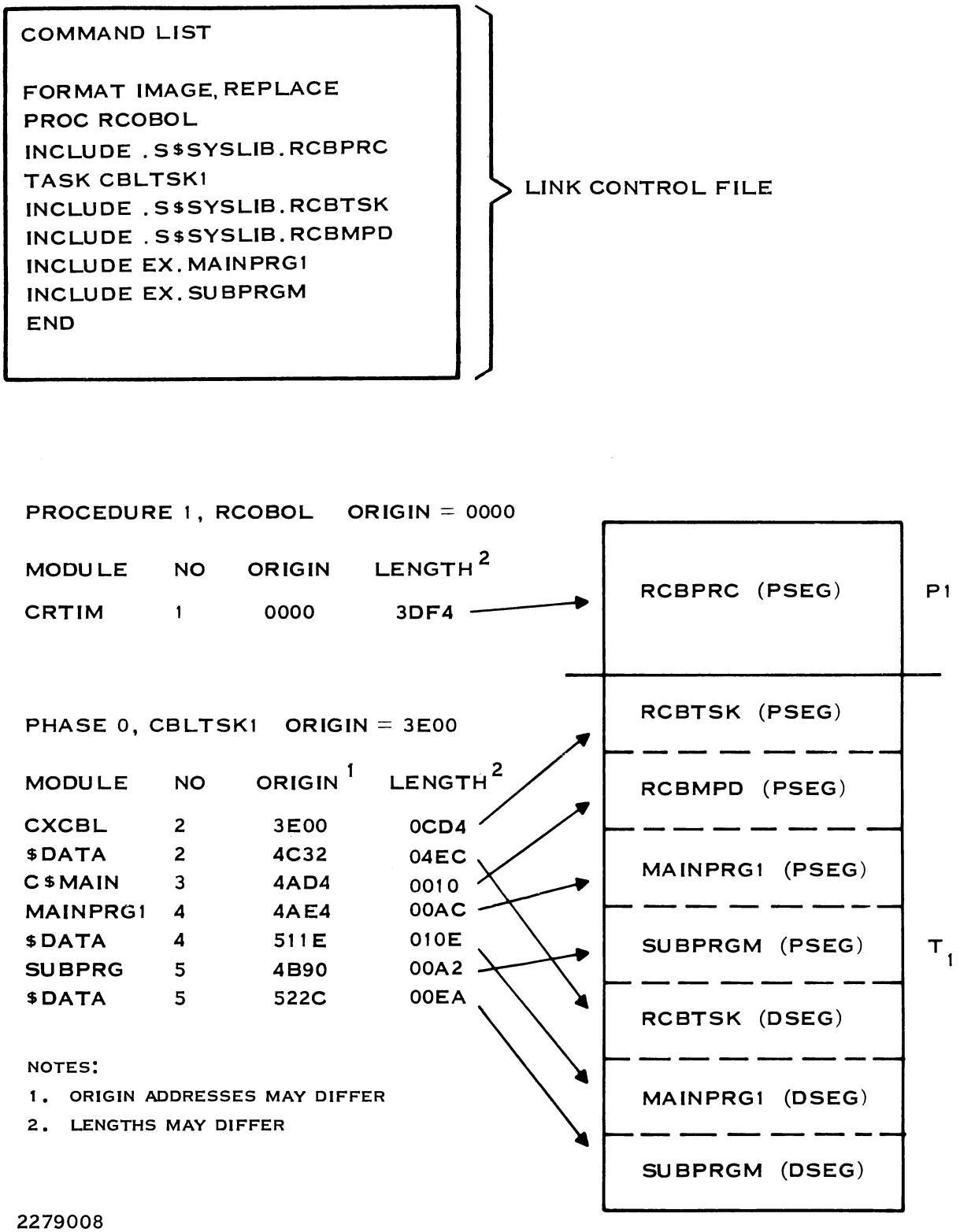


Figure 5-17. Linking a Single Procedure Segment With a Single Task

5.7 LINKING LIBRARIES

The link editor supports two types of library file structures: random libraries and sequential libraries. A *random library* is a directory whose files are the object modules included to resolve external references. Figure 5-18 shows the structure of a random library.

In Figure 5-18, S$SYSLIB, S$PROC, and C$SUBS are directories, with RCBPRC, RCBTSK, RCBMPD, C$TMPF, C$FCFD, C$SCRN, XCCF, and XCPF being data files. Each directory is a node, with the highest level (VCATALOG) being the root node. VCATALOG is assigned a symbolic name when a disk volume is installed or initialized. VCATALOG contains pointers for each directory (node) or file in the level immediately below the VCATALOG. In Figure 5-18, pointers are contained in the VCATALOG for directories S$SYSLIB, S$PROC, and C$SUB.

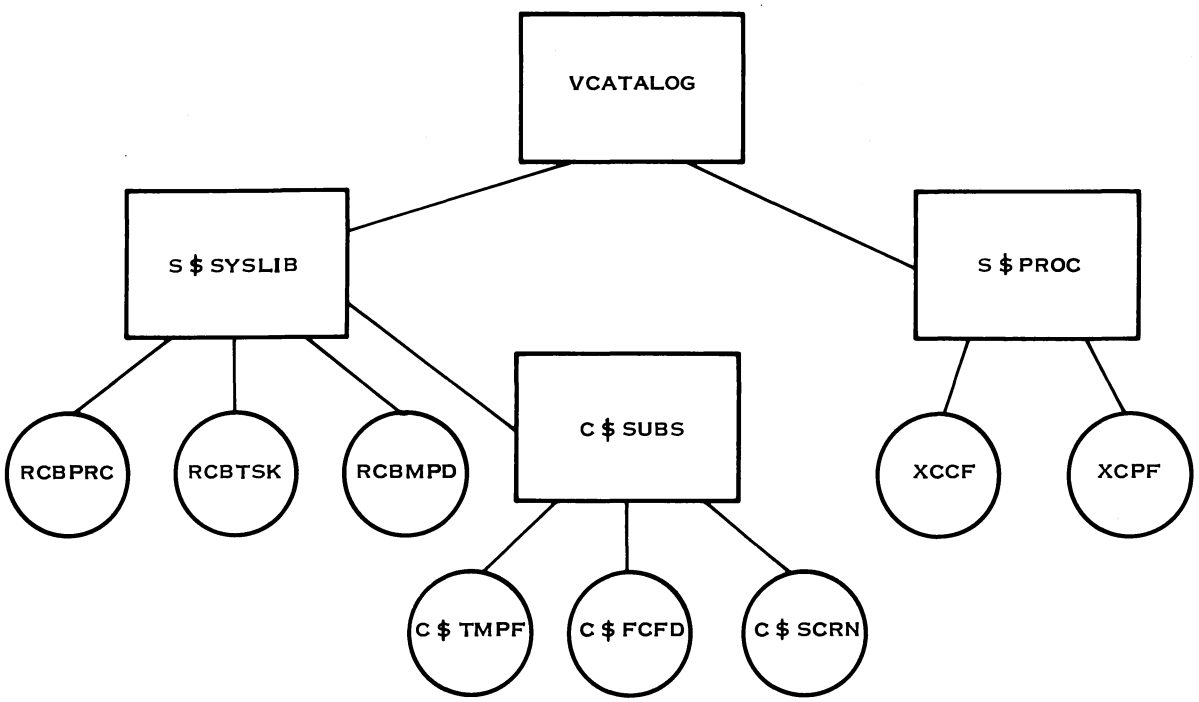


Figure 5-18. Random Library Structure

Modules in a random library can have more than one entry point. However, the secondary entry points are not contained in the directory; consequently, they must be defined to the system as aliases if automatic symbol resolution is being used. An *alias* is an alternate name for a file pathname component. If the module is specifically included (by use of the INCLUDE command), an alias definition is not required.

```
FORMAT IMAGE, REPLACE
LIBRARY .S$SYSLIB.C$SUBS
LIBRARY .SCI990.S$OBJECT
PROCEDURE RCOBOL
DUMMY
INCLUDE .S$SYSLIB.RCBPRC
TASK CBLTSK
INCLUDE .S$SYSLIB.RCBTSK
INCLUDE .S$SYSLIB.RCBMPD
INCLUDE <COBOL object module>
END
```

In the link control file, the INCLUDE command defines modules or files of modules that are to be included in a phase. The LIBRARY command specifies the random or sequential libraries that will be searched to satisfy unresolved external references in the modules to be linked. The link editor automatically processes all control stream commands and then resolves external references in the modules from the libraries specified in the LIBRARY commands. It is possible to use the SEARCH command in link control files instead of the LIBRARY commands. The SEARCH command directs the link editor to perform a search of a library at a particular point in the control stream. However, it is recommended that the LIBRARY command be used when external references need to be resolved. Refer to the *Link Editor Reference Manual* for a detailed explanation of the INCLUDE, LIBRARY, and SEARCH commands and an example of entry points.

A *sequential library* is a sequential file containing one or more object modules generated by a partial link edit. The outputs of the partial link edits are concatenated into a sequential file by use of the Copy Concatenate (CC) or the Append File (AF) commands. The *Link Editor Reference Manual* includes detailed information about sequential libraries and partial link edits.

5.8 LINKING LIMITATIONS

Total memory requirements of a program (task) must be less than the 65,536-byte task address space. Any physical buffers used for blocked I/O do not require space in the user's program because they are allocated as a part of and are maintained by the operating system.

The maximum number of overlays, procedure segments, and task segments permitted in a single program file is 255. If the link editor creates the program file, only enough room is allocated for the task and procedure segments or overlays as needed in the program. The user may create a program file with the desired limitations using the Create Program File (CFPRO) command.

Each phase overlay in the link control file requires one entry in the program file.

6

Execution

6.1 GENERAL

COBOL provides for execution of object modules as well as program images. Object module execution involves the execution of compiler-produced object modules or linked object modules. Program image execution involves the execution of a task that has been installed in a program file.

6.1.1 Use of a Synonym in the COBOL Select Clause

If a synonym is used in a COBOL SELECT statement to define the storage medium (a pathname or device name), the synonym must be assigned prior to execution of the program. Only single level synonym evaluation is performed; that is, the value of a synonym cannot contain another synonym. For example, a synonym named KEYFILE with a value of VOL1.PAYROLL.P00044 is acceptable. A synonym named KEYFILE with a value of A.P00044 (where A is a synonym for VOL1.PAYROLL) is not acceptable. To assign a synonym, use the Assign Synonym (AS) SCI command. The AS command defines a string of one or more characters to substitute for another string of characters.

6.2 OBJECT MODULES EXECUTION

The task loader module, which is included in the reentrant run-time interpreter module, loads the object file into memory. The loader module determines the amount of memory required to contain the interpretive object code, expands the task memory space by the computed amount, and then reads the object file and stores the object code into memory.

Generally, execution of an object module is not used for production programs for the following reasons:

- COBOL program modules that do not require linking are fairly simple since overlay phases and subroutines are not allowed.
- An increased amount of disk and memory space is required to execute an object module.
- The time required to load the object module into memory is increased.

Execution of an object module is permissible under the following conditions:

- The object module must have been produced by the COBOL compiler.
- The object module must be self-contained. Subroutines are not permitted.
- The object module may contain program segmentation.

Execution of a linked object module is permissible under the following conditions:

- The object module must have been produced by the COBOL compiler.
- The object module may contain program segmentation.

Debugging is permitted in the foreground mode only. To execute a COBOL object module or linked object module, enter the XCPF command for foreground execution or the XCP command for background execution. The XCPF command allows the program to use the terminal for I/O operations during execution. The XCP command allows the terminal to be used for other foreground commands during the background execution of the COBOL program.

6.2.1 Execute COBOL Program in Foreground (XCPF)

To execute an object module or a linked object module in the foreground, use the XCPF command. The following prompts appear with the indicated initial values:

```
EXECUTE COBOL PROGRAM FOREGROUND <VERSION: L.R.V YYDDD>
        OBJECT ACCESS NAME: pathname@
                DEBUG MODE: {YES/NO}    (NO)
      MESSAGE ACCESS NAME: [pathname@]
                  SWITCHES: [(integer)] (00000000)
            FUNCTION KEYS: {YES/NO}    (NO)
```

OBJECT ACCESS NAME — Enter the pathname, synonym, or logical name of the file containing the object module.

DEBUG MODE — Enter YES if the program is to execute in the COBOL debug mode. Debugging is permitted in the foreground mode only. The initial value is NO. Section 7 defines debug operations.

MESSAGE ACCESS NAME — No response to this prompt indicates that COBOL system error messages are to be listed to the terminal local file (TLF) of the initiating terminal. The TLF is the default output file to which SCI sends the results of an operation if no other file or device is specified as the destination. Entering a pathname or synonym in response to the prompt indicates COBOL system error messages are printed in a user file or on a device in lieu of the TLF.

If a file name is specified in response to the MESSAGE ACCESS NAME, control returns to the main SCI menu upon completion of the execution of the COBOL program.

However, if two tasks use the same file name for MESSAGE ACCESS NAME, the first task executed opens the file exclusively. The second task abnormally terminates with a TLF error. Refer to Appendix C for a listing of run-time error messages.

SWITCHES — Enter the setting of the software switches to be used by the program. The values should be 0 or 1 for each of the eight switches. Setting a value of 1 gives the switch a status of ON; 0 sets the status to OFF. Example switches are 10010011. The first, fourth, seventh, and eighth switches are ON, while the remaining are OFF. The initial value is 00000000. Refer to Figure 6-1 for an example of the use of software switches in the SPECIAL-NAMES paragraph of a COBOL source program module.

```
LINE    DEBUG    PG/LN    A...B.....................................

  25
  26                     SPECIAL-NAMES.
  27                         SWITCH-1.
  28                              ON STATUS IS SW-1-ON,
  29                              OFF STATUS IS SW-1-OFF;
  30                         SWITCH-5.
  31                              OFF STATUS IS SW-5-OFF,
  32                              ON STATUS IS SW-5-ON;
  33                         SWITCH-7.
  34                              ON IS SW-7-ON;
  35                         SWITCH-8.
  36                              OFF IS SW-8-OFF;
```

Figure 6-1. SPECIAL-NAMES Paragraph Example

FUNCTION KEYS — Enter YES to enable a function key to terminate input and allow function key codes to be returned through the ON EXCEPTION clause (if specified). This prompt pertains to all VDTs to which ACCEPT operations are performed. The initial value is NO (function keys are ignored).

The input text for an ACCEPT operation will be right-justified if all the following conditions are true:

- The ACCEPT operation is performed on a right-justified field.
- The initial value of the function keys is NO.

The input text for an ACCEPT operation will also be right-justified if all of the following conditions are true:

- The ACCEPT operation is performed on a right-justified field.
- The initial value of the function keys is YES.
- An non-blank prompt value is given.

The input text for an ACCEPT operation will be right-justified with zero-fill if all of the following conditions are true:

- The ACCEPT operation is performed on a numeric field.
- The initial value of the function keys is NO.

The input text for an ACCEPT operation will also be right-justified with zero-fill if all of the following conditions are true:

- The ACCEPT operation is performed on a numeric field.
- The initial value of the function keys is YES.
- A prompt is given with no operand.

The input text for an ACCEPT operation will be left-justified with blank-fill if all of the following conditions are true:

- The ACCEPT operation is performed on a numeric field.
- The initial value of the function keys is YES.
- No prompt is given.

6.2.2 Execute COBOL Program in Background (XCP)

Execution of an object module or a linked object module is performed in background with the XCP command. The following prompts appear with the indicated initial values:

```
EXECUTE COBOL PROGRAM <VERSION: L.R.V YYDDD>
       OBJECT ACCESS NAME: pathname@
      MESSAGE ACCESS NAME: [pathname@]
                 SWITCHES: [(integer)]     (00000000)
            FUNCTION KEYS: {YES/NO}        (NO)
```

The parameters are the same as those described for the XCPF command except for the absence of the DEBUG MODE prompt. Debugging is not allowed in background mode.

6.3 EXECUTION COMPLETION CODES AND RUN-TIME ERROR MESSAGES

Execution of a COBOL program through a command procedure causes a condition code to be returned under the synonym $$CC. The possible values of $$CC are as follows:

Value	Meaning
0000	Normal termination
8000	Abnormal termination

Any code set by the user through a STOP literal statement is set in the two rightmost positions of the condition code, as in the following examples:

Value	Meaning
0020	Implies a normal completion with a user code of >20. (An angle bracket preceding a number indicates a hexadecimal value.)
8030	Implies abnormal completion after the user code is set at >30.

The synonym $$CC should be checked in batch streams immediately after program execution. $$CC is used by other processors, and its integrity is not guaranteed after completion of the batch stream or the execution of another command.

Run-time error messages are provided for errors related to object code resulting from incorrect source statements or for system errors. Appendix C contains a listing of these error messages.

6.4 PROGRAM IMAGE EXECUTION

All COBOL programs installed as program images (tasks) on program files must have been linked to the COBOL run-time interpreter module by the link editor. The COBOL run-time interpreter is described in Section 5. At execution time, the operating system task loader loads the user task segments and any associated procedure segments into the task memory space. The operating system expands the task memory space as necessary. Multiple tasks sharing the same procedure segment need only one copy of the procedure segment in memory. This applies whether the procedure segment is on a system program file or in a user program file. When tasks from different program files are executed concurrently, each individual task segment and its associated procedure segment are loaded into memory at execution time.

6.4.1 Execute COBOL Task in Foreground (XCTF)

The XCTF command executes a COBOL task in foreground. The task must have been previously installed on a program file. The following prompts appear with the indicated initial values:

```
EXECUTE COBOL TASK FOREGROUND <VERSION: L.R.V YYDDD>
        PROGRAM FILE LUNO: integer
          TASK ID OR NAME: integer
               DEBUG MODE: {YES/NO}          (NO)
      MESSAGE ACCESS NAME: [(pathname)]
                 SWITCHES: [(integer)]       (00000000)
            FUNCTION KEYS: {YES/NO}          (NO)
```

PROGRAM FILE LUNO — Enter the LUNO (global or station) assigned to the program file on which the task is installed. The LUNO must have been assigned previously with the Assign LUNO (AL) command or the Assign Global LUNO (AGL) command.

The AL command assigns a task-local LUNO to a device or file accessible to the task for I/O operations. The AGL command assigns a LUNO to a device or file that is available to more than one job. For the AL and AGL commands, if you do not specify a LUNO, the system will assign one that is available. If you specify a LUNO to which the device or file is currently assigned, an error is returned.

TASK ID OR NAME — Enter the installed task ID or task name specified in the link control file.

DEBUG MODE — Enter YES if the task is to be executed in the COBOL debug mode. If a YES is entered, the task must have been linked using the run-time task entry module with the COBOL debugger (.S$SYSLIB.RCBTSKD); otherwise, an execution error is generated. The initial value is NO. Debug operations are defined in Section 7.

MESSAGE ACCESS NAME — No response to this prompt indicates that COBOL system error messages are to be printed to the terminal local file (TLF) of the initiating terminal. Entering a pathname or synonym causes COBOL system error messages to be written to this user file instead of the TLF, and control will return to the main SCI menu at completion of the COBOL program.

Users must note that if two tasks use the same file name for MESSAGE ACCESS NAME, the first task executed opens the file exclusively. A subsequent task abnormally terminates with a message access error.

SWITCHES — Enter the setting of software switches to be used by the program. The value should be 0 or 1 for each of the eight switches. Setting a value of one gives the switch a status of ON; 0 sets the status to OFF. Example switches are 10010011. The first, fourth, seventh, and eighth switches are ON, while the remaining are OFF. The initial value is 00000000. Refer to Figure 6-1 for an example of the use of software switches in the SPECIAL-NAMES paragraph of a COBOL source module.

FUNCTION KEYS — Enter YES to enable a function key to terminate input. The function key code will be returned through the ON EXCEPTION clause (if specified). This prompt pertains to all VDTs to which ACCEPT operations are performed. The initial value is NO (function keys are ignored).

The input text for an ACCEPT operation will be right-justified if all the following conditions are true:

- The ACCEPT operation is performed on a right-justified field.
- The initial value of the function keys is NO.

The input text for an ACCEPT operation will also be right-justified if all of the following conditions are true:

- The ACCEPT operation is performed on a right-justified field.
- The initial value of the function keys is YES.
- An non-blank prompt value is given.

The input text for an ACCEPT operation will be right-justified with zero fill if all of the following conditions are true:

- The ACCEPT operation is performed on a numeric field.
- The initial value of the function keys is NO.

The input text for an ACCEPT operation will also be right-justified with zero fill if all of the following conditions are true:

- The ACCEPT operation is performed on a numeric field.
- The initial value of the function keys is YES.
- A prompt is given with no operand.

The input text for an ACCEPT operation will be left-justified with blank fill if all of the following conditions are true:

- The ACCEPT operation is performed on a numeric field.
- The initial value of the function keys is YES.
- No prompt is given.

6.4.2 Execute COBOL Task in Background (XCT)

The XCT command executes a COBOL task in background mode. The task must have been previously installed on a program file. The following prompts appear with the indicated initial values:

```
EXECUTE COBOL TASK <VERSION: L.R.V YYDDD>
        PROGRAM FILE LUNO: integer
          TASK ID OR NAME: integer
      MESSAGE ACCESS NAME: [pathname@]
                  SWITCHES: [(integer)]       (00000000)
            FUNCTION KEYS: {YES/NO}           (NO)
```

The responses to the XCT prompts are the same as those described for the XCTF command except for the absence of the DEBUG MODE prompt. Debugging is not allowed in background mode; therefore, the task must have been linked using the run-time interpreter module without the COBOL debugger (.S$SYSLIB.RCBTSK).

6.5 EXECUTION COMPLETION CODES AND RUN-TIME ERROR MESSAGES

The execution completion codes described previously for the XCPF and XCP commands are the same as for the XCTF and XCT commands. Run-time error messages are described in Appendix C.

6.6 PROGRAM TERMINATION MESSAGES

The COBOL run-time termination messages, STOP RUN AT . . . and END COBOL RUN are not displayed to the message file when a program terminates normally. This enhances performance because Assign LUNO, Open File, Write, and Close operations to the COBOL message file are avoided. If the messages are desired, they can be produced by changing parameter 6 in the .BID or .QBID of the XCTF, XCT, XCP, and XCPF SCI commands. The parameter value should be changed from N to Y to achieve this. That is, the statement

```
EXAMPLE:

   PARMS = ("@@$XCP$O",&DEBUG MODE,@&MESSAGE ACCESS NAME,
   "&SWITCHES",&FUNCTION KEYS,"N",,)
```

should be changed to

```
   PARMS = ("@@$XCP$O",&DEBUG MODE,@&MESSAGE ACCESS NAME,
   "&SWITCHES",&FUNCTION KEYS,"Y",,)
```

7

Debugging

7.1 DEBUG MODE

Debug mode allows you to perform the following functions:

- Specify address stops, single statement execution, or data item dumps
- Eliminate or change address stops
- Modify selected data items
- Designate the next address in the program to be executed
- Write the contents of the screen to the message file
- Exit from the debug mode
- Quit execution of a task

7.2 DEBUGGING A COBOL MODULE

The debugger is designed specifically for the COBOL run-time interpreter. At any of the following times, the debugger assumes control of the video display terminal (VDT) from which the COBOL program is executed:

- Before program execution
- When an address stop is encountered
- When a STOP RUN statement or an untrapped error condition causes the program to terminate

When the debugger is in control of the VDT, it responds to the debug commands described later in this section.

7.2.1 Activating the Debugger

Note that the COBOL debugger runs only in foreground mode. For this reason, use the Execute COBOL Program Foreground (XCPF) or the Execute COBOL Task Foreground (XCTF) SCI command to activate the debugger. Before using XCTF to activate the debugger, you must first link the task using the run-time interpreter task entry module with the COBOL debugger (.S$SYSLIB.RCBTSKD). However, when you use XCPF to activate the debugger, this step is not necessary. The XCPF automatically bids one of two prelinked tasks (either with or without the COBOL debugger), depending on your response to the DEBUG MODE prompt.

To activate the debugger, enter YES after the DEBUG MODE prompt. The debugger responds by displaying the following information on your screen:

```
ADDRESS STOP: <currently active address stops>
mmmmmmxxyyyy D?
```

where:

ADDRESS STOP lists the currently active address stops; a maximum of four address stops can be assigned.

mmmmmm names the module currently being executed.

xxyyyy are hexadecimal digits that specify the address of the next COBOL source statement to be executed. If the source statement address is in a segmented COBOL module, specify the segment number in the first two digits (xx) and the statement address in the next four digits (yyyy). Omit the segment number when the source statement is not in a segmented COBOL module.

D? indicates that you can now enter debug commands.

EXAMPLE 1

```
ADDRESS STOP: SEG    0100, SEG    050300, SUBONE 0050, SUBTWO 1C
SGMEN  010000 D?
```

This example lists the four currently active address stops (SEG 0100, SEG 050300, SUBONE 0050, and SUBTWO 1C). The name of the module that is currently executing is SGMEN. The source statement to be executed next is located at address 010000. Note that this six-digit number indicates that module SGMEN is a segmented COBOL module. The segment number is 01 and the source statement is at address 0000.

EXAMPLE 2

```
ADDRESS STOP: SEG    0100
MAIN 0000 D?
```

This example lists only one currently active address stop (SEG 0100). The name of the module currently executing is MAIN. The source statement to be executed next is located at address 0000. Note that this address is only four digits long, indicating that module MAIN is not located in a segmented COBOL module. The source statement is located at address 0000 in the main program.

EXAMPLE 3

```
ADDRESS STOP: SEG 0100, SEG 050300, SUBONE 0050
GRAPHI 0040 D?
```

This example lists three active address stops (SEG 0100, SEG 050300, and SUBONE 0050). The name of the module currently executing is GRAPHI. The source statement to be executed next is at the address 0040. Note that this address is only four digits long, indicating that module GRAPHI is not located in a segmented COBOL module. The source statement is at the address 0040 in the main program.

7.2.2 COBOL Debug Commands

COBOL debug commands consist of a single letter followed by a string of hexadecimal fields separated by commas. The total length of a debug command cannot exceed 20 characters. In the command formats that follow, brackets indicate optional arguments. Blanks terminate the scan. After executing a valid command, the debugger requests another command by displaying the following prompt:

```
D?
```

If the debugger encounters an error in decoding a command, one of the following messages appears:

Error Code	Explanation
C?	The command is unrecognizable.
S?	A syntax error occurred in the operands.
V?	The value of the operand(s) is out of range.

Debug supports ten commands. Table 7-1 lists these commands, and the following paragraphs explain the commands.

Table 7-1. Debug Commands

Command	Name
A	Assign Address Stop
D	Dump Data Item
E	Exit Debug Mode
L	Change Program Location
M	Modify Data Item
Q	Quit Execution
R	Resume Program Execution
S	Execute Next Single Statement
U	Undo Address Stop
W	Write Screen to Message File

NOTE

Any COBOL run-time errors return control to the debugger after writing the error message to the message file. This allows inspection of data items.

Figure 7-1 contains a compiler output listing for a COBOL program. The paragraphs that follow refer to sections of this figure.

```
 DXCBL          L.R.V YY.DDD COMPILED:MM/DD/YY HH:MM:SS  OPT=M      PAGE    1

SOURCE ACCESS NAME:     MANUAL.PG.SRC.FIG0701
OBJECT ACCESS NAME:     DUMY
LISTING ACCESS NAME:    MANUAL.PG.LST.FIG0701
OPTIONS:                M
PRINT WIDTH:            80
PAGE SIZE:              55
PROGRAM SIZE (LINES):   1000


 DXCBL          L.R.V YY.DDD COMPILED:MM/DD/YY HH:MM:SS  OPT=M     PAGE    2
LINE  DEBUG PG/LN   A...B.................................................
   1                IDENTIFICATION DIVISION.
   2                PROGRAM-ID. DATA-TYPES.
   3                ENVIRONMENT DIVISION.
   4                CONFIGURATION SECTION.
   5                SOURCE-COMPUTER. TI-990.
   6                OBJECT-COMPUTER. TI-990.
   7                DATA DIVISION.
   8                WORKING-STORAGE SECTION.
   9
  10               ****  ALPHANUMERIC
  11                01   ANS      PIC X(20) VALUE  "CORRECT RESULT: 330".
  12
  13               ****  ALPHABETIC
  14                01   ABS      PIC A(20) VALUE  "COMPUTED RESULT:".
  15
  16               ****  DISPLAY SIGNED LEADING
  17                01   NL       PIC S9(6) SIGN LEADING VALUE +45.
  18
  19               ****  DISPLAY SIGNED LEADING SEPARATE
  20                01   NLS      PIC S9(6) SIGN LEADING SEPARATE VALUE  55.
  21
  22               ****  DISPLAY SIGNED TRAILING
  23                01   NT       PIC S9(6) SIGN TRAILING VALUE  50.
  24
  25               ****  NUMERIC DISPLAY SIGNED (TRAILING SEPARATE)
  26                01   NSS      PIC S9(6) VALUE  30.
  27
  28               ****  NUMERIC DISPLAY UNSIGNED
  29                01   NSU      PIC 9(6) VALUE  25.
  30
  31               ****  COMPUTATIONAL UNSIGNED
  32                01   NCU      PIC 9(5)  COMP   VALUE 15.
  33
  34               ****  COMPUTATIONAL SIGNED
  35                01   NCS      PIC S9(5) COMP   VALUE 20.
  36
```

Figure 7-1. Compiler Output Listing (Sheet 1 of 3)

```
 37               ****  BINARY SIGNED OR UNSIGNED (COMPUTATIONAL-1)
 38                01   NBS      PIC S9(5) COMP-1 VALUE 5.
 39
 40               ****  NUMERIC PACKED DECIMAL (COMPUTATIONAL-3)
 41                01   NPS      PIC S9(5) COMP-3 VALUE +10.
 42
 43               ****  MULTI-WORD BINARY UNSIGNED (COMPUTATIONAL-4)
 44                01   NMB      PIC 99    COMP-4 VALUE 35.
 45
 46               ****  MULTI-WORD BINARY SIGNED (COMPUTATIONAL-4)
 47                01   NMS      PIC S9(4) COMP-4 VALUE 40.
 48
 49               ****  NUMERIC EDITED
 50                01   NSE      PIC ZZ9.
 51
 52               ****  ALPHANUMERIC EDITED
 53                01   ANSE     PIC XX/XX/XX.
 54

 DXCBL          L.R.V YY.DDD COMPILED:MM/DD/YY HH:MM:SS  OPT=M      PAGE    3
LINE  DEBUG PG/LN   A...B.....................................................
 55               ****  GROUP
 56                01   GRP.
 57                     02 YR    PIC XX.
 58                     02 MO    PIC XX.
 59                     02 DA    PIC XX.
 60
 61                PROCEDURE DIVISION.
 62  >0000         BEGIN.
 63  >0000              ACCEPT GRP FROM DATE.
 64  >0004              MOVE GRP TO ANSE.
 65  >0008              DISPLAY ANSE LINE 1 ERASE.
 66  >0010              COMPUTE NSE = NBS + NPS + NCU + NCS + NSU +
 67                        NSS + NMB + NMS + NL + NT + NLS.
 68  >0028              DISPLAY ANS LINE 2.
 69  >002E              DISPLAY ABS LINE 3.
 70  >0034              DISPLAY NSE LINE 3 POSITION 18.
 71  >003C              ACCEPT YR.
 72  >0040              STOP RUN.
 73         ZZZZZZ END PROGRAM.                              *** END OF FILE


 DXCBL          L.R.V YY.DDD COMPILED:MM/DD/YY HH:MM:SS   OPT=M      PAGE    4
 ADDRESS  SIZE  DEBUG ORDER TYPE                  NAME

  >002A     20   ANS    0   ALPHANUMERIC         ANS

  >003E     20   ABS    0   ALPHABETIC           ABS

  >0052      6   NL     0   NUM LEAD SIGNED      NL

  >0058      7   NLS    0   NUM SEP LEAD SIGNED  NLS
```

Figure 7-1. Compiler Output Listing (Sheet 2 of 3)

```
>0060      6   NT     0   NUM TRAIL SIGNED      NT

>0066      7   NSS    0   NUM SEP LEAD SIGNED   NSS

>006E      6   NSU    0   NUMERIC UNSIGNED      NSU

>0074      5   NCU    0   COMP UNSIGNED         NCU

>007A      6   NCS    0   COMP SIGNED           NCS

>0080      2   NBS    0   BINARY SIGNED         NBS

>0082      3   NPS    0   PACKED SIGNED         NPS

>0086      1   NMB    0   MULTI BINARY          NMB

>0088      2   NMS    0   MULTI BINARY SIGNED   NMS

>008A      3   NSE    0   NUMERIC EDITED        NSE

>008E      8   ANSE   0   ALPHANUMERIC EDITED   ANSE

>0096      6   GRP    0   GROUP                 GRP
>0096      2   ANS    0   ALPHANUMERIC           YR
>0098      2   ANS    0   ALPHANUMERIC           MO
>009A      2   ANS    0   ALPHANUMERIC           DA


READ ONLY BYTE SIZE =       >00F8

READ/WRITE BYTE SIZE =      >009E

OVERLAY SEGMENT BYTE SIZE = >0000

TOTAL BYTE SIZE =           >0196

    0 ERRORS

    0 WARNINGS
```

Figure 7-1. Compiler Output Listing (Sheet 3 of 3)

7.2.2.1 Assign Address Stop Command (A). The A command indicates the address at which normal execution of the program stops and the debugger assumes control. This address stop is cleared whenever a breakpoint is set, but you can assign up to four address stops. After you assign address stops, use the Resume command (R) to resume execution of the program. You can use the Undo Address Stop (U) command to eliminate any address stops you assigned with the A command. The A command is as follows:

```
A[hh]hhhh[,PROGID]
```

where:

hh is an optional hexadecimal number signifying the segment number in PROGID at which execution stops. When the segment number is omitted, it is assumed to be zero.

hhhh represents up to four hexadecimal digits specifying the address in program PROGID at which execution stops. Leading zeros of the address may be omitted unless the segment number is specified. The address is the address printed in the DEBUG column of the compiler listing. (Refer to the data maps in Figure 7-1.)

PROGID is an optional operand that names the program unit in which the stop is located. If omitted, the stop is assumed to be located in the program unit that is currently executing.

The following are examples of the A command:

Example	Meaning
A0202	Assign an address stop in the current COBOL module at address 0202.
A012345,SUB1	Assign an address stop at location 2345 of segment 01 in program SUB1.

NOTE

If a stop address is reached during execution, that address is eliminated from the list of stop addresses. If you wish to stop at a given address in a loop each time it is executed, you must assign that address each time to program stops before restarting execution.

7.2.2.2 Dump Data Item Command (D). This command displays memory at a specified address in the format of a specified data type. The D command has two possible formats.

D Command: First Format. The first format of the D command displays data items from the DATA DIVISION:

```
Dhhhh,dd[,TYPE]
```

where:

hhhh gives the hexadecimal address printed in the ADDRESS column of the compiler listing that contains the data maps. (Refer to the ADDRESS column of the data maps in Figure 7-1.)

dd is a decimal number indicating the number of characters to be displayed. This information also appears in the listing. (Refer to the SIZE column of the data maps in Figure 7-1.)

TYPE consists of two, three, or four letters that abbreviate the data type. (Refer to the DEBUG column of the data maps in Figure 7-1.)

Both the data type and its valid abbreviation are shown for each data item in the listing. The valid abbreviations are as follows:

Abbreviation	Description
ABS	Alphabetic
ANS	Alphanumeric
ANSE	Alphanumeric edited
GRP	Group
HEX	Default
NBS	Binary signed (COMP-1)
NCS	Computational signed (COMP)
NCU	Computational unsigned (COMP)
NL	Numeric leading signed
NLS	Numeric leading separate signed
NMB	Multiword binary (COMP-4)
NMS	Multiword binary signed (COMP-4)
NPS	Numeric packed signed (COMP-3)
NSE	Numeric edited
NSS	Numeric display signed
NSU	Numeric display unsigned (DISP)
NT	Numeric trailing signed

The following is an example of the first format of the D command:

Example	Meaning
D096,6,GRP	Display the contents of a six-character data item from the data division at address 096.
D06E,6,NSU	Display the six-character numeric field from the data division at address 06E.

D Command: Second Format. The second format of the D command displays data items from the LINKAGE SECTION of a separately compiled program:

```
DLdd[+hhhh],dd[,TYPE]
```

where:

L indicates that a data item from the LINKAGE SECTION of a separately compiled program is to be displayed.

dd is an ordinal number specifying which data item of the linkage data items is to be displayed.

+ hhhh is a hexadecimal offset from the starting address of the data item. You can omit this.

dd is a decimal number indicating the number of characters to be displayed. This information also appears in the listing. (Refer to the SIZE column of the data maps in Figure 7-1.)

TYPE is a two-, three-, or four-letter abbreviation specifying the data type. When you do not enter a type, the display of the data items is in hexadecimal.

The following is an example of the second format of the D command:

Example	Meaning
DL1,5,HEX	Display five characters of the first linkage item in hexadecimal.
DL3 + F0,16,ANS	Display sixteen alphanumeric characters of the third linkage item starting at offset F0.

7.2.2.3 Exit Debug Mode Command (E). This command discontinues execution of the current user module under control of the debugger. The program continues to execute, but in normal mode. The format of the command is as follows:

```
E
```

No operand is required.

7.2.2.4 Change Program Location Command (L). The L command designates the next address in the program to be executed. The format of the command is as follows:

```
L[xx]yyyy
```

where:

xx is an optional hexadecimal digit signifying the segment number. The default is zero.

NOTE

If you are currently executing in the fixed segment, you can only change location within the fixed segment. If you are currently executing in an independent segment, you can change location within either that segment or the fixed segment.

yyyy is a hexadecimal digit showing the address in the current program module where execution begins. This address is printed in the DEBUG column of the compiler listing.

NOTE

You must enter four hexadecimal digits for the address when you specify the segment number. You can omit the leading zeros of the address when you do not specify the segment number. Unpredictable results occur if the value you give is not the beginning of a statement from the DEBUG column. (Refer to the data maps in Figure 7-1.)

The following are examples of the L command:

Example	Meaning
L0404	Execute the current COBOL module starting at address 0404.
L	Execute the current COBOL module at the beginning of the current program module.

NOTE

The L command has no effect when you enter the debugger after normal program termination.

7.2.2.5 Modify Data Item Command (M). The M command is used to overwrite the existing contents of items in the DATA DIVISION of a program or in the LINKAGE SECTION of a separately compiled program. Modifications can consist of either ASCII strings or hexadecimal digits. The command has six possible formats.

M Command: First Format. The first format for the M command is as follows:

```
Mhhhh,>h[,h,...,h]
```

where:

hhhh is the hexadecimal address printed in the ADDRESS column of the compiler listing that contains the data maps.

>h[,h,...,h] indicates a hexadecimal modification. One or more one- or two-digit hexadecimal numbers can follow the right angle bracket (>). Each number is placed into one byte of storage. Any numbers to the right of the first one are placed at memory locations whose addresses are successively greater than the initial hhhh hexadecimal address.

The following is an example of format one:

Example	Meaning
M1237,>FF,20,D,1	Place >FF at location 1237, >20 at location 1238, >0D at location 1239, >01 at location 123A.

M Command: Second Format. The second format for the M command is as follows:

```
Mhhhh,"string"
```

where:

hhhh is the hexadecimal address printed in the ADDRESS column of the compiler listing that contains the data maps.

"string" is the ASCII string to be placed at that hexadecimal address.

NOTE

To print the " character within a string, you must enter that character twice. For example, STR""ING will yield STR"ING.

The following is an example of format two:

Example	Meaning
M1FF0,"TEXAS"	Place the string TEXAS starting at memory location >1FF0.

M Command: Third Format. The third format for the M command is as follows:

```
Mhhhh,"string",dd
```

where:

hhhh is the hexadecimal address, printed in the ADDRESS column of the compiler listing that contains the data maps.

"string" is the ASCII string to be placed at that hexadecimal address.

dd is an optional decimal number indicating the total length of the field to be modified. Operand dd must be equal to or greater than the number of characters in the ASCII string. If operand dd is greater than the number of characters in the string, the difference between them indicates the number of blanks which are appended to the right of the string and written to memory.

The following is an example of format three:

Example	Meaning
M100F,"RIGHT PAD",40	Place the string RIGHT PAD, followed by 31 blanks, starting at memory location 100F.

M Command: Fourth Format. The fourth format for the M command is as follows:

```
MLdd[+hhhh],>h[,h...h]
```

where:

L indicates that a data item from the LINKAGE SECTION of a separately compiled program is to be modified.

WARNING

Users can accidentally modify data outside the LINKAGE SECTION. Addresses for linkage items are not verified. It is the user's responsibility to enter correct addresses.

dd is a decimal ordinal number specifying which data item of the linkage data items is to be modified.

+ hhhh is an optional hexadecimal offset from the starting address of the data item.

>h[,h...h] indicates a hexadecimal modification. One or more one- or two-digit hexadecimal numbers can follow the right angle bracket (>). Each number is placed into one byte of storage. Any numbers to the right of the first one are placed at memory locations whose addresses are successively greater than those of the initial data item.

The following is an example of format four:

Example	Meaning
ML4 + F0,>1,D,20,FF	Place >01, >0D, >20, and >FF starting at the address of linkage item four plus >00F0.

M Command: Fifth Format. The fifth format for the M command is as follows:

```
MLdd[+hhhh],"string"
```

where:

L indicates that a data item from the LINKAGE SECTION of a separately compiled program is to be modified.

WARNING

Users can accidentally modify data outside the LINKAGE SECTION. Addresses for linkage items are not verified. It is the user's responsibility to enter correct addresses.

dd is a decimal ordinal number specifying which data item of the linkage data items is to be modified.

+ hhhh is an optional hexadecimal offset from the starting address of the data item.

"string" is the ASCII string to be placed at that hexadecimal address.

The following is an example of format five:

Example	Meaning
ML3,"INSTRUMENTS"	Place the string INSTRUMENTS starting at the address of the third linkage item.

M Command: Sixth Format. The sixth format for the M command is:

```
MLdd[+hhhh],"string",dd
```

where:

L indicates that a data item from the LINKAGE SECTION of a separately compiled program is to be modified.

WARNING

Users can accidentally modify data outside the LINKAGE SECTION. Addresses for linkage items are not verified. It is the user's responsibility to enter correct addresses.

dd is a decimal ordinal number specifying which data item of the linkage data items is to be modified.

+ hhhh is an optional hexadecimal offset from the starting address of the data item.

"string" is the ASCII string to be placed at that hexadecimal address.

dd is an optional decimal number indicating the total length of the field to be modified. It must be equal to or greater than the number of characters in the ASCII string. If it is greater, the difference indicates the number of blanks that are appended to the right of the string and written to memory.

The following is an example of format six:

Example	Meaning
ML5,"LONG STRING",45	Place the string LONG STRING, followed by 34 blanks, at the address of the fifth linkage item.

7.2.2.6 Quit Execution Command (Q). The Q command terminates the current user program under control of the debugger and returns control to SCI. The format of the command is as follows:

```
Q
```

No operand is required.

7.2.2.7 Resume Program Execution Command (R). The R command resumes program execution after you assign all address stops. The format of the command is as follows:

```
R
```

No operand is required.

7.2.2.8 Execute Next Single Statement Command (S). The S command executes one COBOL statement and returns control to the debugger. The format of the command is as follows:

```
S
```

No operand is required.

NOTE

The S command has no effect when the debugger is entered after normal program termination.

7.2.2.9 Undo Address Stop Command (U). The U command eliminates address stops you assigned with the A command. The format for the U command is as follows:

```
U[hh]hhhh[,progid]
```

where:

hh is an optional hexadecimal number signifying the segment number in progid at which the address stop was assigned. When the segment number is omitted, it is assumed to be zero.

hhhh is four or less hexadecimal digits signifying the address in program progid at which an address stop was assigned. This address is the address printed in the DEBUG column of the compiler listing. (Refer to the data maps in Figure 7-1.) When you omit the segment number, you can omit leading zeros of the address. When you specify a segment number, you must enter four hexadecimal digits for the address.

Progid is an optional operand that names the program unit in which the address stop was assigned. If you omit this operand, the address stop is assumed to be located in the program that is currently executing.

The following are examples of the U command:

Example	Meaning
U0202	Remove an address stop in the current module at address 0202.
U012345,SUB1	Remove an address stop at location 2345 of segment 01 in program SUB1.

7.2.2.10 Write Screen to Message File Command (W). This command writes the contents of the screen to the device or file you specified in response to the prompt MESSAGE ACCESS NAME that appeared when you executed the COBOL program or task. The format for the W command is as follows:

```
W
```

No operand is required.

7.3 DEBUGGING OF ASSEMBLY LANGUAGE SUBROUTINES LINKED TO COBOL PROGRAMS

You can use an interactive symbolic debugging program to debug assembly language program object modules linked to COBOL program object modules as subroutines. The interactive debugger is provided as an operating system utility; it is not the COBOL debugger. The interactive debugger operates from either an interactive VDT or an interactive hard-copy terminal.

The debugger allows you to display and modify central processing unit (CPU) registers, workspace registers, and memory. It also allows controlled execution of a task.

In the run mode, you can halt and resume. You can also set new breakpoints to halt the task. In the simulation mode, the system analyzes the execution between each instruction. You can specify trap conditions that interrogate the program counter (PC) or you can specify memory content. Breakpoints designed to halt task execution can be conditional on a given number of accesses within a specified range of PC values, memory locations, or communications register unit (CRU) addresses. You can set breakpoints at given status register (SR) values or supervisor calls (SVCs).

NOTE

You can use this method of debugging an assembly language module only with a linked program image using the Execute COBOL Task Debug (XCTD) command.

Figure 7-2 through Figure 7-5 are examples of debugging interactively. Figure 7-2 shows how the interactive debugger operates under user control.

WARNING

Because of the way the system debugger is executed, there is a possible conflict between it and a COBOL program. If the COBOL program has been executed with the function keys enabled and the System Debugger is then executed, the function keys will be disabled. This problem is most common when the COBOL program executes an ACCEPT/DISPLAY command while still in the System Debugger. However, this problem may also occur at any time.

```
[]AGL
ASSIGN GLOBAL LUNO
               LUNO:
       ACCESS NAME:  .PROFILE
    PROGRAM FILE ?:  YES
ASSIGNED LUNO = >4
[]XCTD
EXECUTE COBOL TASK DEBUG <VERSION: L.R.V. YYDDD>
              PROGRAM FILE:  >4
           TASK ID OR NAME:  1
       MESSAGE ACCESS NAME:
                  SWITCHES:  00000000
             FUNCTION KEYS:  NO
[]AB
ASSIGN BREAKPOINTS
              RUN ID:  >DC
        ADDRESS(ES):  04E6C+04E
[]RT
RESUME TASK
           RUN ID: >DC
[]SP
SHOW PANEL
               RUN ID:  >DC
       MEMORY ADDRESS:
[]LM
LIST MEMORY
                 RUN ID:  >DC
       STARTING ADDRESS:  #R5
        NUMBER OF BYTES:  #R6
    LISTING ACCESS NAME:
[]DPB
DELETE AND PROCEED FROM BREAKPOINT
                   RUN ID:  >DC
  DESTINATION ADDRESS(ES):
[]WAIT
```

Figure 7-2. Interactive Debugging Example

A debugging session includes these steps:

1. Assign a global LUNO to the program file containing the task to be executed. In Figure 7-2, LUNO >4 is assigned to the program file .PROGFILE.

2. Execute the XCTD command to bid the task in suspended state as shown in Figure 7-2. SCI assigns a task ID that is used as the initial value for the commands issued after this step. Figure 7-2 uses >DC. The workspace registers and memory locations at the beginning of the task appear on the VDT screen.

3. Examine the link edit listing to obtain the origin address of the assembly module to be debugged. Figure 7-2 uses address 04E6C.

4. Assign a breakpoint to a particular instruction address of the assembly language module. This address will be the assembly module origin address (from the link edit listing) plus a displacement of the instruction within the module. (Refer to Figure 7-3.) The task executes until it encounters the breakpoint address. Figure 7-2 uses 04E6C plus a displacement of 04E.

NOTE

Overlay phases must be in memory before you can assign breakpoint addresses within them. Therefore, you should link the modules to be overlayed into the root phase (phase 0) of the task for debugging purposes.

5. Begin execution of the task with the Resume Task (RT) command.

6. Issue a Show Panel (SP) command periodically to determine if execution has reached the breakpoint address. If so, the task status appears on the VDT screen as STATE = 06(BP).

7. Perform other operations such as List Memory (LM), Modify Workspace Register (MWR), Modify Internal Registers (MIR), and Modify Memory (MM) while at this breakpoint.

8. Remove the breakpoint and resume the task with the Delete and Proceed from Breakpoint (DPB) command. If you set another breakpoint, the task executes to the next breakpoint; otherwise, the task executes to completion.

9. Wait for execution to complete with a Wait (WAIT) command.

Figure 7-3 is the COBOL module that calls the assembly language modules. Figure 7-4 and Figure 7-5 are examples of assembly language modules.

```
 DXCBL          L.R.V YY.DDD COMPILED:MM/DD/YY HH:MM:SS  OPT=       PAGE    1

SOURCE ACCESS NAME:     MANUAL.DN.SRC.COBOL
OBJECT ACCESS NAME:     MANUAL.DN.OBJ.COBOL
LISTING ACCESS NAME:    MANUAL.DN.LST.COBOL
OPTIONS:
PRINT WIDTH:            80
PAGE SIZE:              55
PROGRAM SIZE (LINES):   1000
```

Figure 7-3. COBOL Program Calling Assembly Language Modules (Sheet 1 of 2)

```
 DXCBL          L.R.V YY.DDD COMPILED:MM/DD/YY HH:MM:SS  OPT=       PAGE    2
LINE  DEBUG PG/LN   A...B.................................................
   1                IDENTIFICATION DIVISION.
   2                PROGRAM-ID. GETDATE.
   3                ENVIRONMENT DIVISION.
   4                CONFIGURATION SECTION.
   5                SOURCE-COMPUTER. TI990.
   6                OBJECT-COMPUTER. TI990.
   7                DATA DIVISION.
   8                WORKING-STORAGE SECTION.
   9                01   SVC-BLOCK.
  10               *                          SVC >03 RETRIEVES DATE AND TIME.
  11                02   CALL-CODE PIC 9 COMP-4 VALUE >03.
  12                02   ERROR-CODE PIC 9 COMP-4.
  13                02   TIME-BUFFER-ADDRESS PIC 999 COMP-4.
  14                01   TIME-BUFFER.
  15                02   YEAR PIC 999 COMP-4.
  16                02   JUL  PIC 999 COMP-4.
  17                02   HOUR PIC 999 COMP-4.
  18                02   MIN  PIC 999 COMP-4.
  19                02   SEC  PIC 999 COMP-4.
  20                01   FIELD PIC ZZZ9.
  21                PROCEDURE DIVISION.
  22  >0000         MAIN-SECTION.
  23  >0000              CALL "ADDRES" USING TIME-BUFFER, TIME-BUFFER-ADDRESS.
  24  >0002              MOVE >00 TO ERROR-CODE.
  25  >0006              CALL "IOCALL" USING SVC-BLOCK.
  26  >0008              IF ERROR-CODE NOT = >00 STOP RUN.
  27  >0010              MOVE YEAR TO FIELD.
  28  >0014              DISPLAY FIELD ERASE LINE 1.
  29  >001C              MOVE JUL TO FIELD.
  30  >0020              DISPLAY FIELD LINE 1 POS 6.
  31  >0028              MOVE HOUR TO FIELD.
  32  >002C              DISPLAY FIELD LINE 1 POS 11.
  33  >0034              MOVE MIN TO FIELD.
  34  >0038              DISPLAY FIELD LINE 1 POS 16.
  35  >0040              MOVE SEC TO FIELD.
  36  >0044              DISPLAY FIELD LINE 1 POS 21.
  37  >004C              ACCEPT FIELD LINE 24.
  38         ZZZZZZ END PROGRAM.                              *** END OF FILE

READ ONLY BYTE SIZE =        >0102

READ/WRITE BYTE SIZE =       >0044

OVERLAY SEGMENT SIZE =       >0000

TOTAL BYTE SIZE =            >0146

    0 ERRORS

    0 WARNINGS
```

Figure 7-3. COBOL Program Calling Assembly Language Modules (Sheet 2 of 2)

```
           SDSMAC L.R.V YY.DDD    HH:MM:SS   DAY, MMM DD, YYYY.
 NAMES TABLE                                                        PAGE 0001
SOURCE ACCESS NAME=      MANUAL.DN.SRC.ADDRES
OBJECT ACCESS NAME=      MANUAL.DN.OBJ.ADDRES
LISTING ACCESS NAME=     MANUAL.DN.LST.ADDRES
ERROR ACCESS NAME=
OPTIONS=       MACRO LIBRARY PATHNAME=
ADDRES     SDSMAC L.R.V YY.DDD    HH:MM:SS  DAY, MMM DD, YYYY.
RETRIEVE DATA ITEM ADDRESS                                          PAGE 0002
 0002                    IDT        'ADDRES'
 0003              *
 0004              ****************************************************************
 0005              * TITLE: ADDRES
 0006              * REVISION: MM/DD/YY ORIGINAL
 0007              * ABSTRACT: ADDRES IS CALLED TO RETURN THE RUNTIME ADDRESS
 0008              *           OF A DATA ITEM FOR USE BY THE IOCALL SUBROUTINE.
 0009              *
 0010              * CALLING SEQUENCE:
 0011              *         CALL "ADDRES" USING VARIABLE-NAME, VARIABLE-ADDRESS
 0012              *
 0013              ****************************************************************
 0014              *
 0015                     DEF  ADDRES
 0016 0000                DSEQ
 0017 0000 0004" ADDRES   DATA WSP1,START          TRANSFER VECTOR
      0002 0000'
 0018 0004        WSP1    BSS  32                  WORKSPACE
 0019 0024                DEND
 0020 0000                PSEG
 0021 0000 C01D   START   MOV  *R13,R0             PICK ARG LIST POINTER
 0022 0002 C070           MOV  *R0+,R1             GET ARGLIST BYTE COUNT
 0023 0004 0281           CI   R1,4                MUST BE 2 PARAMETERS
      0006 0004
 0024 0008 1603           JNE  RETURN              ELSE DO NOTHING
 0025 000A C0B0           MOV  *R0+,R2             R2<- VARIABLE-NAME ADDR.
 0026 000C C0D0           MOV  *R0,R3              R3<- VARIABLE-ADDR PTR.
 0027 000E C4C2           MOV  R2,*R3              MOVE IN THE ADDRESS
 0028 0010 0380   RETURN  RTWP                     RETURN TO CALLER
 0029 0012                PEND
 0030                     END
NO ERRORS,     NO WARNINGS
```

Figure 7-4. Assembly Language Module ADDRES

```
           SDSMAC L.R.V YY.DDD    HH:MM:SS DAY, MMM DD, YYYY.
 NAMES TABLE                                                    PAGE 0001
SOURCE ACCESS NAME=      MANUAL.DN.SRC.IOCALL
OBJECT ACCESS NAME=      MANUAL.DN.OBJ.IOCALL
LISTING ACCESS NAME=     MANUAL.DN.LST.IOCALL
ERROR ACCESS NAME=
OPTIONS=
MACRO LIBRARY PATHNAME=
IOCALL     SDSMAC L.R.V YY.DDD    HH:MM:SS DAY, MMM DD, YYYY.
ISSUE SUPERVISOR CALL                                           PAGE 0002
 0002                    IDT  'IOCALL'
 0003              *
 0004              ****************************************************************
 0005              * TITLE: IOCALL
 0006              * REVISION: MM/DD/YY
 0007              * ABSTRACT: IOCALL IS CALLED TO ISSUE AN OPERATING SYSTEM
 0008              *           SUPERVISOR CALL.
 0009              *
 0010              * CALLING SEQUENCE:
 0011              *           CALL "IOCALL" USING SVC-CONTROL-BLOCK.
 0012              *
 0013              ****************************************************************
 0014              *
 0015                    DEF  IOCALL
 0016 0000               DSEG
 0017 0000 0004" IOCALL DATA WSP1,START          TRANSFER VECTOR
      0002 0000'
 0018 0004       IOCALL BSS  32                  WORKSPACE
 0019 0024               DEND
 0020 0000               PSEG
 0021 0000 C01D  START  MOV  *R13,R0             PICK ARG LIST POINTER
 0022 0002 C070         MOV  *R0+,R1             GET ARGLIST BYTE COUNT
 0023 0004 0281         CI   R1,2                MUST BE 1 PARAMETER
      0006 0002
 0024 0008 1602         JNE  RETURN              ELSE DO NOTHING
 0025 000A C0B0         MOV  *R0+,R2             R2<- SVC-CALL-BLOCK PTR.
 0026 000C 2FD2         XOP  *R2,15              XOP15-> SVC-CALL-BLOCK
 0027 000E 0380  RETURN RTWP                     RETURN TO CALLER
 0028 0010              PEND
 0029                   END
NO ERRORS,     NO WARNINGS
```

Figure 7-5. Assembly Language Module IOCALL

8

Calling Subroutines

8.1 GENERAL

The CALL statement is used to call subroutines written in COBOL and other languages provided the linkage conventions are compatible. Refer to the *COBOL Reference Manual* for a detailed description of the CALL verb syntax.

8.2 COBOL SUBROUTINE LIBRARY PACKAGE

The COBOL Subroutine Library Package provides you with frequently used functions. Table 8-1 lists the subroutines; Appendix D lists the functions of the routines, calling sequences, descriptions of each required argument, and error codes generated within the subroutines.

All data fields used as parameters to the COBOL subroutines MUST be aligned on word boundaries. This can be accomplished by making the parameter an 01-Level data item in the WORKING STORAGE section of the program. There are no provisions in either the compiler or the run-time package to test for this condition. The increase in program size in the compiler or run-time package could cause a space problem in user programs.

Table 8-1. COBOL Subroutines Library

Name	Description
C$ADDP	Embed the sign character with the last data character.
C$BKSP	Backspace I/O on sequential file.
C$BSRT	Sort an array on a given character string.
C$CARG	Return USING argument information.
C$CBID	Bid a COBOL task.
C$CLOS	Close VDT and output file.
C$CMPR	Compare character strings logically.
C$CVDT	Close all VDTs currently open.
C$DLTE	Delete a file.
C$EXCP	Turn off function key accessibility.
C$GROF	Turn off graphics display option.
C$GRPH	Turn on graphics display option.
C$LOC	Return address of data argument.
C$MAPS	Map and return synonym value.
C$MFAP	Modify file access privilege.
C$MKEY	Modify a KIF alternate key attribute so that it is nonmodifiable in program declaration.
C$OPEN	Open VDT and output file.
C$PARM	Get parameter from terminal communications area.
C$RERR	Return last file I/O completion status.
C$RPRV	Read previous I/O on KIF.
C$SEPP	Separate embedded data character and sign character into data character and separate trailing sign.
C$SETS	Define or redefine synonym in terminal communications area.
C$SRCH	Binary search array for specified key value.
C$SVC	Issue an SVC to operating system.
C$TMPF	Set a temporary file flag that causes the next OPEN . . . OUTPUT statement to create a temporary file.
C$WRIT	Write the VDT screen contents to the output file or device.

All of these subroutines reside on the library .S$SYSLIB.C$SUBS. Use the LIBRARY or SEARCH command to link them with the COBOL program object modules. These routines must be included in the task segment of the link control file. A typical link control file, which can link any of the subroutines with COBOL program object modules, is as follows:

```
FORMAT IMAGE,REPLACE
LIBRARY .S$SYSLIB.C$SUBS
LIBRARY .SCI990.S$OBJECT
PROCEDURE RCOBOL
DUMMY
INCLUDE .S$SYSLIB.RCBPRC
TASK CBLTSK
INCLUDE .S$SYSLIB.RCBTSK
INCLUDE .S$SYSLIB.RCBMPD
INCLUDE <COBOL object module>
END
```

8.3 ASSEMBLY LANGUAGE SUBROUTINES

Assembly language subroutines provide capabilities to the COBOL program not available through COBOL syntax. These capabilities include (but are not limited to) gaining access to system SVCs, and interfacing a routine to application environment processors and specialized data handling routines. To call assembly language routines, use the CALL statement. This statement transfers control from one object module to another within the program.

The CALL statement can be used to call subroutines written in COBOL and other languages provided the linkage conventions are compatible. For example, for the statement

```
CALL "PROGA" USING A1, A2, A3.
```

the COBOL compiler generates an argument list with the following format:

```
ARGLST    DATA 6      byte count of the argument list (twice the number
                      of arguments)
          DATA A1
          DATA A2
          DATA A3
```

NOTE

The argument list contains byte addresses. If the subroutine is designed to address words, the COBOL programmer must ensure that all parameters begin on a word boundary.

When the CALL statement is executed, register 0 (R0) is loaded with the address of the argument list, register 1 (R1) is loaded with the address of the argument decode routine, and subprogram PROGA is entered via a Branch and Load Workspace Pointer (BLWP) instruction. For example:

```
LI    R0,ARGLST
LI    R1,DEADDR
BLWP  @PROGA
```

The argument decode routine is the assembly language programmer's way of accessing information about a data item. The BLWP instruction is used to transfer control to the subroutine module. Subprogram PROGA must have an entry vector PROGA, defined as follows:

```
          .
          .
          .
          DEF    PROGA
PROGA     DATA   WP       WORKSPACE FOR PROGA
          DATA   START    FIRST INSTRUCTION
WP        BSS    32
START     EQU    $
          .
          .
          .
          END
```

Note that the subroutine in Figure 8-2 indirectly references register 0 of the calling routine by using register 13. This occurs in Line 15.

To return to the COBOL module, PROGA must execute a Return With Workspace Pointer (RTWP) instruction (assuming registers 13, 14, and 15 have not been modified by PROGA).

Information about each argument in the USING list of a CALL statement is accessible to the assembly language program through a COBOL run-time subroutine. This subroutine requires two arguments (in R0, R1), as follows:

- The assembly language subroutine workspace register 0 must be loaded with the relative argument number from the USING list for which information is needed.
- The assembly language subroutine workspace register 1 must be loaded with the address of a 10-byte buffer in which to store the descriptive information.

Figure 8-2 shows an assembly subroutine example called from the COBOL example in Figure 8-1.

```
 DXCBL          L.R.V YY.DDD COMPILED:MM/DD/YY HH:MM:SS  OPT=M      PAGE    1

SOURCE ACCESS NAME:    MANUAL.PG.SRC.FIG0801
OBJECT ACCESS NAME:    DUMY
LISTING ACCESS NAME:   MANUAL.PG.LST.FIG0801
OPTIONS:               M
PRINT WIDTH:           80
PAGE SIZE:             55
PROGRAM SIZE (LINES):  1000



 DXCBL          L.R.V YY.DDD COMPILED:MM/DD/YY HH:MM:SS  OPT=M      PAGE    2
LINE  DEBUG PG/LN   A...B.................................................
   1                IDENTIFICATION DIVISION.
   2                PROGRAM-ID. COBOLARG.
   3               *     THIS EXAMPLE SHOWS SUBROUTINE LINKAGE USING
   4               *     THE CALL VERB. IT ALSO DEMONSTRATES A
   5               *     TECHNIQUE OF OBTAINING INFORMATION ABOUT
   6               *     THE "USING" ARGUMENTS BEING PASSED.
   7                ENVIRONMENT DIVISION.
   8                CONFIGURATION SECTION.
   9                SOURCE-COMPUTER. TI-990.
  10                OBJECT-COMPUTER. TI-990.
  11                DATA DIVISION.
  12                FILE SECTION.
  13                WORKING-STORAGE SECTION.
  14
  15               ****  ALPHANUMERIC
  16                01   ANS      PIC X(20) VALUE  "CORRECT RESULT: 330".
  17
  18               ****  ALPHABETIC
  19                01   ABS      PIC A(20) VALUE  "COMPUTED RESULT:".
  20
  21               ****  DISPLAY SIGNED LEADING
  22                01   NL       PIC S9(6) SIGN LEADING VALUE  +45.
  23
  24               ****  DISPLAY SIGNED LEADING SEPARATE
  25                01   NLS      PIC S9(6) SIGN LEADING SEPARATE VALUE  55.
  26
  27               ****  DISPLAY SIGNED TRAILING
  28                01   NT       PIC S9(6) SIGN TRAILING VALUE  50.
  29
  30               ****  NUMERIC DISPLAY SIGNED (TRAILING SEPARATE)
  31                01   NSS      PIC S9(6) VALUE  30.
  32
  33               ****  NUMERIC DISPLAY UNSIGNED
  34                01   NSU      PIC 9(6)  VALUE  25.
  35
  36               ****  COMPUTATIONAL UNSIGNED
  37                01   NCU      PIC 9(5)  COMP   VALUE 15.
```

Figure 8-1. Example of COBOL Routine Calling Assembler Subroutine (Sheet 1 of 4)

```
 38
 39               ****  COMPUTATIONAL SIGNED
 40                01   NCS      PIC S9(5)  COMP   VALUE 20.
 41
 42               ****  BINARY SIGNED OR UNSIGNED (COMPUTATIONAL-1)
 43                01   NBS      PIC S9(5) COMP-1 VALUE 5.
 44
 45               ****  NUMERIC PACKED DECIMAL (COMPUTATIONAL-3)
 46                01   NPS      PIC S9(5) COMP-3 VALUE +10.
 47
 48               ****  MULTI-WORD BINARY UNSIGNED (COMPUTATIONAL-4)
 49                01   NMB      PIC 99    COMP-4 VALUE 35.
 50
 51               ****  MULTI-WORD BINARY SIGNED (COMPUTATIONAL-4)
 52                01   NMS      PIC S9(4) COMP-4 VALUE 40.
 53
 54               ****  NUMERIC EDITED
 55                01   NSE      PIC ZZ9.
 56


 DXCBL          L.R.V YY.DDD COMPILED:MM/DD/YY HH:MM:SS  OPT=M      PAGE    3
LINE  DEBUG PG/LN   A...B.....................................................
 57               ****  ALPHANUMERIC EDITED
 58                01   ANSE     PIC XX/XX/XX.
 59
 60               ****  GROUP
 61                01   GRP.
 62                     02 YR    PIC XX.
 63                     02 MO    PIC XX.
 64                     02 DA    PIC XX.
 65
 66                01   DATA-LENGTH  PIC 9(5).
 67                01   DIGIT-LENGTH PIC 9(5).
 68                01   ACTION       PIC X.
 69                01   SUB          PIC 99 COMP-1.
 70                01   ROW          PIC 99.
 71
 72               * BUFFER AREA IN WHICH ARGUMENT INFORMATION IS PLACED
 73               * BY THE ASSEMBLER SUBROUTINE.
 74
 75                01   ARG-TABLE.
 76                     02 ARG-ENTRY OCCURS 18.
 77                        03 ARG-CODE          PIC 9 COMP.
 78                        03 ARG-SCALE         PIC 9 COMP.
 79                        03 ARG-DATA-LENGTH   PIC S9(5) COMP-1.
 80                        03 ARG-DIGIT-LENGTH  PIC S9(5) COMP-1.
 81                        03 ARG-DATA-ADDRESS  PIC S9(5) COMP-1.
 82                        03 ARG-PIC-ADDRESS   PIC S9(5) COMP-1.
 83
 84                PROCEDURE DIVISION.
```

Figure 8-1. Example of COBOL Routine Calling Assembler Subroutine (Sheet 2 of 4)

```
 85  >0000          MAIN-PROG.
 86  >0000               MOVE SPACES TO ARG-TABLE.
 87  >0004               CALL "DECODE" USING ARG-TABLE ANS ABS NL NLS
 88                          NT NSS NSU NCU NCS NBS NPS NMB NMS
 89                          NSE GRP  "123456" +55 SPACE.
 90  >0006               DISPLAY "DATA      DIGIT"  LINE 1 ERASE.
 91  >000E               DISPLAY "LENGTH    LENGTH".
 92  >0012               PERFORM DISP-ARG VARYING SUB FROM 1 BY 1
 93                         UNTIL SUB > 18.
 94  >0026               ACCEPT ACTION LINE 24 PROMPT.
 95  >002E               STOP RUN.
 96  >0030          DISP-ARG.
 97  >0030               COMPUTE ROW = SUB + 3.
 98  >0036               MOVE ARG-DATA-LENGTH (SUB) TO DATA-LENGTH.
 99  >0040               DISPLAY DATA-LENGTH LINE ROW.
100  >0046               MOVE ARG-DIGIT-LENGTH (SUB) TO DIGIT-LENGTH.
101  >0050               DISPLAY DIGIT-LENGTH LINE ROW POSITION 10.
102  >005A          END-DISP. EXIT.
103          ZZZZZZ END PROGRAM.                                *** END OF FILE
```

```
DXCBL           L.R.V YY.DDD COMPILED:MM/DD/YY HH:MM:SS  OPT=M      PAGE    4
ADDRESS  SIZE   DEBUG ORDER TYPE                   NAME

>002A     20    ANS     0   ALPHANUMERIC          ANS

>003E     20    ABS     0   ALPHABETIC            ABS

>0052      6    NL      0   NUM LEAD SIGNED       NL

>0058      7    NLS     0   NUM SEP LEAD SIGNED   NLS

>0060      6    NT      0   NUM TRAIL SIGNED      NT

>0066      7    NSS     0   NUMERIC SIGNED        NSS

>006E      6    NSU     0   NUMERIC UNSIGNED      NSU

>0074      5    NCU     0   COMP UNSIGNED         NCU

>007A      6    NCS     0   COMP SIGNED           NCS

>0080      2    NBS     0   BINARY SIGNED         NBS

>0082      3    NPS     0   PACKED SIGNED         NPS

>0086      1    NMB     0   MULTI BINARY          NMB

>0088      2    NMS     0   MULTI BINARY SIGNED   NMS
```

Figure 8-1. Example of COBOL Routine Calling Assembler Subroutine (Sheet 3 of 4)

```
>008A      3    NSE    0   NUMERIC EDITED        NSE

>008E      8    ANSE   0   ALPHANUMERIC EDITED   ANSE

>0096      6    GRP    0   GROUP                 GRP
>0096      2    ANS    0   ALPHANUMERIC           YR
>0098      2    ANS    0   ALPHANUMERIC           MO
>009A      2    ANS    0   ALPHANUMERIC           DA

>009C      5    NSU    0   NUMERIC UNSIGNED      DATA-LENGTH

>00A2      5    NSU    0   NUMERIC UNSIGNED      DIGIT-LENGTH

>00A8      1    ANS    0   ALPHANUMERIC          ACTION

>00AA      2    NBS    0   BINARY SIGNED         SUB

>00AC      2    NSU    0   NUMERIC UNSIGNED      ROW

>00AE    180    GRP    0   GROUP                 ARG-TABLE
>00AE     10    GRP    1   GROUP                  ARG-ENTRY
>00AE      1    NCU    1   COMP UNSIGNED          ARG-CODE
>00AF      1    NCU    1   COMP UNSIGNED          ARG-SCALE
>00B0      2    NBS    1   BINARY SIGNED          ARG-DATA-LENGTH
>00B2      2    NBS    1   BINARY SIGNED          ARG-DIGIT-LENGTH
>00B4      2    NBS    1   BINARY SIGNED          ARG-DATA-ADDRESS
>00B6      2    NBS    1   BINARY SIGNED          ARG-PIC-ADDRESS


DXCBL          L.R.V YY.DDD COMPILED:MM/DD/YY HH:MM:SS  OPT=M      PAGE    5
ADDRESS  SIZE  DEBUG ORDER TYPE                 NAME

READ ONLY BYTE SIZE =        >019C

READ/WRITE BYTE SIZE =       >018E

OVERLAY SEGMENT BYTE SIZE =  >0000

TOTAL BYTE SIZE =            >032A

    0 ERRORS

    0 WARNINGS


DXCBL          L.R.V YY.DDD COMPILED:MM/DD/YY HH:MM:SS  OPT=M      PAGE    6
PROGRAM    USING COUNT

DECODE         19
```

Figure 8-1. Example of COBOL Routine Calling Assembler Subroutine (Sheet 4 of 4)

```
           SDSMAC L.R.V YY.DDD    HH:MM:SS FRIDAY, NOV 07, 1980.
ACCESS NAMES TABLE                                              PAGE 0001
SOURCE ACCESS NAME=      MANUAL.PG.SRC.FIG0802
OBJECT ACCESS NAME=      DUMY
LISTING ACCESS NAME=     MANUAL.PG.LST.FIG0802
ERROR ACCESS NAME=
OPTIONS=
MACRO LIBRARY PATHNAME=



DECODE     SDSMAC L.R.V YY.DDD     HH:MM:SS FRIDAY, NOV 07 1980.
                                                                PAGE 0002
 0001                      IDT  'DECODE'
 0002            * TITLE:    DECODE
 0003            * ABSTRACT: OBTAIN COBOL "USING" ARGUMENT INFORMATION AND
 0004            *           RETURN INFO TO CALLER
 0005            * CALLING SEQUENCE:
 0006            *           <R0>::ADDRESS OF ARGUMENT LIST
 0007            *                 WORD 0: LENGTH OF ARG LIST IN BYTES
 0008            *                 WORD 1-N: ARGUMENT ADDRESS
 0009            *           <R1L::ADDRESS OF ARG DECODE ROUTINE
 0010            *           CALL TO 'DECODE' IS MADE VIA 'BLWP' INSTRUCTION
 0011                      DEF   DECODE
 0012 0000 0004' DECODE DATA WS,ARG000
      0002 0024'
 0013 0004       WS     BSS  32
 0014      0024' ARG000 EQU  $
 0015 0024 C09D         MOV  *R13,R2          GET ADDR ARG LIST
 0016 0026 C0F2         MOV  *R2+,R3          GET NUMBER OF ARGUMENTS
 0017 0028 0913         SRL  R3,1             CONVERT TO WORDS
 0018 002A 0206         LI   R6,1             INITIALIZE TO FIRST ARG
      002C 0001
 0019 002E C072         MOV  *R2+,R1          GET ARG TABLE ADDRESS
 0020 0030 0204         LI   R4,10            LENGTH OF DOPE ENTRY
      0032 000A
 0021 0034 C16D         MOV  2(R13),R5        ARG DECODE ROUTINE
      0036 0002
 0022      0038' ARG010 EQU  $
 0023 0038 C006         MOV  R6,R0            SET ARG NUMBER
 0024 003A 0415         BLWP *R5              MAKE CALL
 0025 003C 1605         JNE  ARG020           IF ERROR
 0026 003E 0603         DEC  R3               SET UP FOR NEXT ARG
 0027 0040 1203         JLE  ARG020           NO MORE ARG
 0028 0042 A044         A    R4,R1            INCR BY ARG ENTRY LENGTH
 0029 0044 0586         INS  R6               INCR ARG COUNT
 0030 0046 10F8         JMP  ARG010
 0031 0048 0380  ARG020 RTWP
 0032                   END
NO ERRORS,     NO WARNINGS
```

Figure 8-2. Example of Assembler Subroutine Called by COBOL

Refer to Appendix D for details on the routine, C$CARG. This routine is supplied with the COBOL Subroutine Library Package, which returns descriptive information for any given argument.

The COBOL calling module must provide the following 10-byte buffer.

```
01  DATA-BLOCK.
    02 DATA-CODE            PIC 99 COMP.
    02 DATA-SCALE           PIC 99 COMP.
    02 DATA-LENGTH          PIC S9(5) COMP-1.
    02 DATA-DIGIT-LENGTH    PIC S9(5) COMP-1.
    02 DATA-ADDR            PIC S9(5) COMP-1.
    02 DATA-PIC-ADDR        PIC S9(5) COMP-1.
```

DATA-CODE is the section type containing the argument declaration.

Bits 0-2 contain one of the following:

Bits	Description
110	Overlay segment literal
100	Literal
010	Linkage
001	File or working storage

Bits 3-7 contain the format code, as shown in Table 8-2:

Table 8-2. Format Codes for Calling Module

Bits 3-7	Debug Type	Name Description
0000X	NSE	Numeric String Edited (X = 1 if BLANK WHEN ZERO)
00010	FIG	Figurative Constant
0010X	ABS	Alphabetic String (X = 1 if JUSTIFIED RIGHT)
01000	ANSE	Alphanumeric String Edited
0101X	ANS	Alphanumeric String (X = 1 if JUSTIFIED)
01100	GRP	Group (fixed size)
01101	GRP	Group (variable size)
10000	NSU	Numeric String Unsigned
10010	NSS	Numeric String Separate Trailing Signed Character
10011	NLS	Numeric String Separate Leading Signed Character
10100	NCU	Numeric Computational Unsigned
10110	NCS	Numeric Computational Separate Trailing Sign Character
10111	NT	Numeric String Trailing Signed Character
11010	NPS	Numeric Packed Signed
11011	NL	Numeric String Leading Signed Character
11000	NX	Index Data Item
11100	NUMERIC	Compiler Generated TEMP
11101	NMB	Multiword Binary (COMP-4)
11110	NBS	Numeric Binary Signed (COMP-1)
11111	NMS	Multiword Binary Signed (COMP-4)

DATA-SCALE contains the data scaling factor needed to express the data item as an integer times a power of 10; to express 1.2340 as an integer requires a scale of -3 (that is, 1234×10^{-3}).

DATA-LENGTH contains the actual data item storage size. COMP data length is the number of specified digits in the picture clause plus the sign, if present. For example,

S9(3) COMP has a length of 4.
9(2) COMP has a length of 2.

COMP-1 data length is always 2. For example,

S9(3) COMP-1 has a length of 2.
9(5) COMP-1 has a length of 2.

DISPLAY with SIGN LEADING or SIGN TRAILING or no "S" in the picture clause; data length is the number of specified digits in the picture clause. For example,

S9(3) SIGN TRAILING has a length of 3.
S9(3) SIGN LEADING has a length of 3.
9(3) has a length of 3.

DISPLAY with SIGN SEPARATE clause or with no SIGN clause; data length is the number of specified digits in the picture clause plus the sign. For example,

S9(3) SIGN SEPARATE has a length of 4.
S9(5) SIGN TRAILING SEPARATE has a length of 6.
S9(3) has a length of 4.

COMP-3 data length is the number of specified digits in the picture clause, forced upward to be odd, plus 1, divided by 2. For example,

S9(3) COMP-3 has a length of 2.
S9(4) COMP-3 has a length of 3.

COMP-4 data length is the number of specified digits in the picture clause, as follows:

1–2 digits yield data length of 1 byte.
3–4 digits yield data length of 2 bytes.
5–9 digits yield data length of 4 bytes.
10–18 digits yield data length of 8 bytes.

For example,

S9(2) COMP-4 has a length of 1.
S9(4) COMP-4 has a length of 2.
S9(5) COMP-4 has a length of 4.
S9(15) COMP-4 has a length of 8.

DATA-DIGIT-LENGTH contains the number of digit positions specified in the picture clause. COMP-3 is forced odd. For example,

S9(3) COMP has a value of 3.
9(3) COMP has a value of 3.
S9(3) COMP-1 has a value of 3.
9(5) COMP-1 has a value of 5.
S9(3) has a value of 3.
S9(3) SIGN LEADING has a value of 3.
9(3) has a value of 3.
S9(3) SIGN SEPARATE has a value of 3.
S9(5) SIGN TRAILING SEPARATE has a value of 5.
S9(3) COMP-3 has a value of 3.
S9(4) COMP-3 has a value of 5.

DATA-ADDR contains the address of the data item. DATA-PIC-ADDR contains the address of the data picture for the editing data types NSE and ANSE.

9

Interfacing to Productivity Tools

9.1 GENERAL

The following productivity tools can interface with COBOL modules:

- TIFORM
- Sort/Merge
- Database Management System (DBMS)
- Query
- Communications

9.2 TIFORM

TIFORM is a software utility package for controlling the interactive interface to an application. TIFORM provides convenient control of complex screen formats for COBOL applications. TIFORM includes an interactive screen drawing capability and a screen description language compiler. Through the use of these tools, TIFORM isolates the description of the screen format from the procedural code of the application. This allows applications to become independent of the terminal. TIFORM also includes:

- All available VDT features (blink, dim, high-intensity, no display)
- Character and field level editing
- Significant improvement in the time required to develop interactive applications

The entry points provided for COBOL access to the TIFORM applications interface routines (Table 9-1) are all of the form CF$xxx or CX$xxx, where xxx denotes a unique TIFORM function. Refer to the *TIFORM Reference Manual* for a detailed explanation of these calls.

Table 9-1. COBOL Entry Points to the Applications Interface Routines

Calls	Meaning
CX$AEK	Arm Event Keys
CX$CF	Close Form
CX$CN	Control Functions
CX$DAK	Disarm Event Keys
CX$OF	Open Form
CX$PS	Prepare Segment
CX$REA	Read a Group
CX$REX	Read, Indexed
CX$RF	Reset Form
CX$RFX	Reset Form Indexed
CX$RXC	Read, Indexed, with Cursor Return
CX$STS	Declare Status Block
CX$WM	Write Message
CX$WRC	Write, Indexed, with Reply, and Cursor Return
CX$WRI	Write a Group
CX$WWR	Write with Reply
CX$WX	Write, Indexed
CX$WXR	Write, Indexed, with Reply

Figure 9-1 illustrates how a COBOL module interfaces with TIFORM. Figure 9-2 illustrates the TIFORM screen description. The following serves as the link control file for linking the COBOL module with the TIFORM module.

```
FORMAT IMAGE,REPLACE
LIBRARY .S$TIFORM.O              TIFORM INTERFACE MODULES
PROC RCOBOL
DUMMY
INCLUDE .S$SYSLIB.RCBPRC
TASK TIFRMTSK
INCLUDE .S$SYSLIB.RCBTSK
INCLUDE .S$SYSLIB.RCBMPD
INCLUDE EX.TIFORM                COBOL MODULE
INCLUDE (CX$MTASK)
END
```

Figure 9-1. COBOL Module Interfacing With TIFORM (Sheet 1 of 4)

```
 DXCBL          L.R.V YY.DDD COMPILED:MM/DD/YY HH:MM:SS  OPT=M      PAGE    1

SOURCE ACCESS NAME:    MANUAL.PG.SRC.FIG0901
OBJECT ACCESS NAME:    MANUAL.PG.OBJ.FIG0901
LISTING ACCESS NAME:   MANUAL.PG.LST.FIG0901
OPTIONS:               M
PRINT WIDTH:           80
PAGE SIZE:             55
PROGRAM SIZE (LINES):  1000



 DXCBL          L.R.V YY.DDD COMPILED:MM/DD/YY HH:MM:SS  OPT=M      PAGE    2
LINE  DEBUG PG/LN   A...B.....................................................
   1                IDENTIFICATION DIVISION.
   2                PROGRAM-ID. TIFORM.
   3               *     THIS PROGRAM DEMONSTRATES CALLING TECHNIQUE FOR
   4               *     INTERFACING TIFORM WITH COBOL.
   5                ENVIRONMENT DIVISION.
   6                CONFIGURATION SECTION.
   7                SOURCE-COMPUTER. TI-990.
   8                OBJECT-COMPUTER. TI-990.
   9                DATA DIVISION.
  10                WORKING-STORAGE SECTION.
  11
  12                01 INPUT-DATA.
  13                   03 EMPLOYEE-NO        PIC X(6).
  14                   03 CLEAR-NAME         PIC X(30) VALUE LOW-VALUES.
  15                01 READ-WRITE-DATA.
  16                   03 EMPLOYEE-NAME  PIC X(30).
  17                   03 DONE           PIC X     VALUE "'N".
  18                01 NUMBER-DATA         PIC X(30) VALUE
  19                   "000000111111222222333333444444".
  20                01 NUMBER-TABLE REDEFINES NUMBER-DATA.
  21                   03 NUMBER-ENT PIC X(6) OCCURS 5.
  22                01 NAMES-DATA          PIC X(50) VALUE
  23                   "A. ANTOIN B. BARTOK C. CARTER D. DARWIN E. ERDLE".
  24                01 NAMES-TABLE REDEFINES NAMES-DATA.
  25                   03 NAME-ENT PIC X(10) OCCURS 5.
  26                01 TIFORM-STATUS-BLOCK.
  27                   03 FORM-STATUS    PIC 99.
  28                   03 OPSYS-STATUS   PIC XX.
  29                   03 FILLER         PIC X(36).
  30                01 FORM-NAME           PIC X(6)  VALUE "DEMOFM".
  31               *SYNONYM 'DIRECTRY' IS IMPLIED BY BLANKS.
  32                01 DIRECTORY           PIC XX    VALUE "  ".
  33               *SYNONYM 'ME' IS IMPLIED BY BLANKS.
  34                01 TUBE                PIC XX    VALUE "  ".
  35                01 SEG-NAME            PIC X(6)  VALUE "SEG1  ".
  36                01 GRP1                PIC X(6)  VALUE "GROUP1".
  37                01 GRP                 PIC X(6)  VALUE "GROUPA".
```

Figure 9-1. COBOL Module Interfacing With TIFORM (Sheet 2 of 4)

```
 38                01 X                    PIC 9.
 39                PROCEDURE DIVISION.
 40  >0000         MAIN-PROGRAM.
 41               ***** DECLARE STATUS BLOCK *****
 42  >0000              CALL "CX$STS" USING TIFORM-STATUS-BLOCK.
 43               ***** OPEN FORM *****
 44  >0002              CALL "CX$OF" USING FORM-NAME,
 45                          DIRECTORY, TUBE.
 46               ***** PREPARE SEGMENT *****
 47  >0004              CALL "CX$PS" USING SEG-NAME.
 48  >0008          READ NO.
 49  >0008              MOVE LOW-VALUES TO EMPLOYEE-NO.
 50               ***** WRITE WITH REPLY *****
 51  >000C              CALL "CX$WWR" USING GRP1, INPUT-DATA, INPUT-DATA.
 52  >000E              MOVE SPACES TO EMPLOYEE-NAME.
 53  >0012              PERFORM FIND-NO VARYING X FROM 1 BY 1
 54                        UNTIL X > 5.
 55               ***** WRITE WITH REPLY *****
 56  >0026              CALL "CX$WWR" USING GRP, READ-WRITE-DATA,
```

```
 DXCBL          L.R.V YY.DD COMPILED:MM/DD/YY HH:MM:SS  OPT=M      PAGE    3
LINE  DEBUG PG/LN   A...B.................................................
 57                        READ-WRITE-DATA.
 58  >0028              IF DONE = "N" GO READ-NO.
 59               ***** CLOSE FORM *****
 60  >0030              CALL "CX$CF".
 61  >0034          THATS-ALL.
 62  >0034              STOP RUN.
 63  >0036          FIND-NO.
 64  >0036              IF EMPLOYEE-NO = NUMBER-ENT(X)
 65                        MOVE NAME-ENT(X) TO EMPLOYEE-NAME,
 66                        MOVE 6 TO X;
 67                     ELSE IF X = 5 MOVE "INVALID NUMBER" TO
 68                        EMPLOYEE-NAME.
 69          ZZZZZZ END PROGRAM.                          *** END OF FILE
```

```
 DXCBL          L.R.V YY.DDD COMPILED:MM/DD/YY HH:MM:SS  OPT=M      PAGE    4
 ADDRESS  SIZE  DEBUG ORDER TYPE                  NAME

  >002A     36   GRP     0   GROUP                INPUT-DATA
  >002A      6   ANS     0   ALPHANUMERIC          EMPLOYEE-NO
  >0030     30   ANS     0   ALPHANUMERIC          CLEAR-NAME

  >004E     31   GRP     0   GROUP                READ-WRITE-DATA
  >004E     30   ANS     0   ALPHANUMERIC          EMPLOYEE-NAME
  >006C      1   ANS     0   ALPHANUMERIC          DONE
```

Figure 9-1. COBOL Module Interfacing With TIFORM (Sheet 3 of 4)

```
  >006E     30   ANS    0   ALPHANUMERIC          NUMBER-DATA

  >006E     30   GRP    0   GROUP                 NUMBER-TABLE
  >006E      6   ANS    1   ALPHANUMERIC           NUMBER-ENT

  >008C     50   ANS    0   ALPHANUMERIC          NAMES-DATA

  >008C     50   GRP    0   GROUP                 NAMES-TABLE
  >008C     10   ANS    1   ALPHANUMERIC           NAME-ENT

  >00BE     40   GRP    0   GROUP                 TIFORM-STATUS-BLOCK
  >00BE      2   NSU    0   NUMERIC UNSIGNED       FORM-STATUS
  >00C0      2   ANS    0   ALPHANUMERIC           OPSYS-STATUS

  >00E6      6   ANS    0   ALPHANUMERIC          FORM-NAME

  >00EC      2   ANS    0   ALPHANUMERIC          DIRECTORY

  >00EE      2   ANS    0   ALPHANUMERIC          TUBE

  >00F0      6   ANS    0   ALPHANUMERIC          SEG-NAME

  >00F6      6   ANS    0   ALPHANUMERIC          GRP1

  >00FC      6   ANS    0   ALPHANUMERIC          GRP

  >0102      1   NSU    0   NUMERIC UNSIGNED      X

READ ONLY BYTE SIZE =        >0154

READ/WRITE BYTE SIZE =       >0112

OVERLAY SEGMENT BYTE SIZE =  >0000

TOTAL BYTE SIZE =            >0266

    0 ERRORS

    0 WARNINGS


DXCBL          L.R.V YY.DDD COMPILED:MM/DD/YY HH:MM:SS  OPT=M      PAGE    5
PROGRAM   USING  COUNT

 CX$CF           0
 CX$OF           3
 CS$PS           1
 CX$STS          1
 CX$WWR          3
```

Figure 9-1. COBOL Module Interfacing With TIFORM (Sheet 4 of 4)

```
SEGMENT MASK MSK1,CLEAR=Y.
 .
     DISPLAY GR=Y.
     M(01,01)
  'BLLL
```

```
END FIELD NUMBER.
.
FIELD NAME.
    POSITION (4,40)L30.
    OUTPUT.
END FIELD NAME.
.
FIELD YESNOF.
    POSITION (4,78)L1.
    REQ.
    CHAR LIST=YESNO.
END FIELD YESNOF.
.
LIST CHAR DIGITS=0..9.
.
LIST CHAR YESNO='Y','N'.
.
GROUP GROUPA=NAME,YESNOF.
.
GROUP GROUP1=NUMBER,NAME.
.
.
END SEGMENT SEG1.
```

Figure 9-2. TIFORM VDT Screen Description (Sheet 2 of 2)

9.3 SORT/MERGE

A comprehensive Sort/Merge package is supported. SCI commands provide access to the Sort/Merge package in batch or interactive mode. Both Sort and Merge support the following features:

- Record selection
- Reformatting on input
- Summarizing on output

Ascending key order, descending key order, or an alternate collating sequence may be specified. Any number of keys can be specified as long as the total is less than 256 characters. The merge process supports up to five input files. The sort process allows the following:

- Key sort (tag-along)
- Summary sort (summary tag-along)
- Address only sort

Figure 9-3 is a COBOL routine that calls Sort/Merge and passes records read by COBOL to Sort/Merge. The sorted records are output to a disk file.

```
 DXCBL          L.R.V YY.DDD COMPILED:MM/DD/YY HH:MM:SS  OPT=M      PAGE    1

SOURCE ACCESS NAME:    MANUAL.PG.SRC.FIG0903
OBJECT ACCESS NAME:    DUMY
LISTING ACCESS NAME:   MANUAL.PG.LST.FIG0903
OPTIONS:               M
PRINT WIDTH:           80
PAGE SIZE:             55
PROGRAM SIZE (LINES):  1000



 DXCBL          L.R.V YY.DDD COMPILED:MM/DD/YY HH:MM:SS  OPT=M      PAGE    2
LINE  DEBUG PG/LN    A...B.....................................................
   1                 IDENTIFICATION DIVISION.
   2                 PROGRAM-ID. CPNP.
   3                *THIS IS A SORT/MERGE INFORMATION QUEUE TEST. IT
   4                *TESTS THE CASE WHERE INPUT IS DIRECTED BY THE
   5                *COBOL PROGRAM (@PROC@) AND OUTPUT IS DIRECTLY
   6                *FROM THE SORT/MERGE TO A FILE (NO @PROC@).
   7                 AUTHOR. TEXAS INSTRUMENTS  FDT.
   8                 DATE-WRITTEN. 11-6-76.
   9                 ENVIRONMENT DIVISION.
  10                 CONFIGURATION SECTION.
  11                 SOURCE-COMPUTER. TI-990.
  12                 OBJECT-COMPUTER. TI-990.
  13                 INPUT-OUTPUT SECTION.
  14                 FILE-CONTROL.
  15                      SELECT INFILNAME ASSIGN TO INPUT "INFILE";
  16                         ACCESS MODE IS SEQUENTIAL.
  17                 DATA DIVISION.
  18                 FILE SECTION.
  19                 FD  INFILNAME
  20                     DATA RECORD IS INFILRCRD
  21                     LABEL RECORDS ARE STANDARD
  22                     RECORD CONTAINS 80 CHARACTERS
  23                     BLOCK CONTAINS 10 RECORDS.
  24                 01  INFILRCRD     PIC X(80).
  25                 WORKING-STORAGE SECTION.
  26                 77 MAX-NO-RECS          PIC 9(5)  VALUE IS 10
  27                                                   USAGE IS COMP-1.
  28                 77 STATIS               PIC 9(5)  USAGE IS COMP-1.
  29                 77 OUTSTAT              PIC 9(5).
  30                 77 RECORD-LENGTH        PIC 9(5)  VALUE IS 80
  31                                                   USAGE IS COMP-1.
  32                 77 RECORD-AREA-LENGTH   PIC 9(5)  USAGE IS COMP-1.
  33                 77 RETES-RECEIVED       PIC 9(5)  USAGE IS COMP-1.
  34                 77 OFILRCRD             PIC X(80).
```

Figure 9-3. COBOL Routine Calling Sort/Merge (Sheet 1 of 7)

```
 35                77 ALLDONE             PIC 9(5) VALUE IS ZERO
 36                                       USAGE IS COMP-1.
 37              *SORT-CONTROL-BLOCK CONTAINS THE SORT/MERGE
 38              *CONTROL SPECIFICATIONS.
 39               01 SORT-CONTROL-BLOCK.
 40                    03 HEADER.
 41                       05 SEQ          PIC X(5)  VALUE IS "00000".
 42                       05 FILLER       PIC A     VALUE IS "H".
 43                       05 SORT-TYPE    PIC A(6)  VALUE IS "SORTR".
 44                       05 MAX-TOT-CONTL-LEN PIC 9(5) VALUE IS 6.
 45                       05 ASCND-DSCND PIC A      VALUE IS "A".
 46                       05 FILLER       PIC X(7)  VALUE IS SPACES.
 47                       05 COLLATNG-SEQ PIC X     VALUE IS SPACE.
 48                       05 PRINT-OPTION PIC X     VALUE IS "4".
 49                       05 OUTPUT-OPTION PIC X    VALUE IS SPACE.
 50                       05 OUTPUT-REC-LEN PIC X(4) VALUE IS "0080".
 51                       05 VERIFY-OPTN    PIC XX  VALUE IS SPACE.
 52                       05 WRK-SPACE    PIC X(5)  VALUE IS "08000".
 53                       05 FILLER       PIC X(5)  VALUE IS SPACES.
 54              *OUTPUT IS DIRECTLY FROM THE SORT TO A FILE.
 55                    03 OUT-FILE-SPEC.
 56                       05 SEQ          PIC X(5)  VALUE IS "00001".


 DXCBL          L.R.V YY.DDD COMPILED:MM/DD/YY HH:MM:SS  OPT=M      PAGE    3
LINE  DEBUG PG/LN   A...B.....................................................
 57                       05 FILLER       PIC A     VALUE IS "D".
 58                       05 FILE-USE     PIC A     VALUE IS "O".
 59                       05 FILE-TYPE    PIC A     VALUE IS "S".
 60                             05 PATHNAME   PIC X(36) VALUE IS ".OCPNP".
 61                    03 CNT-OUT-FILE-SPEC.
 62                       05 SEQ          PIC X(5)  VALUE IS "00002".
 63                       05 FILLER       PIC A     VALUE IS "D".
 64                       05 FILE-USE     PIC A     VALUE IS "A".
 65                       05 LOG-REC-SIZ PIC 9(4)   VALUE IS 80.
 66                       05 PHY-REC-SIZ PIC 9(4)   VALUE IS 800.
 67                       05 NUM-PHY-REC PIC X(8)   VALUE IS SPACES.
 68                       05 FILLER       PIC X(21) VALUE IS SPACES.
 69                    03 WRK-FILE-SPEC.
 70                       05 SEQ          PIC X(5)  VALUE IS "00003".
 71                       05 FILLER       PIC A     VALUE IS "D".
 72                       05 FILE-USE     PIC A     VALUE "W".
 73                       05 EXPAND-ALLOC-FLG PIC X VALUE IS "E".
 74                       05 VOLUME       PIC X(8)  VALUE IS "DS01".
 75                       05 FILLER       PIC X(28) VALUE IS SPACES.
 76              *INPUT IS DIRECTED BY THE COBOL PROGRAM.
```

Figure 9-3. COBOL Routine Calling Sort/Merge (Sheet 2 of 7)

```
 77                     03 INPT-FILE-DESCRPT.
 78                        05 SEQ          PIC X(5)  VALUE IS "00004".
 79                        05 FILLER       PIC A     VALUE IS "D".
 80                        05 FILE-USE     PIC A     VALUE IS "I".
 81                        05 FILE-TYPE    PIC A     VALUE IS "S".
 82                        05 PATHNAME     PIC X(36) VALUE IS "@PROC@".
 83                     03 INPT-FILE-CONTIN.
 84                        05 SEQ          PIC X(5)  VALUE IS "00008".
 85                        05 FILLER       PIC A     VALUE IS "D".
 86                        05 FILE-USE     PIC A     VALUE IS "A".
 87                        05 LOG-SIZE     PIC X(4)  VALUE IS "0080".
 88                        05 FILLER       PIC X(4)  VALUE IS SPACES.
 89                        05 NUM-SRT-RECS PIC X(8) VALUE IS "00000401".
 90                        05 FILLER       PIC X(21) VALUE IS SPACES.
 91                     03 REFORMAT-DESCRIPTION-0.
 92                        05 SEQ          PIC X(5)  VALUE IS "00010".
 93                        05 FILLER       PIC A     VALUE IS "F".
 94                        05 FIELD-TYPE-CMMT PIC X VALUE IS "N".
 95                        05 CHARACTR-USE PIC A     VALUE IS "C".
 96                        05 FIELD-LOC.
 97                           07 BEG-RECRD-POS PIC X(4) VALUE IS "0032".
 98                           07 END-RECRD-POS PIC X(4) VALUE IS "0037".
 99                        05 CONDTN-FORCD-CHAR PIC X   VALUE IS SPACE.
100                        05 FORCD-CHAR  PIC X      VALUE IS SPACE.
101                        05 CONTIN-LIN  PIC X      VALUE IS SPACE.
102                        05 OUFLW-FLD-LEN PIC X(3) VALUE IS SPACES.
103                        05 FILLER       PIC X(22) VALUE IS SPACES.
104                     03 REFORMAT-DESCRIPTION.
105                        05 SEQ          PIC X(5)  VALUE IS "00014".
106                        05 FILLER       PIC A     VALUE IS "F".
107                        05 FIELD-TYPE-CMMT PIC X VALUE IS "D".
108                        05 CHARACTR-USE PIC A     VALUE IS "C".
109                        05 FIELD-LOC.
110                           07 BEG-RECRD-POS PIC X(4) VALUE IS "0001".
111                           07 END-RECRD-POS PIC X(4) VALUE IS "0031".
112                        05 CONDTN-FORCD-CHAR PIC X   VALUE IS "C".


 DXCBL          L.R.V YY.DDD COMPILED:MM/DD/YY HH:MM:SS  OPT=M      PAGE    4
LINE  DEBUG PG/LN   A...B.....................................................
113                        05 FORCD-CHAR  PIC X      VALUE IS SPACE.
114                        05 CONTIN-LIN  PIC X      VALUE IS SPACE.
115                        05 OUFLW-FLD-LEN  PIC X(3) VALUE IS SPACES.
116                        05 FILLER       PIC X(22)  VALUE IS SPACES.
117                     03 REFORMAT-DESCRIPTION-3.
118                        05 SEQ          PIC X(5)   VALUE IS "00016".
119                        05 FILLER       PIC A      VALUE IS "F".
120                        05 FIELD-TYPE-CMMT PIC X   VALUE IS "D".
121                        05 CHARACTR-USE PIC A      VALUE IS "C".
122                        05 FIELD-LOC.
```

Figure 9-3. COBOL Routine Calling Sort/Merge (Sheet 3 of 7)

```
123                          07 BEG-RECRD-POS PIC X(4) VALUE IS "0038".
124                          07 END-RECRD-POS PIC X(4) VALUE IS "0080".
125                       05 CONDTN-FORCD-CHAR PIC X   VALUE IS SPACE.
126                       05 FORCD-CHAR  PIC X      VALUE IS SPACE.
127                       05 CONTIN-LIN  PIC X      VALUE IS SPACE.
128                       05 OUFLW-FLD-LEN PIC X(3) VALUE IS SPACES.
129                       05 FILLER      PIC X(22)  VALUE IS SPACES.
130                    03 ENDKRD         PIC X(44)  VALUE IS "/*".
131                 PROCEDURE DIVISION.
132                *----------------------------------------COBOL EXAMPLE 3
133  >0000          MAIN-PROGRAM.
134                *INITIALIZE SORT/MERGE.
135  >0000               CALL "SRTINT" USING SORT-CONTROL-BLOCK,
136                           MAX-NO-RECS, STATIS.
137  >0002                    IF STATIS NOT EQUAL ZERO GO TO ERRSTRT.
138                *----------------------------------------COBOL EXAMPLE 3
139                * START THE INPUT SECTION.
140  >000A               OPEN INPUT INFILNAME.
141  >0012             NEXREC.
142  >0012               READ INFILNAME AT END GO TO BEGWRT.
143  >001A               CALL "SENREC" USING INFILRCRD,
144                           RECORD-LENGTH, STATIS.
145  >001C               IF STATIS NOT EQUAL ZERO GO TO ERRSEN.
146  >0024               GO TO NEXREC.
147                *----------------------------------------COBOL EXAMPLE 3
148                * START THE OUTPUT SECTION.
149  >0026             BEGWRT.
150                *----------------------------------------COBOL EXAMPLE 3
151                *  BEGIN SORT PHASE. SENDING A RECORD LENGTH OF 0 (ALLDONE)
152                *  INDICATES THAT THE LAST RECORD HAS BEEN SENT.
153  >0026               CALL "SENREC" USING INFILRCRD, ALLDONE, STATIS.
154  >0028               IF STATIS NOT EQUAL ZERO GO TO ERRSEN.
155                *----------------------------------------COBOL EXAMPLE 3
156  >0030               CLOSE INFILNAME.
157  >0038          CHKSORT.
158  >0038               CALL "SMSTAT" USING STATIS.
159  >003A               IF STATIS NOT EQUAL ZERO GO TO ERRWRT.
160                * SORT IS DONE.
161  >0042               GO TO END-IT.
162                *----------------------------------------COBOL EXAMPLE 3
163  >0044             ERRSTRT.
164  >0044               DISPLAY " ERROR IN STRINT CALL.".
165  >0048               MOVE STATIS TO OUTSTAT.
166  >004C               DISPLAY OUTSTAT.
167  >0050               GO TO END-IT.
168  >0052             ERRSEN.
```

Figure 9-3. COBOL Routine Calling Sort/Merge (Sheet 4 of 7)

```
 DXCBL          L.R.V YY.DDD COMPILED:MM/DD/YY HH:MM:SS  OPT=M      PAGE    5
LINE  DEBUG PG/LN   A...B.........................................................
 169  >0052              DISPLAY " ERROR IN SENREC CALL.".
 170  >0056              MOVE STATIS TO OUTSTAT.
 171  >005A              DISPLAY OUTSTAT.
 172  >005E              GO TO END-IT.
 173  >0060          ERRWRT.
 174  >0060              DISPLAY " ERROR IN SMSTAT.".
 175  >0064              MOVE STATIS TO OUTSTAT.
 176  >0068              DISPLAY OUTSTAT.
 177                *-------------------------------------------COBOL EXAMPLE 3
 178  >006E          END-IT.
 179  >006E              STOP RUN.
 180         ZZZZZZ END PROGRAM.                                *** END OF FILE
```

```
 DXCBL          L.R.V YY.DDD COMPILED:MM/DD/YY HH:MM:SS  OPT=M      PAGE    6
 ADDRESS  SIZE  DEBUG ORDER TYPE                  NAME

           800              FILE                  INFILNAME
  >0026     80   ANS    0   ALPHANUMERIC           INFILRCRD

  >007A      2   NBS    0   BINARY SIGNED         MAX-NO-RECS

  >007C      2   NBS    0   BINARY SIGNED         STATIS

  >007E      5   NSU    0   NUMERIC UNSIGNED      OUTSTAT

  >0084      2   NBS    0   BINARY SIGNED         RECORD-LENGTH

  >0086      2   NBS    0   BINARY SIGNED         RECORD-AREA-LENGTH

  >0088      2   NBS    0   BINARY SIGNED         RETES-RECEIVED

  >008A     80   ANS    0   ALPHANUMERIC          OFILRCRD

  >00DA      2   NBS    0   BINARY SIGNED         ALLDONE

  >00DC    440   GRP    0   GROUP                 SORT-CONTROL-BLOCK
  >00DC     44   GRP    0   GROUP                  HEADER
  >00DC      5   ANS    0   ALPHANUMERIC            SEQ
  >00E2      6   ABS    0   ALPHABETIC              SORT-TYPE
  >00E8      5   NSU    0   NUMERIC UNSIGNED        MAX-TOT-CONTL-LEN
  >00ED      1   ABS    0   ALPHABETIC              ASCND-DSCND
  >00F5      1   ANS    0   ALPHANUMERIC            COLLATNG-SEQ
  >00F6      1   ANS    0   ALPHANUMERIC            PRINT-OPTION
  >00F7      1   ANS    0   ALPHANUMERIC            OUTPUT-OPTION
  >00F8      4   ANS    0   ALPHANUMERIC            OUTPUT-REC-LEN
  >00FC      2   ANS    0   ALPHANUMERIC            VERIFY-OPTN
  >00FE      5   ANS    0   ALPHANUMERIC            WRK-SPACE
```

Figure 9-3. COBOL Routine Calling Sort/Merge (Sheet 5 of 7)

```
>0108     44   GRP     0   GROUP                OUT-FILE-SPEC
>0108      5   ANS     0   ALPHANUMERIC          SEQ
>010E      1   ABS     0   ALPHABETIC            FILE-USE
>010F      1   ABS     0   ALPHABETIC            FILE-TYPE
>0110     36   ANS     0   ALPHANUMERIC          PATHNAME
>0134     44   GRP     0   GROUP                CNT-OUT-FILE-SPEC
>0134      5   ANS     0   ALPHANUMERIC          SEQ
>013A      1   ABS     0   ALPHABETIC            FILE-USE
>013B      4   NSU     0   NUMERIC UNSIGNED      LOG-REC-SIZ
>013F      4   NSU     0   NUMERIC UNSIGNED      PHY-REC-SIZ
>0143      8   ANS     0   ALPHANUMERIC          NUM-PHY-REC
>0160     44   GRP     0   GROUP                WRK-FILE-SPEC
>0160      5   ANS     0   ALPHANUMERIC          SEQ
>0166      1   ABS     0   ALPHABETIC            FILE-USE
>0167      1   ANS     0   ALPHANUMERIC          EXPAND-ALLOC-FLG
>0168      8   ANS     0   ALPHANUMERIC          VOLUME
>018C     44   GRP     0   GROUP                INPT-FILE-CONTIN
>018C      5   ANS     0   ALPHANUMERIC          SEQ
>0192      1   ABS     0   ALPHABETIC            FILE-USE
>0193      1   ABS     0   ALPHABETIC            FILE-TYPE
>0194     36   ANS     0   ALPHANUMERIC          PATHNAME
>01B8     44   GRP     0   GROUP                INPT-FILE-CONTIN
>01B8      5   ANS     0   ALPHANUMERIC          SEQ
>01BE      1   ABS     0   ALPHABETIC            FILE-USE
```

```
DXCBL          L.R.V YY.DDD COMPILED:MM/DD/YY HH:MM:SS  OPT=M      PAGE    7
ADDRESS  SIZE  DEBUG ORDER TYPE                    NAME
 >01BF     4   ANS     0   ALPHANUMERIC           LOG-SIZE
 >01C7     8   ANS     0   ALPHANUMERIC           NUM-SRT-RECS
 >01E4    44   GRP     0   GROUP                 REFORMAT-DESCRIPTION-0
 >01E4     5   ANS     0   ALPHANUMERIC           SEQ
 >01EA     1   ANS     0   ALPHANUMERIC           FIELD-TYPE-CMMT
 >01EB     1   ABS     0   ALPHABETIC             CHARACTR-USE
 >01EC     8   GRP     0   GROUP                  FIELD-LOC
 >01EC     4   ANS     0   ALPHANUMERIC            BEG-RECRD-POS
 >01F0     4   ANS     0   ALPHANUMERIC            END-RECRD-POS
 >01F4     1   ANS     0   ALPHANUMERIC           CONDTN-FORCD-CHAR
 >01F5     1   ANS     0   ALPHANUMERIC           FORCD-CHAR
 >01F6     1   ANS     0   ALPHANUMERIC           CONTIN-LIN
 >01F7     3   ANS     0   ALPHANUMERIC           OUFLW-FLD-LEN
 >0210    44   GRP     0   GROUP                 REFORMAT-DESCRIPTION
 >0210     5   ANS     0   ALPHANUMERIC           SEQ
 >0216     1   ANS     0   ALPHANUMERIC           FIELD-TYPE-CMMT
 >0217     1   ABS     0   ALPHABETIC              CHARACTR-USE
 >0218     8   GRP     0   GROUP                  FIELD-LOC
 >0218     4   ANS     0   ALPHANUMERIC            BEG-RECRD-POS
 >021C     4   ANS     0   ALPHANUMERIC            END-RECRD-POS
 >0220     1   ANS     0   ALPHANUMERIC           CONDTN-FORCD-CHAR
 >0221     1   ANS     0   ALPHANUMERIC           FORCD-CHAR
```

Figure 9-3. COBOL Routine Calling Sort/Merge (Sheet 6 of 7)

```
  >0222      1   ANS    0   ALPHANUMERIC          CONTIN-LIN
  >0223      3   ANS    0   ALPHANUMERIC          OUFLW-FLD-LEN
  >023C     44   GRP    0   GROUP                REFORMAT-DESCRIPTION-3
  >023C      5   ANS    0   ALPHANUMERIC          SEQ
  >0242      1   ANS    0   ALPHANUMERIC          FIELD-TYPE-CMMT
  >0243      1   ABS    0   ALPHABETIC            CHARACTR-USE
  >0244      8   GRP    0   GROUP                 FIELD-LOC
  >0244      4   ANS    0   ALPHANUMERIC           BEG-RECRD-POS
  >0248      4   ANS    0   ALPHANUMERIC           END-RECRD-POS
  >024C      1   ANS    0   ALPHANUMERIC          CONDTN-FORCD-CHAR
  >024D      1   ANS    0   ALPHANUMERIC          FORCD-CHAR
  >024E      1   ANS    0   ALPHANUMERIC          CONTIN-LIN
  >024F      3   ANS    0   ALPHANUMERIC          OUFLW-FLD-LEN
  >0268     44   ANS    0   ALPHANUMERIC         ENDKRD

READ ONLY BYTE SIZE =        >015A

READ/WRITE BYTE SIZE =       >02EE

OVERLAY SEGMENT BYTE SIZE =  >0000

TOTAL BYTE SIZE =            >0048

    0 ERRORS

    0 WARNINGS



 DXCBL          L.R.V YY.DDD COMPILED:MM/DD/YY HH:MM:SS  OPT=M      PAGE    8
 PROGRAM   USING  COUNT

  SENREC           3
  SMSTAT           1
  SRTINT           3
```

Figure 9-3. COBOL Routine Calling Sort/Merge (Sheet 7 of 7)

The library S$SMRG.SMLIB contains the required Sort/Merge interface modules. The modules and their functions are as follows:

Module	Function
SRTINT	Performs the initialization of Sort/Merge before records can be sent to or received from the Sort/Merge module
SENREC	Transmits records from calling module to Sort/Merge
RCVREC	Transmits records from Sort/Merge to calling module
SMSTAT	Suspends calling tasks until Sort/Merge completes writing records to output file
COBINT	Contains other modules called by one of the above
IPCBUF	Contains buffer for IPC communication

Refer to the *DX Sort/Merge User's Guide* for a detailed description of these functions, their CALL statement syntax, and conditions under which each is required. The following link control file shows how to link the COBOL module shown in Figure 9-3.

```
FORMAT IMAGE,REPLACE
PROC RCOBOL
DUMMY
INCLUDE .S$$SYSLIB.RCBPRC
TASK CPNP
INCLUDE .S$$SYSLIB.RCBTSK
INCLUDE .S$$SYSLIB.RCBMPD
INCLUDE EX.CPNP
INCLUDE .S$$SMRG.SMLIB.SRTINT
INCLUDE .S$$SMRG.SMLIB.SENREC
INCLUDE .S$$SMRG.SMLIB.SMSTAT
INCLUDE .S$$SMRG.SMLIB.COBINT
INCLUDE .S$$SMRG.SMLIB.IPCBUF
END
```

9.4 DATABASE MANAGEMENT SYSTEM

The Database Management System (DBMS-990) is designed for minicomputer database applications. DBMS-990 handles data access in a logical format similar to physical documents and records in daily business transactions. DBMS-990 allows the user to define and access a centralized, integrated data base without the physical data access requirements imposed by conventional file management software. Considerations such as access method, record size, blocking, and relative field positions are resolved when the database is initially defined. Thus the user can concentrate fully on the logical data structure needed for interface.

9.4.1 DBMS-990 Features

Because the data definitions are independent from the application software, the data base can be changed without affecting existing programs. DBMS-990 also provides a single, centralized copy of the data to be used for all application subsystems. (Conventional file management results in fragmented and/or multiple copies of data, one for each application.) A centralized copy results in more efficient data storage on disk, uniform processing of data requests, and simplified data base maintenance. DBMS-990 optionally includes logging and access control.

Security is an optional feature of DBMS-990. Its purpose is to eliminate unauthorized use of the data base. Password security is provided to control file access. Access authorization is provided to define the type of access allowed to the data elements of a file for a particular password and/or user. Each file that requires a password also requires access authorization. For detailed information about DBMS-990, refer to the *DX10 Data Base Management System Programmer's Guide.*

9.4.2 DBMS-990 User Interface

The primary user interface to DBMS-990 consists of the data manipulation language (DML) and the data definition language (DDL). DML provides a means to manipulate data base information by supporting the reading and/or writing of the information. DBMS-990 data can be accessed by embedding the appropriate DML syntax in a COBOL application program module. (Refer to Figure 9-4). The application program module is used to construct a call to DBMS-990 that specifies the function to be performed on the data. The Data Base Manager processes the request and returns the results to the COBOL module. DDL allows the user to describe the DBMS-990 data base and the associated data elements. The definition source for the DDL logical data base is compiled by the DDL compiler; the output is stored on disk with the associated data. (Refer to Figure 9-5).

9.4.3 Linking DBMS-990 and COBOL Modules

The library S$DBMS contains the required DBMS-990 interface modules. The following link control file may be used to link the COBOL module:

```
    FORMAT IMAGE,REPLACE
    PROC RCOBOL
    DUMMY
    INCLUDE .S$$SYLIB.RCBPRC
    TASK GENEO
    INCLUDE .S$$SYSLIB.RCBTSK
    INCLUDE .S$$SYSLIB.RCBMPD
    INCLUDE EX.GENEO
    INCLUDE S$$DBMS.SNDMSG
    INCLUDE S$$DBMS.COBINT
    END

 DXCBL          L.R.V YY.DDD COMPILED:MM/DD/YY HH:MM:SS  OPT=M      PAGE    1

SOURCE ACCESS NAME:     MANUAL.PG.SRC.FIG0904
OBJECT ACCESS NAME:     DUMY
LISTING ACCESS NAME:    MANUAL.PG.LST.FIG0904
OPTIONS:                M
PRINT WIDTH:            80
PAGE SIZE:              55
PROGRAM SIZE (LINES):   1000
```

Figure 9-4. COBOL Interfacing With DBMS-990 (Sheet 1 of 14)

```
 DXCBL          L.R.V YY.DDD COMPILED:MM/DD/YY HH:MM:SS  OPT=M      PAGE    2
LINE  DEBUG PG/LN   A...B.....................................................
   1                IDENTIFICATION DIVISION.
   2                PROGRAM-ID. GENEALOGY.
   3               *     THIS PROGRAM WAS DEVELOPED AS A FUNCTIONAL
   4               *     DEMONSTRATION TEST FOR TESTING THE DATA BASE
   5               *     MANAGEMENT SYSTEM.
   6                ENVIRONMENT DIVISION.
   7                CONFIGURATION SECTION.
   8                SOURCE-COMPUTER. TI-990.
   9                OBJECT-COMPUTER. TI-990.
  10                DATA DIVISION.
  11                WORKING-STORAGE SECTION.
  12                01   ERR-FLG          PIC 99    VALUE 0.
  13                     88 ERR                      VALUES 1 THRU 99.
  14                     88 NO-ERR                   VALUE  0.
  15                01   PERSONS          PIC X.
  16                     88 PERSON                   VALUE "Y".
  17                     88 NO-PERSON                VALUE "N".
  18                01   SPOUSES          PIC X.
  19                     88 SPOUSE                   VALUE "Y".
  20                     88 NO-SPOUSE                VALUE "N".
  21                01   CHILDREN         PIC X.
  22                     88 CHILD                    VALUE "Y".
  23                     88 NO-CHILD                 VALUE "N".
  24                01   ACTIVITY         PIC X.
  25                     88 ACT-ADD                  VALUE "A".
  26                     88 ACT-UPDTE                VALUE "U".
  27                     88 ACT-DELTE                VALUE "D".
  28                     88 QUIT                     VALUE "Q".
  29                01   ACTION           PIC X.
  30                01   ANSWER           PIC X.
  31                01   PSC-TYPE         PIC X.
  32                     88 PSC-PERSON               VALUE "P".
  33                     88 PSC-SPOUSE               VALUE "S".
  34                     88 PSC-CHILD                VALUE "C".
  35                     88 NO-PSC                   VALUE " ".
  36                01   TEMP-NAME        PIC X(20) VALUE " ".
  37                01   FUNC-LIST.
  38                     02 PASSWORD       PIC X(4)  VALUE " ".
  39                     02 FUNCTION       PIC XX    VALUE "OF".
  40                     02 FILE-STAT      PIC XX    VALUE "**".
  41                     02 FILE-NAME      PIC X(4)  VALUE "GENE".
  42                     02 LOC1           PIC X(4)  VALUE "****".
  43                        88 EOL                   VALUE "****".
  44                     02 LOC2           PIC X(4)  VALUE "****".
  45                     02 KEY-NAME       PIC X(4)  VALUE "NAME".
  46                     02 KEY-VALUE      PIC X(30).
```

Figure 9-4. COBOL Interfacing With DBMS-990 (Sheet 2 of 14)

```
 47                01   LINE1-LIST.
 48                     02 FILLER        PIC X(7)  VALUE "LINE=01".
 49                     02 TST-1         PIC X     VALUE "*".
 50                     02 FILLER        PIC X(16) VALUE "PERSPSEXPDOBPPOB".
 51                     02 FILLER        PIC X(16) VALUE "MARDFATHMOTH****".
 52                     02 HR-1          PIC X(4)  VALUE "RLSE".
 53                01   LINE2-LIST.
 54                     02 FILLER        PIC X(7)  VALUE "LINE=02".
 55                     02 TST-2         PIC X     VALUE ",".
 56                     02 FILLER        PIC X(16) VALUE "SPOUSSEXSDOBSPOB".


 DXCBL          L.R.V YY.DDD COMPILED:MM/DD/YY HH:MM:SS  OPT=M      PAGE    3
LINE  DEBUG PG/LN   A...B.....................................................
 57                     02 FILLER        PIC X(4)  VALUE "****".
 58                     02 HR-2          PIC X(4)  VALUE "RLSE".
 59                01   LINE3-LIST.
 60                     02 FILLER        PIC X(7)  VALUE "LINE=03".
 61                     02 TST-3         PIC X     VALUE ",".
 62                     02 FILLER        PIC X(16) VALUE "CHLDCSEXCDOBCPOB".
 63                     02 FILLER        PIC X(4)  VALUE "****".
 64                     02 HR-3          PIC X(4)  VALUE "RLSE".
 65                01   LINE1-DATA.
 66                     02 PERSON-NAME   PIC X(30) VALUE SPACES.
 67                     02 PERSON-SEX    PIC X     VALUE SPACES.
 68                     02 DATE-OF-BIRTH.
 69                        03 PERSON-DOB-MO     PIC XX   VALUE  SPACES.
 70                        03 PERSON-DOB-DA     PIC XX   VALUE  SPACES.
 71                        03 PERSON-DOB-YR     PIC XX   VALUE  SPACES.
 72                     02 PLACE-OF-BIRTH.
 73                        03 PERSON-POB-STATE  PIC XXX  VALUE  SPACES.
 74                     02 MARITAL-STAT  PIC X     VALUE SPACES.
 75                     02 FATHER        PIC X(30) VALUE SPACES.
 76                     02 MOTHER        PIC X(30) VALUE SPACES.
 77                01   LINE2-DATA.
 78                     02 SPOUSE-NAME   PIC X(30).
 79                     02 SPOUSE-SEX    PIC X.
 80                     02 DATE-OF-BIRTH.
 81                        03 SPOUSE-DOB-MO     PIC XX   VALUE  SPACES.
 82                        03 SPOUSE-DOB-DA     PIC XX   VALUE  SPACES.
 83                        03 SPOUSE-DOB-YR     PIC X(4) VALUE  SPACES.
 84                     02 PLACE-OF-BIRTH.
 85                        03 SPOUSE-POB-STATE  PIC XXX  VALUE  SPACES.
```

Figure 9-4. COBOL Interfacing With DBMS-990 (Sheet 3 of 14)

```
 86                01   LINE3-DATA.
 87                     02 CHILD-NAME     PIC X(30).
 88                     02 CHILD-SEX      PIC X.
 89                     02 DATE-OF-BIRTH.
 90                        03 CHILD-DOB-MO      PIC XX   VALUE  SPACES.
 91                        03 CHILD-DOB-DA      PIC XX   VALUE  SPACES.
 92                        03 CHILD-DOB-YR      PIC X(4) VALUE  SPACES.
 93                     02 PLACE-OF-BIRTH.
 94                        03 CHILD-POB-STATE   PIC XXX  VALUE  SPACES.
 95                01   DELIM             PIC XX     VALUE "/*".


 DXCBL          L.R.V YY.DDD COMPILED:MM/DD/YY HH:MM:SS  OPT=M      PAGE    4
LINE  DEBUG PG/LN   A...B.....................................................
 96                /
 97                 PROCEDURE DIVISION.
 98  >0000          MAIN SECTION.
 99  >0000          MAIN-PROG.
100  >0000               DISPLAY "ENTER PASSWORD" LINE 1 ERASE.
101  >0008               ACCEPT PASSWORD PROMPT.
102  >000E               DISPLAY " " LINE 1 ERASE.
103  >0016               CALL "DBMSYS" USING FUNC-LIST LINE1-LIST
104                         DELIM DELIM, DELIM DELIM.
105  >0018               IF FILE-STAT NOT = "**"
106                         ADD 1 TO ERR-FLG
107                         DISPLAY "OPEN ERR " LINE 12
108                         FILE-STAT LINE 1 POSITION 18.
109  >0032               IF NO-ERR
110                         PERFORM ACTIVITY UNTIL QUIT.
111  >0042               MOVE "CL" TO FUNCTION.
112  >0046               CALL "DBMSYS" USING FUNC-LIST FUNC-LIST
113                         DELIM DELIM, DELIM DELIM.
114  >0048               STOP RUN.
115
116  >004A          ACTIVITY SECTION.
117  >004A          BEGIN.
118  >004A               DISPLAY "FUNCTION: ADD, UPDTE, DELTE, QUIT - A,U,D,Q"
119                         LINE 1 ERASE.
120  >0052               ACCEPT ACTIVITY LINE 1 POSITION 45 PROMPT.
121  >005C               MOVE SPACE TO SPOUSES CHILDREN.
122  >0062               IF ACT-ADD   PERFORM ADD-SEC   UNTIL NO-PSC.
123  >0072               IF ACT-UPDTE PERFORM UPDTE-SEC UNTIL NO-PSC.
124  >0082               IF ACT-DELTE PERFORM DELTE-SEC UNTIL NO-PSC.
125  >0092               MOVE 0 TO ERR-FLG PSC-TYPE.
126
```

Figure 9-4. COBOL Interfacing With DBMS-990 (Sheet 4 of 14)

```
127  >009C          ADD-SEC SECTION.
128  >009C          BEGIN.
129  >009C               MOVE "****" TO LOC1 LOC2.
130  >00A2               MOVE "RLSE" TO HR-1 HR-2 HR-3.
131  >00AA               DISPLAY "ADD PERSON, SPOUSE, CHILD - P,S,C"
132                         LINE 1 ERASE.
133  >00B2               ACCEPT PSC-TYPE LINE 1 POSITION 45 PROMPT.
134  >00BC               IF PSC-PERSON PERFORM ADD-PERSON.
135  >00C4               IF PSC-SPOUSE PERFORM POSITION-SPOUSE
136                         IF NO-ERR  PERFORM ADD-SPOUSE.
137  >00D4               IF PSC-CHILD  PERFORM POSITION-CHILD
138                         IF NO-ERR  PERFORM ADD-CHILD.
139  >00E4               MOVE 0 TO ERR-FLG.
140
141  >00EC          UPDTE-SEC SECTION.
142  >00EC          BEGIN.
143  >00EC               MOVE "****" TO LOC1 LOC2.
144  >00F2               MOVE "HOLD" TO HR-1 HR-2 HR-3.
145  >00FA               DISPLAY "UPDATE PERSON, SPOUSE, CHILD - P,S,C"
146                         LINE 1 ERASE.
147  >0102               ACCEPT PSC-TYPE LINE 1 POSITION 45 PROMPT.
148  >010C               IF PSC-PERSON PERFORM UPDTE-PERSON.
149  >0114               IF PSC-SPOUSE PERFORM UPDTE-SPOUSE.
150  >001C               IF PSC-CHILD  PERFORM UPDTE-CHILD.
151  >0124               MOVE 0 TO ERR-FLG.


 DXCBL          L.R.V YY.DDD COMPILED:MM/DD/YY HH:MM:SS  OPT=M      PAGE    5
LINE  DEBUG PG/LN   A...B.....................................................
152  >0128               MOVE "NAME" TO KEY-NAME.
153
154  >0130          DELTE-SEC SECTION.
155  >0130          BEGIN.
156  >0130               MOVE "****" TO LOC1 LOC2.
157  >0136               MOVE "HOLD" TO HR-1 HR-2 HR-3.
158  >013E               DISPLAY "DELETE PERSON, SPOUSE, CHILD - P,S,C"
159                         LINE 1 ERASE.
160  >0146               ACCEPT PSC-TYPE LINE 1 POSITION 45.
161  >014E               IF PSC-PERSON PERFORM DELTE-PERSON.
162  >0156               IF PSC-SPOUSE PERFORM POSITION-SPOUSE
163                         IF NO-ERR  PERFORM DELTE-SPOUSE.
164  >0166               IF PSC-CHILD  PERFORM POSITION-CHILD
165                         IF NO-ERR  PERFORM DELTE-CHILD.
166  >0176               MOVE 0 TO ERR-FLG.
167  >017A               MOVE "NAME" TO KEY-NAME.
168
```

Figure 9-4. COBOL Interfacing With DBMS-990 (Sheet 5 of 14)

```
169  >0182          MISC SECTION.
170  >0182          ADD-PERSON.
171  >0182               MOVE SPACES TO LINE1-DATA.
172  >0186               PERFORM DISPLAY-LINE1-FORMAT THRU ACCEPT-LINE1-DATA.
173  >0188               MOVE PERSON-NAME TO KEY-VALUE.
174  >018C               MOVE "AA" TO FUNCTION.
175  >0190               PERFORM ACCESS-LINE1.
176  >0192               IF ERR
177                         DISPLAY "ERROR ADDING PERSON LINE 01 " LINE 22,
178                         FILE-STAT LINE 22 POSITION 35,
179                         ACCEPT ANSWER LINE 22 POSITION 40 PROMPT.
180  >01B6               IF MARITAL-STAT NOT = "S"
181                         IF NO-ERR
182                            PERFORM ADD-SPOUSE UNTIL ERR OR NO-SPOUSE.
183
184  >01DA          ADD-SPOUSE.
185  >01DA              MOVE SPACES TO LINE2-DATA.
186  >01DE              PERFORM DISPLAY-LINE2-FORMAT THRU ACCEPT-LINE2-DATA.
187  >01E0              IF SPOUSE
188                        MOVE "AA" TO FUNCTION
189                        PERFORM ACCESS-LINE2
190                        IF ERR
191                           DISPLAY "ERROR ADDING SPOUSE LINE 02 " LINE 22,
192                           FILE-STAT LINE 22 POSITION 35,
193                           ACCEPT ANSWER LINE 22 POSITION 40 PROMPT
194                        ELSE
195                           PERFORM ADD-CHILD UNTIL ERR OR NO-CHILD.
196
197  >022A          ADD-CHILD.
198  >022A                 MOVE SPACES TO LINE3-DATA CHILDREN.
199  >0230                 PERFORM DISPLAY-LINE3-FORMAT THRU ACCEPT-LINE3-DATA.
200  >0232                 IF CHILD
201                           MOVE "AA" TO FUNCTION
202                           PERFORM ACCESS-LINE3
203                           IF ERR
204                             DISPLAY "ERROR ADDING CHILD LINE 03 " LINE 22,
205                             FILE-STAT LINE 22 POSITION 35,
206                             ACCEPT ANSWER LINE 22 POSITION 40 PROMPT.
207


 DXCBL          L.R.V YY.DDD COMPILED:MM/DD/YY HH:MM:SS  OPT=M      PAGE    6
LINE  DEBUG PG/LN   A...B.................................................
208  >0264          POSITION-PERSON.
209  >0264               MOVE "RF" TO FUNCTION.
210  >0268               DISPLAY "POSITION ON PERSON: " LINE 4.
211  >026E               ACCEPT KEY-VALUE LINE 4 POSITION 22 PROMPT.
212  >0278               PERFORM ACCESS-LINE1.
213
```

Figure 9-4. COBOL Interfacing With DBMS-990 (Sheet 6 of 14)

```
214  >027C          POSITION-SPOUSE.
215  >027C               PERFORM POSITION-PERSON.
216  >027E               IF NO-ERR
217                         DISPLAY "POSITION ON SPOUSE: " LINE 4
218                         ACCEPT TEMP-NAME LINE 4 POSITION 22 PROMPT.
219                         IF TEMP-NAME NOT = " "
220                            PERFORM ACCESS-LINE2 UNTIL ERR OR EOL OR
221                               TEMP-NAME = SPOUSE-NAME.
222
223  >02B8          POSITION-CHILD.
224  >02B8               PERFORM POSITION-SPOUSE.
225  >02BA               IF NO-ERR
226                         DISPLAY "POSITION ON CHILD: " LINE 4
227                         ACCEPT TEMP-NAME LINE 4 POSITION 22 PROMPT
228                         IF TEMP-NAME NOT = " "
229                            PERFORM ACCESS-LINE3 UNTIL ERR OR EOL OR
230                               TEMP-NAME = CHILD-NAME.
231
232  >02F4          UPDTE-PERSON.
233  >02F4               DISPLAY "PERSON'S FULL NAME: " LINE 3 ERASE.
234  >02FC               ACCEPT KEY-VALUE LINE 3 POSITION 22 PROMPT.
235  >0306               MOVE "RF" TO FUNCTION.
236  >030A               PERFORM ACCESS-LINE1.
237  >030C               IF ERR
238                         DISPLAY "ERROR READING PERSON LINE 01 " LINE 24
239                         FILE-STAT LINE 24 POSITION 35
240                         ACCEPT ANSWER LINE 24 POSITION 40 PROMPT
241                      ELSE
242                         PERFORM DISPLAY-LINE1-FORMAT
243                         PERFORM DISPLAY-LINE1-DATA
244                         PERFORM ACCEPT-LINE1-DATA
245                         MOVE "WT" TO FUNCTION
246                         PERFORM ACCESS-LINE1
247                         IF ERR
248                           DISPLAY "ERROR UPDATING PERSON LINE 01 " LINE 24
249                           FILE-STAT LINE 24 POSITION 35
250                           ACCEPT ANSWER LINE 24 POSITION 40 PROMPT.
251
252  >0364          UPDTE-SPOUSE.
253  >0364               DISPLAY "SPOUSE'S FULL NAME: " LINE 3 ERASE.
254  >036C               ACCEPT KEY-VALUE LINE 3 POSITION 22 PROMPT.
255  >0376               MOVE "RF" TO FUNCTION.
256  >037A               MOVE "SPOU" TO KEY-NAME.
257  >037E               PERFORM ACCESS-LINE2.
258  >0380               IF ERR OR EOL
259                         DISPLAY "ERROR READING SPOUSE LINE 02 " LINE 24
260                         FILE-STAT LINE 24 POSITION 35
261                         ACCEPT ANSWER LINE 24 POSITION 40 PROMPT
262                      ELSE
263                         PERFORM DISPLAY-LINE2-FORMAT
```

Figure 9-4. COBOL Interfacing With DBMS-990 (Sheet 7 of 14)

```
 DXCBL          L.R.V YY.DDD COMPILED:MM/DD/YY HH:MM:SS  OPT=M      PAGE   7
LINE  DEBUG PG/LN   A...B.....................................................
 264                          PERFORM DISPLAY-LINE2-DATA
 265                          PERFORM ACCEPT-LINE2-DATA
 266                          MOVE "WT" TO FUNCTION
 267                          PERFORM ACCESS-LINE2
 268                          IF ERR
 269                            DISPLAY "ERROR UPDATING SPOUSE LINE 03 " LINE 24
 270                            FILE-STAT LINE 24 POSITION 35
 271                            ACCEPT ANSWER LINE 24 POSITION 40 PROMPT.
 272
 273  >03E9          UPDTE-CHILD.
 274  >03E9               DISPLAY "CHILD'S FULL NAME: " LINE 3 ERASE.
 275  >03F2               ACCEPT KEY-VALUE LINE 3 POSITION 22 PROMPT.
 276  >03FC               MOVE "RF" TO FUNCTION.
 277  >0400               MOVE "CHLD" TO KEY-NAME.
 278  >0405               PERFORM ACCESS-LINE3.
 279  >0407               IF ERR OR EOL
 280                            DISPLAY "ERROR READING CHILD LINE 03 " LINE 24,
 281                            FILE-STAT LINE 24 POSITION 35,
 282                            ACCEPT ANSWER LINE 24 POSITION 40 PROMPT
 283                         ELSE
 284                            PERFORM DISPLAY-LINE3-FORMAT
 285                            PERFORM DISPLAY-LINE3-DATA
 286                            PERFORM ACCEPT-LINE3-DATA
 287                            MOVE "WT" TO FUNCTION
 288                            PERFORM ACCESS-LINE3
 289                            IF ERR
 290                              DISPLAY "ERROR UPDATING CHILD LINE 03 " LINE 24,
 291                              FILE-STAT LINE 24 POSITION 35,
 292                              ACCEPT ANSWER LINE 24 POSITION 40 PROMPT.
 293
 294  >0470          DELTE-PERSON.
 295  >0470               DISPLAY "PERSON'S FULL NAME: " LINE 3 ERASE.
 296  >0478               ACCEPT KEY-VALUE LINE 3 POSITION 22 PROMPT.
 297  >0482               MOVE "DR" TO FUNCTION.
 298  >0487               PERFORM ACCESS-LINE1.
 299  >0489               IF ERR
 300                            DISPLAY "ERROR READING SPOUSE LINE 02 " LINE 24,
 301                            FILE-STAT LINE 24 POSITION 35,
 302                            ACCEPT ANSWER LINE 24 POSITION 40 PROMPT.
 303
 304  >04B2          DELTE-SPOUSE.
 305  >04B2               IF ERR
 306                            DISPLAY "ERROR READING SPOUSE LINE 02 " LINE 24,
 307                            FILE-STAT LINE 24 POSITION 35,
 308                            ACCEPT ANSWER LINE 24 POSITION 40 PROMPT
```

Figure 9-4. COBOL Interfacing With DBMS-990 (Sheet 8 of 14)

```
309                     ELSE
310                        MOVE "DL" TO FUNCTION
311                        PERFORM ACCESS-LINE2
312                        IF ERR
313                          DISPLAY "ERROR DELETING SPOUSE LINE 03 " LINE 24
314                          FILE-STAT LINE 24 POSITION 35,
315                          ACCEPT ANSWER LINE 24 POSITION 40 PROMPT
316                        ELSE
317                           MOVE " " TO LOC1 LOC2
318                           PERFORM DELTE-CHILDREN UNTIL EOL OR ERR.
319
```

```
 DXCBL          L.R.V YY.DDD COMPILED:MM/DD/YY HH:MM:SS  OPT=M      PAGE    8
LINE  DEBUG PG/LN   A...B.....................................................
 320  >0530         DELTE-CHILD.
 321  >0530              IF ERR
 322                       DISPLAY "ERROR READING CHILD LINE 03 " LINE 24,
 323                       FILE-STAT LINE 24 POSITION 35,
 324                       ACCEPT ANSWER LINE 24 POSITION 40 PROMPT
 325                   ELSE
 326                       MOVE "DL" TO FUNCTION
 327                       PERFORM ACCESS-LINE3
 328                       IF ERR
 329                         DISPLAY "ERROR DELETING CHILD LINE 03 " LINE 24,
 330                         FILE-STAT LINE 24 POSITION 35,
 331                         ACCEPT ANSWER LINE 24 POSITION 40 PROMPT.
 332
 333  >058A         DELTE-CHILDREN.
 334  >058A              MOVE "RF" TO FUNCTION.
 335  >058E              PERFORM ACCESS-LINE3.
 336  >0590              IF NO-ERR
 337                       MOVE "DL" TO FUNCTION
 338                       PERFORM ACCESS-LINE3.
 339
 340  >05A2         DBMS-ACCESS SECTION.
 341  >05A2         ACCESS-LINE1.
 342  >05A2              CALL "DBMSYS" USING FUNC-LIST LINE1-LIST
 343                       LINE1-LIST LINE2-LIST LINE1-DATA LINE2-DATA.
 344  >05A5              IF FILE-STAT NOT = "**" ADD 1 TO ERR-FLG.
 345
 346  >05B4         ACCESS-LINE2.
 347  >05B4              CALL "DBMSYS" USING FUNC-LIST LINE1-LIST
 348                       LINE2-LIST LINE3-LIST LINE2-DATA LINE3-DATA.
 349  >05B7              IF FILE-STAT NOT = "**" ADD 1 TO ERR-FLG.
 350
 351  >05C6         ACCESS-LINE3.
 352  >05C6               CALL "DBMSYS" USING FUNC-LIST LINE1-LIST
 353                          LINE3-LIST LINE1-DATA LINE3-DATA DELIM.
 354  >05C9               IF FILE-STAT NOT = "**" ADD 1 TO ERR-FLG.
```

Figure 9-4. COBOL Interfacing With DBMS-990 (Sheet 9 of 14)

```
355
356  >05DA          DISPLAY-LINE1-FORMAT SECTION.
357  >05DA          BEGIN.
358  >05DA             DISPLAY "PERSON'S FULL NAME:" LINE 3  POSITION 1 ERASE
359                           "SEX: "               LINE 4  POSITION 1
360                           "DATE OF BIRTH: "     LINE 5  POSITION 1
361                           "   MONTH: "          LINE 6  POSITION 1
362                           "   DAY:   "          LINE 7  POSITION 1
363                           "   YEAR:  "          LINE 8  POSITION 1
364                           "PLACE OF BIRTH: "    LINE 9  POSITION 1
365                           "   STATE/COUNTRY:"   LINE 10 POSITION 1
366                           "MARITAL STATUS: "    LINE 11 POSITION 1
367                            "FATHER: "           LINE 12 POSITION 1
368                            "MOTHER: "           LINE 13 POSITION 1.
369
370  >064B          ACCEPT-LINE1-DATA SECTION.
371  >064B          BEGIN.
372  >064B             ACCEPT PERSON-NAME           LINE 3 POSITION 22.
373  >0653             IF PERSON-NAME = " "
374                       MOVE "N" TO PERSONS
375                    ELSE


 DXCBL          L.R.V YY.DDD COMPILED:MM/DD/YY HH:MM:SS  OPT=M      PAGE    9
LINE  DEBUG PG/LN   A...B.....................................................
376                       MOVE "Y" TO PERSONS
377                       ACCEPT PERSON-SEX          LINE 4  POSITION 22
378                            PERSON-DOB-MO         LINE 6  POSITION 22
379                            PERSON-DOB-DA         LINE 7  POSITION 22
380                            PERSON-DOB-YR         LINE 8  POSITION 22
381                            PERSON-POB-STATE      LINE 10 POSITION 22
382                            MARITAL-STAT          LINE 11 POSITION 22
383                            FATHER                LINE 12 POSITION 22
384                            MOTHER                LINE 13 POSITION 22.
385
386  >06B8          DISPLAY-LINE1-DATA SECTION.
387  >06B8          BEGIN.
388  >06B8               DISPLAY PERSON-NAME          LINE 3  POSITION 22
389                             PERSON-SEX           LINE 4  POSITION 22
390                             PERSON-DOB-MO        LINE 6  POSITION 22
391                             PERSON-DOB-DA        LINE 7  POSITION 22
392                             PERSON-DOB-YR        LINE 8  POSITION 22
393                             PERSON-POB-STATE     LINE 10 POSITION 22
394                             MARITAL-STAT         LINE 11 POSITION 22
395                             FATHER               LINE 12 POSITION 22
396                             MOTHER               LINE 13 POSITION 22.
397
```

Figure 9-4. COBOL Interfacing With DBMS-990 (Sheet 10 of 14)

```
398  >070F          DISPLAY-LINE2-FORMAT.
399  >070F              DISPLAY "SPOUSE'S FULL NAME: " LINE 3 POSITION 1 ERASE
400                             "SEX: "                LINE 4  POSITION 1
401                             "DATE OF BIRTH: "      LINE 5  POSITION 1
402                             "   MONTH: "           LINE 6  POSITION 1
403                             "   DAY:   "           LINE 7  POSITION 1
404                             "   YEAR:  "           LINE 8  POSITION 1
405                             "PLACE OF BIRTH: "     LINE 9  POSITION 1
406                             "  STATE/COUNTRY: "    LINE 10 POSITION 1.
407
408  >0760          ACCEPT-LINE2-DATA.
409  >0760               ACCEPT SPOUSE-NAME           LINE 3 POSITION 22.
410  >0768               IF SPOUSE-NAME = " "
411                        MOVE "N" TO SPOUSES
412                     ELSE
413                        MOVE "Y" TO SPOUSES
414                        ACCEPT SPOUSE-SEX          LINE 4  POSITION 22
415                             SPOUSE-DOB-MO         LINE 6  POSITION 22
416                             SPOUSE-DOB-DA         LINE 7  POSITION 22
417                             SPOUSE-DOB-YR         LINE 8  POSITION 22
418                             SPOUSE-POB-STATE      LINE 10 POSITION 22.
419
420  >07AD          DISPLAY-LINE2-DATA.
421  >07AD               DISPLAY SPOUSE-NAME          LINE 3  POSITION 22
422                             SPOUSE-SEX            LINE 4  POSITION 22
423                             SPOUSE-DOB-MO         LINE 6  POSITION 22
424                             SPOUSE-DOB-DA         LINE 7  POSITION 22
425                             SPOUSE-DOB-YR         LINE 8  POSITION 22
426                             SPOUSE-POB-STATE      LINE 10 POSITION 22.
427
428  >07E8          DISPLAY-LINE3-FORMAT.
429  >07E8               DISPLAY "CHILD'S FULL NAME: " LINE 3 POSITION 1 ERASE
430                             "SEX: "               LINE 4  POSITION 1
431                             "DATE OF BIRTH: "     LINE 5  POSITION 1


 DXCBL          L.R.V YY.DDD COMPILED:MM/DD/YY HH:MM:SS  OPT=M      PAGE   10
LINE  DEBUG PG/LN   A...B.....................................................
 432                            "  MONTH: "           LINE 6  POSITION 1
 433                            "  DAY:   "           LINE 7  POSITION 1
 434                            "  YEAR:  "           LINE 8  POSITION 1
 435                            " PLACE OF BIRTH: "   LINE 9  POSITION 1
 436                            "  STATE/COUNTRY: "   LINE 10 POSITION 1.
 437
 438  >083A          ACCEPT-LINE3-DATA.
 439  >083A               ACCEPT CHILD-NAME            LINE 3  POSITION 22.
 440  >0842               IF CHILD-NAME = " "
 441                        MOVE "N" TO CHILDREN
```

Figure 9-4. COBOL Interfacing With DBMS-990 (Sheet 11 of 14)

```
442                     ELSE
443                        MOVE "Y" TO CHILDREN
444                        ACCEPT CHILD-SEX          LINE 4  POSITION 22
445                             CHILD-DOB-MO         LINE 6  POSITION 22
446                             CHILD-DOB-DA         LINE 7  POSITION 22
447                             CHILD-DOB-YR         LINE 8  POSITION 22
448                             CHILD-POB-STATE      LINE 10 POSITION 22.
449
450  >0887          DISPLAY-LINE3-DATA.
451  >0887               DISPLAY CHILD-NAME          LINE 3  POSITION 22
452                             CHILD-DOB-MO         LINE 6  POSITION 22
453                             CHILD-DOB-DA         LINE 7  POSITION 22
454                             CHILD-DOB-YR         LINE 8  POSITION 22
455                             CHILD-POB-STAT       LINE 10 POSITION 22.
456         ZZZZZZ END PROGRAM.                                  *** END OF FILE


DXCBL          L.R.V YY.DDD COMPILED:MM/DD/YY HH:MM:SS  OPT=M      PAGE   11
ADDRESS  SIZE  DEBUG ORDER TYPE                    NAME

 >002A      2   NSU     0   NUMERIC UNSIGNED      ERR-FLG
                        0   CONDITION-NAME         ERR
                        0   CONDITION-NAME         NO-ERR

 >002C      1   ANS     0   ALPHANUMERIC          PERSONS
                        0   CONDITION-NAME         PERSON
                        0   CONDITION-NAME         NO-PERSON

 >002E      1   ANS     0   ALPHANUMERIC          SPOUSES
                        0   CONDITION-NAME         SPOUSE
                        0   CONDITION-NAME         NO-SPOUSE

 >0030      1   ANS     0   ALPHANUMERIC          CHILDREN
                        0   CONDITION-NAME         CHILD
                        0   CONDITION-NAME         NO-CHILD

 >0032      1   ANS     0   ALPHANUMERIC          ACTIVITY
                        0   CONDITION-NAME         ACT-ADD
                        0   CONDITION-NAME         ACT-UPDTE
                        0   CONDITION-NAME         ACT-DELTE
                        0   CONDITION-NAME         QUIT

 >0034      1   ANS     0   ALPHANUMERIC          ACTION

 >0036      1   ANS     0   ALPHANUMERIC          ANSWER

 >0038      1   ANS     0   ALPHANUMERIC          PSC-TYPE
                        0   CONDITION-NAME         PSC-PERSON
                        0   CONDITION-NAME         PSC-SPOUSE
                        0   CONDITION-NAME         PSC-CHILD
                        0   CONDITION-NAME         NO-PSC
```

Figure 9-4. COBOL Interfacing With DBMS-990 (Sheet 12 of 14)

```
 >003A      20   ANS     0   ALPHANUMERIC          TEMP-NAME

 >004E      54   GRP     0   GROUP                 FUNC-LIST
 >004E       4   ANS     0   ALPHANUMERIC           PASSWORD
 >0052       2   ANS     0   ALPHANUMERIC           FUNCTION
 >0054       2   ANS     0   ALPHANUMERIC           FILE-STAT
 >0056       4   ANS     0   ALPHANUMERIC           FILE-NAME
 >005A       4   ANS     0   ALPHANUMERIC           LOC1
                     0   CONDITION-NAME          EOL
 >005E       4   ANS     0   ALPHANUMERIC           LOC2
 >0062       4   ANS     0   ALPHANUMERIC           KEY-NAME
 >0066      30   ANS     0   ALPHANUMERIC           KEY-VALUE

 >0084      44   GRP     0   GROUP                 LINE1-LIST
 >008B       1   ANS     0   ALPHANUMERIC           TST-1
 >00AC       4   ANS     0   ALPHANUMERIC           HR-1

 >00B0      32   GRP     0   GROUP                 LINE2-LIST
 >00B7       1   ANS     0   ALPHANUMERIC           TST-2
 >00CC       4   ANS     0   ALPHANUMERIC           HR-2

 >00D0      32   GRP     0   GROUP                 LINE3-LIST
 >00D7       1   ANS     0   ALPHANUMERIC           TST-3



DXCBL          L.R.V YY.DDD COMPILED:MM/DD/YY HH:MM:SS  OPT=M      PAGE   12
ADDRESS  SIZE  DEBUG ORDER TYPE                    NAME
 >00EC      4   ANS     0   ALPHANUMERIC           HR-3

 >00F0    103   GRP     0   GROUP                 LINE1-DATA
 >00F0     30   ANS     0   ALPHANUMERIC           PERSON-NAME
 >010E      1   ANS     0   ALPHANUMERIC           PERSON-SEX
 >010F      8   GRP     0   GROUP                  DATE-OF-BIRTH
 >010F      2   ANS     0   ALPHANUMERIC            PERSON-DOB-MO
 >0111      2   ANS     0   ALPHANUMERIC            PERSON-DOB-DA
 >0113      4   ANS     0   ALPHANUMERIC            PERSON-DOB-YR
 >0117      3   GRP     0   GROUP                  PLACE-OF-BIRTH
 >0117      3   ANS     0   ALPHANUMERIC            PERSON-POB-STATE
 >011A      1   ANS     0   ALPHANUMERIC           MARITAL-STAT
 >011B     30   ANS     0   ALPHANUMERIC           FATHER
 >0139     30   ANS     0   ALPHANUMERIC           MOTHER
```

Figure 9-4. COBOL Interfacing With DBMS-990 (Sheet 13 of 14)

```
  >0158     42   GRP    0   GROUP              LINE2-DATA
  >0158     30   ANS    0   ALPHANUMERIC        SPOUSE-NAME
  >0176      1   ANS    0   ALPHANUMERIC        SPOUSE-SEX
  >0177      8   GRP    0   GROUP               DATE-OF-BIRTH
  >0177      2   ANS    0   ALPHANUMERIC         SPOUSE-DOB-MO
  >0179      2   ANS    0   ALPHANUMERIC         SPOUSE-DOB-DA
  >017B      4   ANS    0   ALPHANUMERIC         SPOUSE-DOB-YR
  >017F      3   GRP    0   GROUP               PLACE-OF-BIRTH
  >017F      3   ANS    0   ALPHANUMERIC         SPOUSE-POB-STATE

  >0182     42   GRP    0   GROUP              LINE3-DATA
  >0182     30   ANS    0   ALPHANUMERIC        CHILD-NAME
  >01A0      1   ANS    0   ALPHANUMERIC        CHILD-SEX
  >01A1      8   GRP    0   GROUP               DATE-OF-BIRTH
  >01A1      2   ANS    0   ALPHANUMERIC         CHILD-DOB-MO
  >01A3      2   ANS    0   ALPHANUMERIC         CHILD-DOB-DA
  >01A5      4   ANS    0   ALPHANUMERIC         CHILD-DOB-YR
  >01A9      3   GRP    0   GROUP               PLACE-OF-BIRTH
  >01A9      3   ANS    0   ALPHANUMERIC         CHILD-POB-STATE

  >01AC      2   ANS    0   ALPHANUMERIC       DELIM

READ ONLY BYTE SIZE =        >1094

READ/WRITE BYTE SIZE =       >0208

OVERLAY SEGMENT BYTE SIZE =  >0000

TOTAL BYTE SIZE =            >129C

    0 ERRORS

    0 WARNINGS


 DXCBL          L.R.V YY.DDD COMPILED:MM/DD/YY HH:MM:SS  OPT=M      PAGE   13
 PROGRAM   USING  COUNT

  DBMSYS          6
```

Figure 9-4. COBOL Interfacing With DBMS-990 (Sheet 14 of 14)

```
FILE=GENE,LINES=600
ID=NAME=CH/30,VOL=100
LINE=01
FIELD=PERS=CH/30
FIELD=PSEX=CH/1
GROUP=PDOB
FIELD=PMO =CH/2
FIELD=PDA =CH/2
FIELD=PYR =CH/4
ENDG
FIELD=PPOB=CH/3
FIELD=MARD=CH/1
FIELD=FATH=CH/30
FIELD=MOTH=CH/30
ENDL
LINE=02
FIELD=SPOU=CH/30
FIELD=SSEX=CH/1
GROUP=SDOB
FIELD=SMO =CH/2
FIELD=SDA =CH/2
FIELD=SYR =CH/4
ENDG
FIELD=SPOB=CH/3
ENDL
LINE=03
FIELD=CHLD=CH/30
FIELD=CSEX=CH/1
GROUP=CDOB
FIELD=CMO =CH/2
FIELD=CDA =CH/2
FIELD=CYR =CH/4
ENDG
FIELD=CPOB=CH/3
ENDL
SECONDARY-REFERENCES
SPOU=VOL=100
SPOU=VOL=100
END.(FORMAT,SECL)
```

Figure 9-5. Data Definition Language (DDL) File

9.5 QUERY-990

The Query-990 software package provides a convenient and efficient means of retrieving data from a DBMS-990 database file. Query-990 enables you to gather, modify, and review data without writing a program.

The Query-990 language is an English-like nonprocedural language with statements composed of several clauses. The clauses allow you to specify the content and format of each line, as well as complex conditions that a database record or line must meet to be qualified for output. Totals, counts, or averages can be performed on output fields; default columnar headings and user-defined headings are supported.

When the Query-990 language is used, a complex report may be specified in a few lines, whereas an application program to obtain the same report can require several hundred lines. Refer to the *Query-990 User's Guide* for a detailed explanation of Query-990.

You can access Query-990 from COBOL programs through a set of assembly language subroutines that interface between the Query-990 processor and the application task. The following subroutines can be linked to the calling task.

- QCOMP — Compiles, loads, and prepares a Query-990 statement for execution. The Query-990 statement is passed from the application task as an array of characters.
- QINIT — Loads and prepares a Query-990 statement for execution that has already been compiled (using QCOMPILE) and stored as an object file.
- QEXEC — Executes a Query-990 statement started by QCOMP or QINIT and lists the results to an output file.
- QRECV — Processes one cycle of a Query-990 statement. For example, if the Query-990 is a list function, QRECV returns one logical report line.
- QSEND — Resets and sends change data values using the contents of the data buffer.
- QCLR — Reinitializes the Query-990 processor for a particular Query-990 statement (a clearing function).
- QEND — Terminates the Query-990 processor for a particular Query-990 statement.

These routines are contained on the library S$QUERY. The following link control file shows how to link the COBOL module in Figure 9-6.

```
    FORMAT IMAGE,REPLACE
    LIBRARY .SCI990.S$$OBJECT
    PROC RCOBOL
    DUMMY
    INCLUDE .S$$SYSLIB.RCBPRC
    TASK CTEST
    INCLUDE .S$$SYSLIB.RCBTSK
    INCLUDE .S$$SYSLIB.RCBMPD
    INCLUDE EX.CTEST
    INCLUDE S$$QUERY.COBINT
    INCLUDE S$$QUERY.PLIOBJ
    END
 DNCBL          L.R.V YY.DDD COMPILED:MM/DD/YY HH:MM:SS  OPT=M      PAGE    1

SOURCE ACCESS NAME:    MANUAL.PG.SRC.FIG0906
OBJECT ACCESS NAME:    DUMY
LISTING ACCESS NAME:   MANUAL.PG.LST.FIG0906
OPTIONS:               M
PRINT WIDTH:           80
PAGE SIZE:             55
PROGRAM SIZE (LINES):  1000



 DXCBL          L.R.V YY.DDD COMPILED:MM/DD/YY HH:MM:SS  OPT=M      PAGE    2
LINE  DEBUG PG/LN   A...B.....................................................
   1                IDENTIFICATION DIVISION.
   2                PROGRAM-ID. QUERY.
   3               *     THIS PROGRAM WAS DEVELOPED AS A FUNCTIONAL
   4               *     DEMONSTRATION TEST TO CHECK INTERFACING
   5               *     COBOL APPLICATIONS WITH QUERY.
   6                ENVIRONMENT DIVISION.
   7                CONFIGURATION SECTION
   8                SOURCE-COMPUTER. TI-990.
   9                OBJECT-COMPUTER. TI-990.
  10                DATA DIVISION.
  11                WORKING-STORAGE SECTION.
```

Figure 9-6. COBOL Module Linked to Query (Sheet 1 of 3)

```
12                01   QUERY-NUMBER         PIC 9(5) COMP-1 VALUE 1.
13                01   RESULT-STATUS        PIC 9(5) COMP-1.
14                01   STATEMENT-LENGTH     PIC 9(5) COMP-1 VALUE 36.
15                01   RESULT-CODE          PIC XX.
16                01   QUERY-STATEMENT      PIC X(80) VALUE
17                     "LIST PERS SPOU CHLD FROM GENE NO HEADER  ".
18                01   EXTEND-FILE          PIC 9(5) COMP-1.
19                01   FORMAT-TEXT          PIC 9(5) COMP-1 VALUE 1.
20                01   LIST-TEXT            PIC 9(5) COMP-1 VALUE 1.
21                01   PAGELENGTH           PIC 9(5) COMP-1 VALUE 60.
22                01   PAGEWIDTH            PIC 9(5) COMP-1 VALUE 80.
23                01   LIST-PN              PIC X(48) VALUE "D.LIST ".
24                01   ALT-COLLATING-PN     PIC X(48) VALUE " ".
25                01   PASSWORD             PIC X(4) VALUE "DBMS".
26                01   CHAR-NUM             PIC X(6).
27                01   X                    PIC X.
28
29                PROCEDURE DIVISION.
30   >0000        MAIN SECTION.
31   >0000        BEGIN.
32
33   >0000             CALL "QCOMP" USING QUERY-NUMBER, RESULT-STATUS,
34                           RESULT-CODE, QUERY-STATEMENT, STATEMENT-LENGTH,
35                           PASSWORD, FORMAT-TEXT, LIST-TEXT, LIST-PN,
36                           PAGELENGTH, PAGEWIDTH, ALT-COLLATING-PN.
37
38   >0002             DISPLAY "RESULT STATUS FROM QCOMP = "
39                                 LINE 3 POSITION 1 ERASE.
40   >000C             MOVE RESULT-STATUS TO CHAR-NUM.
41   >0010             DISPLAY CHAR-NUM LINE 3 POSITION 30.
42   >0018             ACCEPT X LINE 24.
43
44   >001E             IF RESULT-STATUS = 0
45                        MOVE 1 TO EXTEND-FILE
46                        CALL "QEXEC" USING QUERY-NUMBER, RESULT-STATUS,
47                              RESULT-CODE, LIST-PN, EXTEND-FILE
48                        DISPLAY "RETURN STATUS AFTER QEXEC = "
49                                          LINE 3 POSITION 1 ERASE
50                        MOVE RESULT-STATUS TO CHAR-NUM
51                        DISPLAY CHAR-NUM LINE 3 POSITION 30
52                        ACCEPT X LINE 24.
53
54   >0046             CALL "QEND " USING QUERY-NUMBER, RESULT-STATUS.
55   >0048             STOP RUN.
56           ZZZZZZ END PROGRAM.                                *** END OF FILE
```

Figure 9-6. COBOL Module Linked to Query (Sheet 2 of 3)

```
 DXCBL          L.R.V YY.DDD COMPILED:MM/DD/YY HH:MM:SS  OPT=M      PAGE    3
 ADDRESS  SIZE  DEBUG ORDER TYPE                   NAME

  >002A      2   NBS     0   BINARY SIGNED         QUERY-NUMBER

  >002C      2   NBS     0   BINARY SIGNED         RESULT-STATUS

  >002E      2   NBS     0   BINARY SIGNED         STATEMENT-LENGTH

  >0030      2   ANS     0   ALPHANUMERIC          RESULT-CODE

  >0032     80   ANS     0   ALPHANUMERIC          QUERY-STATEMENT

  >0082      2   NBS     0   BINARY SIGNED         EXTEND-FILE

  >0084      2   NBS     0   BINARY SIGNED         FORMAT-TEXT

  >0086      2   NBS     0   BINARY SIGNED         LIST-TEXT

  >0088      2   NBS     0   BINARY SIGNED         PAGELENGTH

  >008A      2   NBS     0   BINARY SIGNED         PAGEWIDTH

  >008C     48   ANS     0   ALPHANUMERIC          LIST-PN

  >00BC     48   ANS     0   ALPHANUMERIC          ALT-COLLATING-PN

  >00EC      4   ANS     0   ALPHANUMERIC          PASSWORD

  >00F0      6   ANS     0   ALPHANUMERIC          CHAR-NUM

  >00F6      1   ANS     0   ALPHANUMERIC          X

READ ONLY BYTE SIZE =        >0156

READ/WRITE BYTE SIZE =       >0112

OVERLAY SEGMENT BYTE SIZE =  >0000

TOTAL BYTE SIZE =            >0268

    0 ERRORS

    0 WARNINGS


DXCBL          L.R.V YY.DDD COMPILED:MM/DD/YY HH:MM:SS  OPT=M      PAGE    4
PROGRAM   USING COUNT

 QCOMP         12
 QEND           2
 QEXEC          5
```

Figure 9-6. COBOL Module Linked to Query (Sheet 3 of 3)

9.6 COMMUNICATIONS

Several alternative communications packages are available to the 990 user. Depending on your application, you can generate a custom system to meet your needs.

9.7 COMMUNICATION EQUIPMENT

The communications modules available include the communications interface module, a choice of an asynchronous or synchronous modem, and an accessory auto-call unit (ACU). The communications interface module can be used with Belldata sets, which include modems and data-access arrangements.

The communications interface module provides an RS-232C interface with full modem control signals for asynchronous and synchronous modems. Baud rates of 75, 110, 150, 200, 300, 1200, 2400, 4800, and 9600 meet almost any communications requirement. Character size is selected from 5 to 9 bits with programmable parity (odd, even, or none). Other features include line break detection/generation, a 250-millisecond timer, programmable SYN, DLE stripping, false-bit-start-bit detection, stop-bit selection, and programmable self-test.

9.8 3780 EMULATOR COMMUNICATIONS SOFTWARE

The 3780 emulator communications software package provides a means of remote job entry (RJE) communications with an IBM 360/370 host computer or another 3780 emulator. Communication consists of exchanging data files between master and slave stations over leased point-to-point or switched telephone lines.

Using the 3780 emulator, systems running DX10 can serve as satellite and/or central stations in distributed processing networks, or can be used to handle RJE or batch data entry for processing by a host. Remote stations can be dialed manually or automatically with an optional ACU and a modem. Remote stations can also be operated in an unattended mode as a called station in a distributed network.

TI 3780 emulator communications software emulates the operation of the IBM 3780 Data Communications Terminal. However, unlike the IBM 3780, the source and destination of the transferred files using the TI 3780 emulator are not restricted to the card reader/punch and line printer. Any file, input device, or output device available to your system can be used for input or output.

10

Using SCI Command Procedures to Execute COBOL Tasks

10.1 GENERAL

This section introduces the application of custom tailored SCI command procedures (procs) to the execution of COBOL tasks. It does not attempt to provide materials sufficient for the general mastery of writing and using SCI procedures: this has been addressed in the *DX10 Systems Programmer's Guide* and the tutorial *SCI: A Self-Study Approach to Writing Command Procedures and Batch Streams*. Here the objective is to give you, the COBOL programmer, an understanding of the applicability of procs to your work, and to provide you with specific examples you may be able to adapt and use directly. If you have already had some experience with SCI procs, this chapter may serve as a review.

10.2 SCI COMMAND PROCEDURE ELEMENTS

SCI, the System Command Interpreter which runs under the DX10 Operating System, can be tailored for specific applications by writing and using new commands. Adding a new command involves writing a *command procedure*. This command procedure is a sequence of SCI statements stored in a file under a user-specified name and executed by SCI each time that name is invoked. The command procedure constitutes a new SCI command in its own right.

A command procedure is composed of SCI commands and their associated parameters, SCI primitives, and special statements that produce interactive field prompts at the user terminal. In some cases, the procedures may also invoke a user-supplied *command processor*.

SCI *primitives* are system routines which constitute the lowest-level components of the SCI language. They cannot be modified, deleted, or added by users. They are invoked by name, and their names are syntactically distinguished from other SCI commands by an initial period (.).

Interactive prompts allow procs to be generalized for use under varying conditions (for example, with different programs and files). Up to 22 such prompts may be incorporated in a single proc.

A *command processor* is a task—which you can supply—invoked within a command procedure. The processor can be an application program written in any language supported by DX10, or it can be a utility program written specifically to perform operations required by a command. Command processors are invoked from a command procedure using one of the primitives .BID, .DBID, or .QBID. The .BID primitive invokes the processor as a foreground task, .QBID invokes the processor as a background task, and .DBID places the processor in a suspended state in background mode. Processors executed by .BID must terminate to allow SCI to resume execution.

All procedures and processors must be *installed* before use.

10.3 EXAMPLE COMMAND PROCEDURES

Figure 10-1, Figure 10-2, and Figure 10-3 are sample command procedures. Each is explained in detail in the following discussions.

10.3.1 Example 1

Figure 10-1 shows a simple example of a user-supplied command procedure that functions primarily to issue a sequence of standard SCI commands. In all the examples in this section, the line numbers appearing at the left are included to facilitate a line-by-line explanation which is given following the proc. They are not part of the proc.

```
1)  COBCX (COBOL COMPILE AND EXECUTE)
2)  !
3)  XCCF SRC=MANUAL.PG.SRC.FIG1001,
4)  OBJ=MANUAL.PG.OBJ.FIG1001,
5)  LST=MANUAL.PG.LST.FIG1001
6)  !
7)  .IF @$$CC, EQ, 0     !IF COMPILE WORKED, RUN PROGRAM
8)  XCPF OBJ=MANUAL.PG.OBJ.FIG1001
9)  .ENDIF
10) !
11) .IF @$$CC, EQ, 0
12) MSG T="SUCCESSFUL COMPLETION: CODE = @$$CC"
13) .ELSE
14) MSG T="UNSUCCESSFUL COMPLETION: CODE = @$$CC"
15) .ENDIF
```

Figure 10-1. Simple SCI Procedure

```
1)  COBCX (COBOL COMPILE AND EXECUTE)
```

Line 1 gives the name of the proc (COBCX) and a note concerning its function. This is a non-executing but required statement.

```
2)  !
3)  XCCF SRC=MANUAL.PG.SRC.FIG1001,
4)  OBJ=MANUAL.PG.OBJ.FIG1001,
5)  LST=MANUAL.PG.LST.FIG1001
6)  !
7)  .IF @$$CC, EQ, 0     !IF COMPILE WORKED, RUN PROGRAM
```

The exclamation points (!) in lines 2, 6, and 7 denote the start of non-executing comments. These can be used at any column in a line (for example, following a command as in line 7) to signal the beginning of a comment. By including on a line *only* an exclamation point in column 1 (lines 2 and 6) a blank line can be introduced for spacing. An asterisk (*) can also be used to denote a non-executing comment, but only when placed at the beginning of a line. It has other meanings elsewhere.

```
3)  XCCF SRC=MANUAL.PG.SRC.FIG1001,
4)  OBJ=MANUAL.PG.OBJ.FIG1001,
5)  LST=MANUAL.PG.LST.FIG1001
```

In line 3, the example proc issues the SCI command XCCF (Execute COBOL Compiler in Foreground). The values given to SRC, OBJ, and LST in lines 3, 4, and 5 satisfy the command processor's requirements for the specification of file locations. Were you to issue the XCCF command interactively, you would be prompted (non-optionally) for these filenames. Thus, they must be included as part of the command as entered from the proc.

```
 7)  .IF @$$CC, EQ, 0     !IF COMPILE WORKED, RUN PROGRAM
 8)  XCPF OBJ=MANUAL.PG.OBJ.FIG1001
 9)  .ENDIF
10)  !
11)  .IF @$$CC, EQ, 0
12)  MSG T="SUCCESSFUL COMPLETION: CODE = @$$CC"
13)  .ELSE
14)  MSG T="UNSUCCESSFUL COMPLETION: CODE = @$$CC"
15)  .ENDIF
```

In lines 7 through 9 the example proc employs the .IF/.ENDIF construct, and in lines 11 through 15 the .IF/.ELSE/.ENDIF construct. .IF, .ELSE, and .ENDIF are SCI primitives. Their usage is similar to that in most programming languages.

In order to explain lines 7 through 9, it is necessary to review the topic of SCI *synonyms*. These are short words or acronyms which appear in procs preceded by the "@" sign, and which function as macros. That is, before a line containing a synomym is executed, a value is substituted for the synonym. In line 7, $$CC is a synonym and will be replaced before execution with a four-digit hexadecimal completion code returned by a command processor. The value of the code indicates whether the compilation initiated by the XCCF command was successful.

$$CC is a *global synonym*; that is, a synonym whose value is set by the DX10 operating system and made available to a user automatically. You can also define your own personal synonyms by using the SCI command AS (Assign Synonym) or the primitive .SYN (discussed in the next example). Most typically, synonyms are used to substitute for long pathnames, but they can also substitute for SCI commands, complete with argument lists. For a complete discussion of synonyms, refer to the *DX10 Systems Programmer's Guide*.

Depending on the value of the compilation completion code $$CC, the compiled program is executed with the XCPF command (Execute COBOL Program File) in proc line 8. Whatever the result of the compilation attempt, an appropriate message is displayed at the terminal by means of the .IF/.ELSE/.ENDIF control structure in lines 11 through 15. This completes the operations performed by the proc.

10.3.2 Example 2

Figure 10-2 shows an example of a user-supplied command procedure to execute a COBOL task. As shown, both primitives and other command procedures are used to assign needed synonyms, create necessary directories, and assign LUNOs. Note the use of asterisks, as an alternative to exclamation points, to denote comment lines. Unlike exclamation points, asterisks must be placed in column 1 to establish a line as a non-executing comment. As in the previous example, the line numbers appearing to the left of the listing are not part of the proc, but are included to facilitate the line-by-line explanation that follows.

```
1)  SALES (SALES ANALYSIS)=3,
2)  COMPANY NAME = *ACNM(@CNAME)
3)  *
4)  .SYN CNAME = "&COMPANY NAME"
5)  .SYN CMAS = "@CNAME.CMAS"
6)  .SYN OUTFILE = "@CNAME.PRINT"
7)  *
8)  CFDIR PATH = "@CNAME", MAX=2
9)  *
10) AL ACCESS=.ARPROG, PR=YES
11) *
12) XCTF P=@$$LU, T=SALEANAL
13) *
14) .SHOW @OUTFILE
15) *
16) MSG TEXT="PRINT RESULTS (Y/N)?", REPLY=ANS
17) *
18) .IF @ANS, LT, "Y"
19) .EXIT
20) .ENDIF
21) *
22) MSG TEXT="HOW MANY COPIES?", REPLY=NUMCOPIES
23) *
24) .LOOP
25) .WHILE @NUMCOPIES, GT, 0
26) PF FILE PATHNAME = @OUTFILE, LISTING DEVICE = LP01
27) .EVAL NUMCOPIES = @NUMCOPIES - 1
28) .REPEAT
29) *
30) .SYN NUMCOPIES=""
31) .SYN CMAS="", OUTFILE="",ANS=""
32) *
33) DD PATHNAME = @CNAME, ARE YOU SURE? = Y
34) *
35) RL L=@$$LU
```

Figure 10-2. Tailored SCI Procedure

The proc (SALES) functions as follows:

```
1)  SALES (SALES ANALYSIS)=3,
```

In line 1, the proc's name is defined, an expansion provided for that name, and a privilege level of 3 set. The latter means that only users at privilege level 3 or higher will be able to execute the proc.

```
2)  COMPANY NAME = *ACNM(@CNAME)
```

Because a variable (COMPANY NAME) is set to an input type (ACNM, for ACcess NaMe), line 2 results in an interactive prompt. The user is asked to supply a value for COMPANY NAME. This will be assigned later (in line 4) to the synonym CNAME.

Because a value (@CNAME) is given in parentheses following the ACNM keyword, that value will be displayed at the terminal along with the prompt. It constitutes a default response to the prompt.

The asterisk preceding the input type ACNM indicates to SCI that a reply to this prompt is optional. The user may choose not to input any value in response to the prompt. This is done by pressing the Return key without entering a value. Such a response, taken in concert with the specification of a default response, constitutes acceptance of the default.

```
4)  .SYN CNAME = "&COMPANY NAME"
```

In line 4, the ampersand (&) preceding COMPANY NAME tells SCI to use the value entered by the user when he was prompted for COMPANY NAME by the statement in line 2. If he entered RETURN to accept the default (@CNAME), the value for that synonym will be used for &COMPANY NAME. In either case, a new value will be set for the synonym CNAME, equal to whatever has been passed as &COMPANY NAME.

Note the difference between a synonym (preceded by @) and the user response to a prompt (preceded by &). Although both function as placeholders for some other literal, the user response is local to the proc which prompted for it. The synonym, by contrast, is available to any proc or command referencing it, until it is specifically removed from the user's synonym table. Thus if the proc above were to bid another proc, @CNAME would be defined for that second proc, whereas &COMPANY NAME would not.

```
5)  .SYN CMAS = "@CNAME.CMAS"
6)  .SYN OUTFILE = "@CNAME.PRINT"
```

In lines 5 and 6, the .SYN primitive is invoked to assign the synonyms CMAS and OUTFILE using the synonym CNAME as a directory path. The synonyms assigned are employed within the user's COBOL program. The proc may thus be directed, by means of interactive responses, to receive its input from, and direct its output to, different files at different times. It will be unnecessary to change (and recompile, and relink) the referenced COBOL program.

```
8)  CFDIR PATH = "@CNAME", MAX=2
```

In line 8, the SCI command CFDIR (Create File DIRectory) creates a directory using synonym CNAME as the directory access name. This will be used to store output files temporarily.

```
10)  AL ACCESS=.ARPROG, PR=YES
```

In line 10, the SCI command AL assigns a LUNO to the program file containing the task to be executed. $$LU is a global synonym automatically set by SCI to the LUNO number assigned.

```
12)  XCTF P=@$$LU, T=SALEANAL
```

The SCI command XCTF is invoked in line 12 to execute the COBOL task in foreground mode. The values given for the variables P and T satisfy the requirements of the command.

```
14)  .SHOW @OUTFILE
```

In line 14, the .SHOW primitive is used to display the output file on the VDT screen.

```
16)  MSG TEXT="PRINT RESULTS (Y/N)?", REPLY=ANS
17)  *
18)  .IF @ANS, LT, "Y"
19)  .EXIT
20)  .ENDIF
21)  *
22)  MSG TEXT="HOW MANY COPIES?", REPLY=NUMCOPIES
```

In lines 16 through 20 the user is asked whether a printout of the run's results is desired. A negative reply causes the proc to be exited (line 19). If the user indicates he does want a printout, he is asked how many copies (line 22).

```
24)  .LOOP
25)  .WHILE @NUMCOPIES, GT, 0
26)  PF FILE PATHNAME = @OUTFILE, LISTING DEVICE = LP01
27)  .EVAL NUMCOPIES = @NUMCOPIES - 1
28)  .REPEAT
```

The primitives .LOOP, .WHILE, and .REPEAT in lines 24 through 28 comprise a control structure that functions like loop constructs in other programming languages. The loop begins with a check on the value of the integer-valued synonym NUMCOPIES. If it is positive, the SCI command PF (Print File) is issued, with OUTFILE the destination. The primitive .EVAL performs the specified arithmetic on NUMCOPIES, reducing it by 1. (Note that NUMCOPIES, being a synonym, is stored as a character string; nevertheless, this character string must represent an integer in order to be used with .EVAL .) The .REPEAT primitive signals the boundary of the loop. Hence, control returns to line 25, and the loop is continued until the value of NUMCOPIES has been reduced to 0.

```
30)  .SYN NUMCOPIES=""
31)  .SYN CMAS="", OUTFILE="",ANS=""
32)  *
33)  DD PATHNAME = @CNAME, ARE YOU SURE? = Y
34)  *
35)  RL L=@$$LU
```

The proc has now finished the work it was created to do, and the remaining statements enact cleanup operations. The synonyms NUMCOPIES, CMAS, OUTFILE, and ANS are removed from the user's synonym list by being assigned null values (lines 30 and 31). The directory CNAME, created to retain output until any requested printouts are performed, is now deleted with the SCI command DD (Delete Directory). Finally, the LUNO assigned in line 10 is released (line 35).

10.3.3 Example 3

When a procedure bids a processor (COBOL task), SCI places all the parameters specified in the PARMS option of the bid primitive (.BID, .DBID, or .QBID) into a table in the task communications area (TCA). Parameters 1 through 5 are used by COBOL; parameters 6 through 8 must be reserved for future use. If it is desired to obtain the parameters from the TCA, the processor must be coded appropriately. The procedure shown in Figure 10-3 illustrates how to pass an additional parameter to the COBOL run-time interpreter. The COBOL task is executed by the .BID primitive.

```
1)  SALES (SALES ANALYSIS)=3,
2)  COMPANY NAME = *ACNM(@CNAME),
3)  PROGRAM FILE LUNO = INT("@$XCT$P"),
4)  TASK ID OR NAME = STRING("@$XCT$T"),
5)  DEBUG MODE = YESNO(NO),
6)  MESSAGE ACCESS NAME = *ACNM("@$XCT$L"),
7)  SWITCHES = *STRING("00000000"),
8)  FUNCTION KEYS = YESNO(NO)
9)  .SYN  CNAME =  "&COMPANY NAME"
10) .SYN $XCT$P =  "&PROGRAM FILE LUNO"
11) .SYN $XCT$T =  "&TASK ID OR NAME"
12) .SYN $XCT$L =  "&MESSAGE ACCESS NAME"
13) .BID TASK = &TASK ID OR NAME, LUNO = "&PROGRAM FILE LUNO",
14) PARMS = (,"&DEBUG MODE",@&MESSAGE ACCESS NAME,
15) "&SWITCHES",&FUNCTION KEYS,,,,,@CNAME)
16) *** PARMS # 6, 7, & 8 ARE RESERVED FOR FUTURE COBOL USE
```

Figure 10-3. COBOL Procedure

Only a few new elements are introduced in this third example. Notice the three new input types: INT (integer), STRING, and YESNO. Note that the dollar signs ($) used in the synonym names have no special meaning within the context of an SCI prompt. However, dollar signs should be used with care in synonym names since most global (system-defined) synonyms use them, and you might inadvertently redefine a global synonym needed for other purposes.

In line 13 the .BID primitive is invoked to bid a command processor. Values passed as parameters to the task are those for which the user was interactively prompted.

NOTE

Future releases of software products from Texas Instruments may modify the standard command procedures such that tailored command procedures utilizing the method shown in Figure 10-3 may require modification.

Figure 10-4 shows a COBOL program module and illustrates the technique for retrieving additional SCI parameters in the module at execution time. Refer to Appendix D for details on the COBOL subroutine library module C$PARM.

```
 DXCBL          L.R.V YY.DDD COMPILED:MM/DD/YY HH:MM:SS  OPT=M      PAGE    1

SOURCE ACCESS NAME:    MANUAL.PG.SRC.FIG1004
OBJECT ACCESS NAME:    MANUAL.PG.OBJ.FIG1004
LISTING ACCESS NAME:   MANUAL.PG.LST.FIG1004
OPTIONS:               M
PRINT WIDTH:           80
PAGE SIZE:             55
PROGRAM SIZE (LINES):  1000



 DXCBL          L.R.V YY.DDD COMPILED:MM/DD/YY HH:MM:SS  OPT=M      PAGE    2
LINE  DEBUG PG/LN   A...B.....................................................
   1                IDENTIFICATION DIVISION.
   2                PROGRAM-ID. COBOLPRM.
   3               *     THIS PROGRAM WAS DESIGNED TO ILLUSTRATE A METHOD
   4               *     OF OBTAINING SCI BID PARAMETERS THRU COBOL.
   5                ENVIRONMENT DIVISION.
   6                CONFIGURATION SECTION.
   7                SOURCE-COMPUTER. TI-990.
   8                OBJECT-COMPUTER. TI-990.
   9                DATA DIVISION.
  10                FILE SECTION.
  11                WORKING-STORAGE SECTION.
  12                01  ACTION PIC 99.
  13                01  ERR PIC XX.
  14                01  PARM-NO PIC S99 COMP-1.
  15                01  PARMS PIC X(40) VALUE " ".
  16                PROCEDURE DIVISION.
  17  >0000        MAIN-PROG.
  18  >0000            PERFORM GET-PARM UNTIL ACTION = 0.
  19  >000A            STOP RUN.
  20  >000C        GET-PARM.
  21  >000C            DISPLAY "PARM NO?:" LINE 1 ERASE.
  22  >0014            ACCEPT ACTION LINE 1 POSITION 10 PROMPT.
  23  >001E            IF ACTION NOT = 0
  24                      MOVE ACTION TO PARM-NO
  25                      CALL "C$PARM" USING ERR PARM-NO PARMS
  26                      DISPLAY "PARMS = " PARMS LINE 4
  27                      ACCEPT ACTION PROMPT.
  28         ZZZZZZ END PROGRAM.                          *** END OF FILE
```

Figure 10-4. COBOL Program Module Retrieving Additional SCI Parameters (Sheet 1 of 2)

```
 DXCBL          L.R.V YY.DDD COMPILED:MM/DD/YY HH:MM:SS  OPT=M      PAGE    3
 ADDRESS  SIZE  DEBUG ORDER TYPE                   NAME

  >002A      2   NSU    0   NUMERIC UNSIGNED      ACTION

  >002C      2   NSU    0   NUMERIC UNSIGNED      PARM-NO

  >002E     40   ANS    0   ALPHANUMERIC          PARMS

READ ONLY BYTE SIZE =        >00D0

READ/WRITE BYTE SIZE =       >005E

OVERLAY SEGMENT BYTE SIZE =  >0000

TOTAL BYTE SIZE =            >012E

    0 ERRORS

    0 WARNINGS


 DXCBL          L.R.V YY.DDD COMPILED:MM/DD/YY HH:MM:SS  OPT=M      PAGE    4
 PROGRAM    USING COUNT

 C$PARM           2
```

Figure 10-4. COBOL Program Module Retrieving Additional SCI Parameters (Sheet 2 of 2)

11

COBOL Device-Dependent Attributes

11.1 FUNCTION KEYS

Function keys are accessible to COBOL programs through the use of the ACCEPT statement with the ON EXCEPTION option. This option causes a function key terminator character to be mapped to a user code (refer to Table 11-1), which is returned to the program. The ON EXCEPTION option is executed only if a function key terminates the input. Refer to the *COBOL Reference Manual* for complete details of the ACCEPT statement with the ON EXCEPTION option.

Function keys include event keys, system edit keys, and task edit keys. Event keys terminate input and cause immediate return to the calling routine. If input is not in progress, the next executed terminal input is terminated by the event character. Data is not lost. Event keys on the VDT include F1, F2, F3, F4, F5, F6, F7, F8, Command, Print, F9, F10, F11, F12, F13, and F14. System edit keys include cursor and display control keys including Erase Field, Previous Character, Forward Tab, Skip, Home, Next Field, Return, Erase Input, Delete Character, Insert Character, Next Character, and Attention. Some of these keys are also task edit keys. Task edit keys include Forward Tab, Next Line, Skip, Home, Next Field, Return, Erase Input, Initialize Input, Enter, Previous Field, and Previous Line.

NOTE

Because the Forward Tab and Return keys have a special termination effect on COBOL programs, they are not treated as function keys; the code returned is always zero.

Table 11-1. Function Key Mapping

VDT Key Character	Function Code
F1	01
F2	02
F3	03
F4	04
F5	05
F6	06
F7	07
F8	08
F9	09
F10	10
F11	11
F12	12
F13	13
F14	14
Command	40
Exit	41
Print	49
Previous Line	52
Next Line	53
Home	54
Next Field	55
Previous Field	56
Skip	57
Initialize Input	59
Erase Input	61
Enter	64

11.2 LOW VOLUME INPUT/OUTPUT (I/O)

The program listed in Figure 11-1 illustrates the use of the COBOL statements ACCEPT and DISPLAY. The program is discussed in the following paragraphs. The LINE number preceding each explanation corresponds to the line number where the statements appear in the listing in Figure 11-1. Refer to the *COBOL Reference Manual* for syntax and other more detailed information relating to the ACCEPT and DISPLAY statements and optional phrases.

LINE 27:

The DISPLAY statement displays a specified literal. The ERASE phrase clears the VDT screen prior to displaying the specified literal on line 1. If the LINE phrase is not present, the literal is displayed on line 2.

LINE 29:

The ACCEPT statement receives data from the screen. The OFF phrase allows data to be input at the VDT keyboard without being displayed on the VDT screen. The data is entered into the buffer at line 1, position 20. If the LINE phrase is not present, the data is displayed at the next line following the line currently containing the cursor. If no POSITION phrase is present, the position defaults to 1.

LINE 31:

This DISPLAY statement contains multiple operands. The first operand is displayed at line 2, position 1. Position 1 is the default for the first of multiple operands when the POSITION phrase is not present. The cursor position is the next character position after the operand. The second operand is displayed on line 3, position 2. Since the LINE phrase is not present, the operand is displayed on the next line following the line currently containing the cursor.

If the POSITION phrase is not present when a DISPLAY or ACCEPT statement is used, the second operand's position is 0, which is the default for all operands after the first in a multiple operand ACCEPT or DISPLAY statement. Also, the second operand is displayed on the same line at the current cursor position.

LINE 34:

This ACCEPT statement contains multiple operands. The first operand is accepted at line 2, position 24. The second operand is accepted at line 3, position 24. In both cases, the line and position values are specified to meet program requirements.

If the LINE phrase is not present for the two operands (PRINCIPAL and INT-RATE), the first operand is displayed on line 4; the second operand is displayed on line 5. If the POSITION phrase is not present for the two operands, the operands are accepted at position 1, which overwrites the instructional literals previously written at those positions.

For both operands, the CONVERT phrase is specified to change the accepted data into the receiving field format. Operations such as removal of illegal characters, decimal points, and commas; setting the correct sign if numeric; justification; and data type conversions take place if the CONVERT phrase is present.

The ECHO phrase is specified to redisplay the accepted data item after any necessary conversions are performed. It has meaning only if the CONVERT phrase is also present. The line and position of the data is the same on the echo as when originally accepted.

The ON EXCEPTION phrase forces reporting of errors occurring during the CONVERT operation. It is also used to report an ACCEPT operation terminated by a function key. This option is only available if the COBOL task was executed with the FUNCTION KEYS enabled. The ON EXCEPTION phrase is only required for the last operand if the ACCEPT statement contains multiple operands.

LINE 46:

This DISPLAY statement also contains multiple operands. The purpose is to display a literal and a data item on the same line. The first operand is displayed at line number 4, as specified. The default position is 1 for the first operand of a multiple operand DISPLAY statement.

The second operand must specify the same line number as the first operand, or it will be displayed on the following line. The position on the first line must also be specified so as not to overwrite the first operand displayed.

The SIZE phrase is present on the second operand to truncate the least significant character position.

LINE 49:
This DISPLAY statement illustrates a way of generating a display of any specified size up to 80 by specifying the SIZE phrase with either the figurative constant SPACE or QUOTE as the operand.

LINE 51:
This DISPLAY statement illustrates the results of displaying an operand at a line and position value that is greater than the maximum defined for the VDT device.

On a VDT, the maximum number of lines displayed is 24. The maximum number of characters on a line is 80.

When a line number is specified that is greater than the maximum defined line number, it is adjusted modulo the maximum line number. In this case, the data is displayed on Line 8.

When a position number is specified that is greater than the maximum defined position number, it is adjusted modulo the maximum position number. In this case, the data is displayed at position 10.

LINE 53:
The ACCEPT statement accepts 10 characters of data from the VDT screen at line 8, position 5. This 10-character input field overlaps data previously displayed on the screen.

If the Return key is pressed before 10 data characters are entered, the TAB phrase causes all data characters from the beginning through the end of the field, including the previously displayed data, to be received. If the TAB phrase is not present, only the entered data characters are received. However, if function keys are enabled when the program is executed, all data characters are received from the beginning through the ending of the input field regardless of the presence of the TAB phrase.

If 10 data characters are entered, the TAB phrase forces a wait until the Forward Tab key or Return key is pressed. All other data characters are ignored when waiting for the Forward Tab or Return key.

The PROMPT phrase initializes the input field prior to accepting any data characters. This prevents reception of erroneous data remaining on the VDT screen from prior operations. If program requirements dictate acceptance of previously displayed information, the Skip key can be pressed by the user at the terminal to clear a field of any remaining data prior to acceptance. The Skip key fills the field with blanks from the cursor position through the end of the field.

LINES 57, 59:

These ACCEPT statements illustrate how to accept numeric data items whose usage is not DISPLAY. The CONVERT phrase must be present to convert the ASCII input to the specified format of the receiving operand.

Use the SIZE phrase when accepting numeric operands described other than DISPLAY. The SIZE phrase specifies a length equivalent to the length given in the PICTURE clause for the receiving operand.

LINE 61:

This ACCEPT statement illustrates how to change the prompt character. Use the keyword PROMPT followed by the desired one-character, nonnumeric literal enclosed in quotes. The default PROMPT character is the underscore.

```
 DXCBL          L.R.V YY.DDD COMPILED:MM/DD/YY HH:MM:SS  OPT=M      PAGE    1

SOURCE ACCESS NAME:    MANUAL.PG.SRC.FIG1101
OBJECT ACCESS NAME:    DUMY
LISTING ACCESS NAME:   MANUAL.PG.LST.FIG1101
OPTIONS:               M
PRINT WIDTH:           80
PAGE SIZE:             55
PROGRAM SIZE (LINES):  1000



 DXCBL          L.R.V YY.DDD COMPILED:MM/DD/YY HH:MM:SS  OPT=M      PAGE    2
LINE  DEBUG PG/LN   A...B.................................................
   1                IDENTIFICATION DIVISION.
   2                PROGRAM-ID. ACCDIS.
   3               *     THIS PROGRAM WAS DEVELOPED TO ILLUSTRATE
   4               *     THE EFFECTS OF THE OPTIONAL PHRASES ON THE
   5               *     ACCEPT/DISPLAY COMMANDS
   6                ENVIRONMENT DIVISION.
   7                CONFIGURATION SECTION.
   8                SOURCE-COMPUTER. TI-990.
   9                OBJECT-COMPUTER. TI-990.
  10                DATA DIVISION.
  11                WORKING-STORAGE SECTION.
  12                01   ASTERISKS PIC X(10) VALUE ALL "*".
  13                01   PASSCODE  PIC X(6).
  14                01   PRINCIPAL PIC S999V99.
  15                01   INT-RATE  PIC 99V999.
  16                01   INTEREST  PIC 999V999.
  17                01   STATUS-CODE PIC 9(4).
  18                01   CVRT-FUNC REDEFINES STATUS-CODE.
  19                     02  CVRT  PIC 99.
  20                     02  FUNC  PIC 00.
  21                01   COMP-FOUR PIC S99 COMP-4.
  22                01   COMP-ONE  PIC S999 COMP-1.
```

Figure 11-1. Use of ACCEPT and DISPLAY Statements (Sheet 1 of 3)

```
 23                01   ACTION PIC X(4).
 24                PROCEDURE DIVISION.
 25  >0000         MAIN-PROG.
 26
 27  >0000              DISPLAY "ENTER PASSCODE:" LINE 1 ERASE.
 28
 29  >0008              ACCEPT PASSCODE LINE 1 POSITION 20 OFF.
 30
 31  >0013              DISPLAY "INPUT PRINCIPAL AMOUNT: " LINE 2
 32                             "INPUT INTEREST RATE:" POSITION 1.
 33
 34  >001F              ACCEPT PRINCIPAL LINE 2 POS 24 PROMPT CONVERT ECHO
 35                            INT-RATE  LINE 3 POS 24 PROMPT CONVERT ECHO
 36                                ON EXCEPTION STATUS-CODE CONTINUE.
 37
 38  >0035              IF CVRT  = 98
 39
 40                        DISPLAY "DATA CONVERSION ERROR" LINE 4
 41
 42                     ELSE
 43
 44                        COMPUTE INTEREST = PRINCIPAL * (INT-RATE / 100)
 45
 46                        DISPLAY "COMPUTED INTEREST:" LINE 4
 47                                INTEREST LINE 4 POSITION 24 SIZE 5.
 48
 49  >005B              DISPLAY QUOTE LINE 6 SIZE 30.
 50
 51  >0063              DISPLAY ASTERISKS LINE 32 POSITION 90.
 52
 53  >006B              ACCEPT  ASTERISKS LINE 8 POSITION 5 TAB.
 54
 55  >0075              DISPLAY ASTERISKS.
 56


 DXCBL          L.R.V YY.DDD COMPILED:MM/DD/YY HH:MM:SS  OPT=M      PAGE    3
LINE  DEBUG PG/LN   A...B.....................................................
 57  >0079              ACCEPT COMP-FOUR LINE 12 SIZE 2 PROMPT CONVERT.
 58
 59  >0083              ACCEPT COMP-ONE  LINE 13 SIZE 3 PROMPT CONVERT.
 60
 61  >008D              ACCEPT ACTION PROMPT "?".
 62
 63  >0094              STOP RUN.
 64         ZZZZZZ END PROGRAM.                              *** END OF FILE
```

Figure 11-1. Use of ACCEPT and DISPLAY Statements (Sheet 2 of 3)

```
 DXCBL          L.R.V YY.DDD COMPILED:MM/DD/YY HH:MM:SS  OPT=M      PAGE    4
 ADDRESS  SIZE  DEBUG ORDER TYPE                    NAME

  >002A     10   ANS     0   ALPHANUMERIC          ASTERISKS

  >0034      6   ANS     0   ALPHANUMERIC          PASSCODE

  >003A      6   NSS     0   NUMERIC SIGNED        PRINCIPAL

  >0040      5   NSU     0   NUMERIC UNSIGNED      INT-RATE

  >0046      6   NSU     0   NUMERIC UNSIGNED      INTEREST

  >004C      4   NSU     0   NUMERIC UNSIGNED      STATUS-CODE

  >004C      4   GRP     0   GROUP                 CVRT-FUNC
  >004C      2   NSU     0   NUMERIC UNSIGNED       CVRT
  >004E      2   NSU     0   NUMERIC UNSIGNED       FUNC

  >0050      1   NMS     0   MULTI BINARY SIGNED   COMP-FOUR

  >0052      2   NBS     0   BINARY SIGNED         COMP-ONE

  >0054      4   ANS     0   ALPHANUMERIC          ACTION

READ ONLY BYTE SIZE =        >01DC

READ/WRITE BYTE SIZE =       >005A

OVERLAY SEGMENT BYTE SIZE =  >0000

TOTAL BYTE SIZE =            >0236
   2
    0 ERRORS

    0 WARNINGS
```

Figure 11-1. Use of ACCEPT and DISPLAY Statements (Sheet 3 of 3)

The diagram in Figure 11-2 shows the contents of the VDT screen at completion of the program. Both line numbers and position numbers are indicated in the diagram. Line numbers are enclosed in parentheses to the right of the diagram; position numbers appear above the diagram. Both line and position numbers correspond to discussions in the explanation mentioned previously.

```
          (Position Numbers)                    (Line
                                                Numbers)

0012
1504
ENTER PASSCODE:                                  (1)
INPUT PRINCIPAL AMOUNT:20000ñ                    (2)
INPUT INTEREST RATE: 05500                       (3)
COMPUTED INTEREST: 01100                         (4)
                                                 (5)
''''''''''''''''''''''''''''''''                 (6)
'
                                                 (7)
AAA **********                                   (8)
AAA *****                                        (9)
                                                (10)
                                                (11)
15                                              (12)
128                                             (13)
????                                            (14)
```

Figure 11-2. Contents of VDT Screen at Program Completion

11.3 GRAPHIC INPUT/OUTPUT

Graphic characters may be represented on VDTs by calling a subprogram to turn on the graphic flag in the SVC call block. To enable the graphics option with the CALL statement, use:

```
CALL "C$GRPH".
```

To disable the graphics option, use:

```
CALL "C$GROF".
```

The routine "C$GRPH" need only be called one time (prior to the first ACCEPT/DISPLAY transfer of graphics characters). Refer to the *COBOL Reference Manual* for details on the ACCEPT and DISPLAY commands. Graphics are enabled for the duration of the program run or until "C$GROF" is called to disable the graphics option.

When graphics have been enabled by a call to C$GRPH, control characters (characters >00 through >1F) are displayed as graphic characters for input and output. All characters are treated as data; no action is taken on control characters that are requested.

Refer to Appendix D for a sample link control file to access routines in the COBOL subroutine library.

In the following figures, the graphics capabilities are enabled by calling the subroutine C$GRPH and disabled by calling C$GROF. Graphic characters must be generated as binary fields either by use of COMP-1 data items as in Figure 11-3 or COMP-4 data items as in Figure 11-4.

```
 DXCBL          L.R.V YY.DDD COMPILED:MM/DD/YY HH:MM:SS   OPT=M      PAGE    1

SOURCE ACCESS NAME:    MANUAL.PG.SRC.FIG1102
OBJECT ACCESS NAME:    DUMY
LISTING ACCESS NAME:   MANUAL.PG.LST.FIG1102
OPTIONS:               M
PRINT WIDTH:           80
PAGE SIZE:             55
PROGRAM SIZE (LINES):  1000


 DXCBL          L.R.V YY.DDD COMPILED:MM/DD/YY HH:MM:SS  OPT=M      PAGE    2
LINE  DEBUG PG/LN   A...B.....................................................
   1                IDENTIFICATION DIVISION.
   2                PROGRAM-ID. GRAPHICS.
   3               * THIS PROGRAM BOXES THE VDT SCREEN WITH A SOLID LINE.
   4                ENVIRONMENT DIVISION.
   5                CONFIGURATION SECTION.
   6                SOURCE-COMPUTER.       TI-990.
   7                OBJECT-COMPUTER.       TI-990.
   8                DATA DIVISION.
   9                WORKING-STORAGE SECTION.
  10                01  ACTION              PIC X.
  11                01  DISP-CHAR.
  12                    02 FILLER           PIC X.
  13                    02 CHR              PIC X.
  14                01  COMP-CHAR REDEFINES DISP-CHAR.
  15                    02 WRD              PIC 99 COMP-1.
  16                01  ROW                 PIC 99.
  17                01  COL                 PIC 99.
  18                01  LIN1                PIC 99 VALUE 1.
  19                01  LIN2                PIC 99 VALUE 24.
  20                01  POS1                PIC 99 VALUE 1.
  21                01  POS2                PIC 99 VALUE 80.
  22                PROCEDURE DIVISION.
  23  >0000         MAIN-PROGRAM.
  24  >0000             DISPLAY " " LINE 1 ERASE.
  25  >0008             CALL "C$GRPH".
  26  >000C         DISPLAY-BOX.
  27  >000C             MOVE >0A TO WRD.
  28  >0010             DISPLAY CHR LINE 1 POSITION 1.
  29  >0018             MOVE >1A TO WRD.
  30  >001C             DISPLAY CHR LINE 1 POSITION 80.
  31  >0024             MOVE >0D TO WRD.
  32  >0028             DISPLAY CHR LINE 24 POSITION 1.
  33  >0030             MOVE >1D TO WRD.
  34  >0034             DISPLAY CHR LINE 24 POSITION 80.
  35  >003C             MOVE >16 TO WRD.
  36  >0040             PERFORM DISPLAY-ROLL VARYING COL FROM 2 BY 1 UNTIL
  37                        COL > 79.
  38  >0054             MOVE >09 TO WRD.
  39  >0058             PERFORM DISPLAY-COL VARYING ROW FROM 2 BY 1 UNTIL
  40                        ROW > 23.
```

Figure 11-3. Graphics (Sheet 1 of 2)

```
 41  >006C              CALL "C$GROF".
 42  >006E              ACCEPT ACTION LINE 12 POSITION 40 PROMPT.
 43  >0078              STOP RUN.
 44  >007A          DISPLAY-ROLL.
 45  >007A              DISPLAY CHR LINE LIN1 POSITION COL
 46                             CHR LINE LIN2 POSITION COL.
 47  >008C          DISPLAY-COL.
 48  >008C              DISPLAY CHR LINE ROW POSITION POS1
 49                             CHR LINE ROW POSITION POS2.
 50  >009E          END-DSP. EXIT.
 51         ZZZZZZ END PROGRAM.                              *** END OF FILE

DXCBL          L.R.V YY.DDD COMPILED:MM/DD/YY HH:MM:SS  OPT=M      PAGE    3
ADDRESS  SIZE  DEBUG ORDER TYPE                    NAME

 >002A      1   ANS    0   ALPHANUMERIC           ACTION

 >002C      2   GRP    0   GROUP                  DISP-CHAR
 >002D      1   ANS    0   ALPHANUMERIC            CHR

 >002C      2   GRP    0   GROUP                  COMP-CHAR
 >002C      2   NBS    0   BINARY SIGNED           WRD

 >002E      2   NSU    0   NUMERIC UNSIGNED       ROW

 >0030      2   NSU    0   NUMERIC UNSIGNED       COL

 >0032      2   NSU    0   NUMERIC UNSIGNED       LIN1

 >0034      2   NSU    0   NUMERIC UNSIGNED       LIN2

 >0036      2   NSU    0   NUMERIC UNSIGNED       POS1

 >0038      2   NSU    0   NUMERIC UNSIGNED       POS2


READ ONLY BYTE SIZE =        >017E

READ/WRITE BYTE SIZE =       >0044

OVERLAY SEGMENT BYTE SIZE =  >0000

TOTAL BYTE SIZE =            >01C2

    0 ERRORS

    0 WARNINGS

DXCBL          L.R.V YY.DDD COMPILED:MM/DD/YY HH:MM:SS   OPT=M      PAGE   4
PROGRAM   USING  COUNT

 C$GROF          0
 C$GRPH          0
```

Figure 11-3. Graphics (Sheet 2 of 2)

```
 DXCBL          L.R.V YY.DDD COMPILED:MM/DD/YY HH:MM:SS  OPT=M       PAGE    1

SOURCE ACCESS NAME:    MANUAL.PG.SRC.FIG1103
OBJECT ACCESS NAME:    DUMY
LISTING ACCESS NAME:   MANUAL.PG.LST.FIG1103
OPTIONS:               M
PRINT WIDTH:           80
PAGE SIZE:             55
PROGRAM SIZE (LINES):  1000



 DXCBL          L.R.V YY.DDD COMPILED:MM/DD/YY HH:MM:SS  OPT=M      PAGE    2
LINE  DEBUG PG/LN   A...B.....................................................
   1                IDENTIFICATION DIVISION.
   2                PROGRAM-ID. GRAPHIC-CHARACTERS.
   3               * THIS PROGRAM WAS DEVELOPED AS A FUNCTIONAL
   4               * DEMONSTRATION TO TEST GRAPHICS CAPABILITY VIA
   5               * "DISPLAY" COMMAND.
   6                ENVIRONMENT DIVISION.
   7                CONFIGURATION SECTION.
   8                SOURCE-COMPUTER.        TI-990.
   9                OBJECT-COMPUTER.        TI-990.
  10                DATA DIVISION.
  11                WORKING-STORAGE SECTION.
  12                01   GRAPHICS-HEX-CHARACTERS.
  13                     02 X-00    PIC 99 COMP-4 VALUE >00.
  14                     02 X-01    PIC 99 COMP-4 VALUE >01.
  15                     02 X-02    PIC 99 COMP-4 VALUE >02.
  16                     02 X-03    PIC 99 COMP-4 VALUE >03.
  17                     02 X-04    PIC 99 COMP-4 VALUE >04.
  18                     02 X-05    PIC 99 COMP-4 VALUE >05.
  19                     02 X-06    PIC 99 COMP-4 VALUE >06.
  20                     02 X-07    PIC 99 COMP-4 VALUE >07.
  21                     02 X-08    PIC 99 COMP-4 VALUE >08.
  22                     02 X-09    PIC 99 COMP-4 VALUE >09.
  23                     02 X-0A    PIC 99 COMP-4 VALUE >0A.
  24                     02 X-0B    PIC 99 COMP-4 VALUE >0B.
  25                     02 X-0C    PIC 99 COMP-4 VALUE >0C.
  26                     02 X-0D    PIC 99 COMP-4 VALUE >0D.
```

Figure 11-4. Graphic Characters (Sheet 1 of 4)

```
 27                    02 X-0E    PIC 99 COMP-4 VALUE >0E.
 28                    02 X-0F    PIC 99 COMP-4 VALUE >0F.
 29                    02 X-10    PIC 99 COMP-4 VALUE >10.
 30                    02 X-11    PIC 99 COMP-4 VALUE >11.
 31                    02 X-12    PIC 99 COMP-4 VALUE >12.
 32                    02 X-13    PIC 99 COMP-4 VALUE >13.
 33                    02 X-14    PIC 99 COMP-4 VALUE >14.
 34                    02 X-15    PIC 99 COMP-4 VALUE >15.
 35                    02 X-16    PIC 99 COMP-4 VALUE >16.
 36                    02 X-17    PIC 99 COMP-4 VALUE >17.
 37                    02 X-18    PIC 99 COMP-4 VALUE >18.
 38                    02 X-19    PIC 99 COMP-4 VALUE >19.
 39                    02 X-1A    PIC 99 COMP-4 VALUE >1A.
 40                    02 X-1B    PIC 99 COMP-4 VALUE >1B.
 41                    02 X-1C    PIC 99 COMP-4 VALUE >1C.
 42                    02 X-1D    PIC 99 COMP-4 VALUE >1D.
 43                    02 X-1E    PIC 99 COMP-4 VALUE >1E.
 44                    02 X-1F    PIC 99 COMP-4 VALUE >1F.
 45                01  GRAPHIC-HEX-CHARS REDEFINES GRAPHIC-HEX-CHARACTERS.
 46                     02 HEX-CHAR   PIC 99 COMP-4 OCCURS 32 INDEXED BY X.
 47                01  ACTION              PIC X.
 48                01  DISP-HEX-CHAR.
 49                    02 HEX               PIC 99 COMP-4.
 50                    02 CHR REDEFINES HEX PIC X.
 51                01  ROW                 PIC 99 VALUE 0.
 52                01  COL                 PIC 99 VALUE 0.


 DXCBL          L.R.V YY.DDD COMPILED:MM/DD/YY HH:MM:SS  OPT=M      PAGE    3
LINE  DEBUG PG/LN   A...B.....................................................
 53                /
 54                 PROCEDURE DIVISION.
 55  >0000          MAIN-PROGRAM.
 56  >0000              DISPLAY " " LINE 1 ERASE.
 57  >0008              CALL "C$GRPH".
 58  >000A              PERFORM DISPLAY-GRAPHIC VARYING X FROM 1 BY 1
 59                       UNTIL X > 8.
 60  >001E              CALL "C$GROF".
 61  >0020              ACCEPT ACTION LINE 24.
 62  >0026              STOP RUN.
 63  >0028          DISPLAY-GRAPHIC.
 64  >0028              ADD 2 TO ROW.
 65  >002E              MOVE HEX-CHAR (X) TO HEX.
 66  >0038              DISPLAY CHR LINE ROW POSITION 5.
 67  >0040              MOVE HEX-CHAR (X + 8) TO HEX.
 68  >004A              DISPLAY CHR LINE ROW POSITION 25.
 69  >0052              MOVE HEX-CHAR (X + 16) TO HEX.
 70  >005C              DISPLAY CHR LINE ROW POSITION 45.
 71  >0064              MOVE HEX-CHAR (X + 24) TO HEX.
 72  >006E              DISPLAY CHR LINE ROW POSITION 65.
 73         ZZZZZZ END PROGRAM.                              *** END OF FILE
```

Figure 11-4. Graphic Characters (Sheet 2 of 4)

```
DXCBL          L.R.V YY.DDD COMPILED:MM/DD/YY HH:MM:SS  OPT=M      PAGE    4
ADDRESS  SIZE  DEBUG ORDER TYPE                    NAME

 >002A     32   GRP    0   GROUP                  GRAPHIC-HEX-CHARACTERS
 >002A      1   NMB    0   MULTI BINARY            X-00
 >002B      1   NMB    0   MULTI BINARY            X-01
 >002C      1   NMB    0   MULTI BINARY            X-02
 >002D      1   NMB    0   MULTI BINARY            X-03
 >002E      1   NMB    0   MULTI BINARY            X-04
 >002F      1   NMB    0   MULTI BINARY            X-05
 >0030      1   NMB    0   MULTI BINARY            X-06
 >0031      1   NMB    0   MULTI BINARY            X-07
 >0032      1   NMB    0   MULTI BINARY            X-08
 >0033      1   NMB    0   MULTI BINARY            X-09
 >0034      1   NMB    0   MULTI BINARY            X-0A
 >0035      1   NMB    0   MULTI BINARY            X-0B
 >0036      1   NMB    0   MULTI BINARY            X-0C
 >0037      1   NMB    0   MULTI BINARY            X-0D
 >0038      1   NMB    0   MULTI BINARY            X-0E
 >0039      1   NMB    0   MULTI BINARY            X-0F
 >003A      1   NMB    0   MULTI BINARY            X-10
 >003B      1   NMB    0   MULTI BINARY            X-11
 >003C      1   NMB    0   MULTI BINARY            X-12
 >003D      1   NMB    0   MULTI BINARY            X-13
 >003E      1   NMB    0   MULTI BINARY            X-14
 >003F      1   NMB    0   MULTI BINARY            X-15
 >0040      1   NMB    0   MULTI BINARY            X-16
 >0041      1   NMB    0   MULTI BINARY            X-17
 >0042      1   NMB    0   MULTI BINARY            X-18
 >0043      1   NMB    0   MULTI BINARY            X-19
 >0044      1   NMB    0   MULTI BINARY            X-1A
 >0045      1   NMB    0   MULTI BINARY            X-1B
 >0046      1   NMB    0   MULTI BINARY            X-1C
 >0047      1   NMB    0   MULTI BINARY            X-1D
 >0048      1   NMB    0   MULTI BINARY            X-1E
 >0049      1   NMB    0   MULTI BINARY            X-1F

 >002A     32   GRP    0   GROUP                  GRAPHIC-HEX-CHARS

 >0052      2   NBS    0   INDEX-NAME             X
 >002A      1   NMB    1   MULTI BINARY            HEX-CHAR

 >004A      1   ANS    0   ALPHANUMERIC           ACTION

 >004C      1   GRP    0   GROUP                  DISP-HEX-CHAR
 >004C      1   NMB    0   MULTI BINARY            HEX
 >004C      1   ANS    0   ALPHANUMERIC            CHR

 >004E      2   NSU    0   NUMERIC UNSIGNED       ROW

 >0050      2   NSU    0   NUMERIC UNSIGNED       COL
```

Figure 11-4. Graphic Characters (Sheet 3 of 4)

```
 DXCBL          L.R.V YY.DDD COMPILED:MM/DD/YY HH:MM:SS  OPT=M      PAGE    5
 ADDRESS  SIZE  DEBUG ORDER TYPE                   NAME

READ ONLY BYTE SIZE =        >0124

READ/WRITE BYTE SIZE =       >0058

OVERLAY SEGMENT BYTE SIZE =  >0000

TOTAL BYTE SIZE =            >017C

    0 ERRORS

    0 WARNINGS



 DXCBL          L.R.V YY.DDD COMPILED:MM/DD/YY HH:MM:SS  OPT=M      PAGE    6
 PROGRAM   USING  COUNT

  C$GROF          0
  C$GRPH          0
```

Figure 11-4. Graphic Characters (Sheet 4 of 4)

12

Error Processing

12.1 GENERAL

The COBOL run time retains the status of the most recent I/O operation requested by the COBOL program. The status can be either successful or unsuccessful completion, caused by either a condition detected by COBOL or an error detected by the operating system.

12.2 FILE I/O STATUS

If you specify the FILE STATUS clause in a file-control-entry, a status value is placed into a two-character data item. This is applicable during execution of an OPEN, CLOSE, READ, WRITE, REWRITE, DELETE, or START statement and before any USE procedure is executed. Table 12-1 lists the file status codes and the I/O operations that can generate each code. The ANSI COBOL specifications define these status codes for specific I/O operations. Table 12-2 indicates the file status code that is returned for specific errors detected by the operating system. COBOL file status code 30 includes any error not listed under errors detected by the operating system.

Table 12-1. File Status Table

FILE STATUS	I/O OPERATION							
	OPEN	CLOSE	READ	WRITE	REWRITE	START	DELETE	UNLOCK
00	X	X	X	X	X	X	X	X
02			X					
10			X					
21				X	X			
22				X	X			
23			X		X	X	X	
24				X				
30	X	X	X	X	X	X	X	
34				X				
90			X	X	X	X	X	
91		X	X	X	X	X	X	
92	X							
93	X							
94	X							
95	X							
96			X					
97	X		X	X	X	X		
99	X		X		X		X	

2277280

Table 12-2. Operating System Errors and COBOL File Status Errors

Operating System	COBOL
Errors	File Status
30	30
31	30
32	23
35	30
B1	30
D2	99
Others	30
KIF Informative Errors	
B3	10
B4	22
B5	23
B7	99
B8	23
BD	23
D2	99
Others	30

12.3 FILE I/O STATUS VALUES

The leftmost character position of the COBOL file status data item is zero through three for ANSI-defined conditions and nine for implementor-defined conditions as follows:

Value	Condition
0	Successful completion
1	At end
2	Invalid key
3	Permanent error
9	Error detected by the operating system

The rightmost character position of the COBOL file status data item further describes the results of the I/O operation. The following list contains the COBOL file status codes and the conditions under which each status code can occur.

Code	Condition
00	SUCCESSFUL COMPLETION The I/O operation completed successfully.
02	SUCCESSFUL COMPLETION, DUPLICATE KEY A READ statement on a KIF completed successfully. The key value for the current key of reference is equal to the value of that same key in the next record of the current key of reference.
10	AT END Sequential READ statement on a sequential, relative record, or key-indexed file is unsuccessful because an attempt was made to read a record but no next logical record exists in the file.
21	INVALID KEY SEQUENCE Sequential WRITE statement on a KIF is unsuccessful because of a violation of the ascending sequence requirement for successive record key values. Sequential REWRITE statement on a KIF was unsuccessful because the key was not the same as the key returned by the preceding successful read.
22	INVALID KEY DUPLICATE Random WRITE statement on a relative record or key-indexed file was unsuccessful because an attempt was made to write a record that would create a duplicate key.
23	INVALID KEY NO RECORD Random READ, DELETE, or REWRITE statement on a relative record or key indexed file was unsuccessful because an attempt was made to reference a record identified by a key that did not exist in the file. Sequential START statement on a relative record or key-indexed file was unsuccessful because the stated comparison was not satisfied by any record in the file.
24	INVALID KEY BOUNDARY Sequential or random WRITE statement on a relative record or key indexed file was unsuccessful because an attempt was made to write beyond the externally defined boundaries of the file.
30	PERMANENT ERROR I/O statement was unsuccessful because a permanent I/O error (such as data check, parity error, or transmission error) was detected.
34	BOUNDARY VIOLATION Sequential write access on a sequential organization file was unsuccessful because an attempt was made to write beyond the externally defined boundaries of the file.

Code	Condition
90	INVALID OPERATION I/O statement was unsuccessful because of a violation of the COBOL I/O operation validity table. Refer to Table 12-3. Sequential delete or rewrite operation was unsuccessful because the operation was not preceded by a successful read.
91	FILE NOT OPENED I/O statement other than OPEN was unsuccessful because of a reference to an unopened file.
92	FILE NOT CLOSED OPEN statement was unsuccessful because an attempt was made to open a file already opened.
93	FILE NOT AVAILABLE OPEN statement was unsuccessful because an attempt was made to open a file either in the "closed with lock" state, or one for which no external correspondence exists.
94	INVALID OPEN OPEN statement was unsuccessful because of invalid open parameters such as the following: • Open mode disagrees with select statement assignment. • Open extend attempted on relative record or key-indexed file. • Incompatibility in variable length record. • Key count in OPEN statement does not agree with count specified in the file definition (KIF only). • Key displacement in OPEN statement does not agree with displacement specified in the file definition (KIF only). • Key attributes "modifiable" and "duplicatable" in OPEN statement does not agree with those specified in the file definition (KIF only). • Bad disk name. • Access name syntax problem.
95	INVALID DEVICE OPEN statement was unsuccessful because a mismatch occurred between the device requested and the device assigned by the external correspondence. Refer to Table 12-4.

Code	Condition
96	UNDEFINED RECORD POINTER Sequential READ statement for all file types was unsuccessful because the current record pointer was in an undefined state. This can occur only if a previous READ or START statement was unsuccessful.
97	INVALID RECORD LENGTH WRITE statement was unsuccessful because the record length was outside the bounds defined by the minimum and maximum record sizes. The record length is in hexadecimal. Sequential or random REWRITE statement for all file types was unsuccessful because the new record length was different from that of the record to be rewritten. An OPEN statement was unsuccessful because the maximum record length was less than the minimum record length or the minimum record length was equal to zero.
99	RECORD LOCKED/OPEN EXCLUSIVE ACCESS A DELETE, READ, REWRITE, or START I/O statement on a relative record or key-indexed file was unsuccessful because another task locked the reference record. This error can occur on execution of a READ statement if both a USE procedure and a FILE STATUS data item are in effect for the associated file. An OPEN statement is unsuccessful because another task opened the file with exclusive access.

Table 12-3 shows the open mode necessary to perform various I/O operations for sequential, relative record, and key-indexed files with sequential, random, and dynamic access. Table 12-4 shows the program-requested device class versus the corresponding actual device at execution time, with the permissible open modes.

Table 12-3. COBOL I/O Operation Validity Table

		OPEN MODE			
ACCESS	OPERATION	INPUT	OUTPUT	I-O	EXTEND
SEQUENTIAL	READ[2] WRITE REWRITE[3] DELETE[3] START[2]	SEQ REL KEY REL KEY	SEQ REL KEY	SEQ REL KEY SEQ REL KEY REL KEY REL KEY	SEQ
RANDOM	READ[2] WRITE REWRITE DELETE START[2]	REL KEY	REL KEY	REL KEY REL KEY REL KEY REL KEY	
(1) DYNAMIC	READ[2] WRITE REWRITE DELETE START[2] READ NEXT[2]	REL KEY REL KEY	REL KEY	REL KEY REL KEY REL KEY REL KEY REL KEY REL KEY	

SEQ = SEQUENTIAL FILES
REL = RELATIVE RECORD FILES
KEY = KEY INDEXED FILES

NOTES:

1. ALL ACCESSES ARE RANDOM EXCEPT READ NEXT, AND DELETE OR REWRITE PRECEDED BY A READ NEXT.
2. SETS CURRENT RECORD POINTER (OPEN ALSO SETS CURRENT RECORD POINTER.)
3. MUST BE PRECEDED BY A SUCCESSFUL READ.

2277281

Table 12-4. Device Correspondence Table

REQUESTED DEVICE	CORRESPONDING DEVICE				
	0 DISK	1 CARD READER	2 RESERVED	3 PRINTER	4[1] TAPE
0 RANDOM	IN, OUT, IO, EXT				
1 INPUT	IN	IN	IN		IN
2 OUTPUT	OUT, EXT		OUT		OUT
3 PRINT	OUT, EXT			OUT	OUT
4 INPUT-OUTPUT	IN, OUT, EXT				IN, OUT, EXT

IN = OPEN MODE INPUT
OUT = OPEN MODE OUTPUT
IO = OPEN MODE I-O
EXT = OPEN MODE EXTEND

NOTES:

1. TAPE INCLUDES MAGNETIC TAPE, MAGNETIC CASSETTE, VDT, AND ASR 733.

2277282

12.4 USE OF DECLARATIVES

If a COBOL program is to intercept and process errors, the program must contain the USE statement in the declaratives section for each file. Also, you should specify the file-control-entry to receive the file status code returned on each I/O request. Refer to the *COBOL Reference Manual* for detailed information on the FILE STATUS clause and the USE statement.

Program control transfers to the designated declarative procedure when any of the following occurs:

- When the leftmost character of the file status code is not equal to zero
- Upon recognition of the invalid key condition when the INVALID KEY phrase has not been specified in the I/O statement
- Upon recognition of the at-end condition when the AT END phrase has not been specified in the I/O statement

The file status code can be checked and processed under program control. Depending on the code received, corrective action can be taken by the program or errors can be ignored. If errors are ignored, results are unpredictable. Execution of an INVALID KEY or AT END phrase takes precedence over DECLARATIVES. If an INVALID KEY phrase, an AT END phrase, or a declarative procedure is not present for a file, the program automatically terminates when an I/O error is detected during an I/O request.

By knowing which operating system error was originally detected, you can determine the cause and correct the error. Some errors, such as COBOL error 30, have multiple meanings. The subroutine C$RERR is available for obtaining the operating system I/O status of the most recent I/O request attempted. Refer to Appendix D for details on the subroutine C$RERR. COBOL programs should always include the call to C$RERR in the declarative section for every file. Then, should an error occur, you can examine the appropriate COBOL file status and the OS error to determine the cause of the error.

Figure 12-1 shows how to intercept I/O errors through the use of a declarative section in a COBOL program. Subroutine C$RERR is called in the program to obtain the operating system I/O status of the I/O request. The example program ignores an invalid record length error (error 97). The file is successfully opened and subsequent I/O requests can be performed against the file.

```
 DXCBL          L.R.V YY.DDD COMPILED:MM/DD/YY HH:MM:SS  OPT=M      PAGE    1

SOURCE ACCESS NAME:    MANUAL.PG.SRC.FIG1201
OBJECT ACCESS NAME:    DUMY
LISTING ACCESS NAME:   MANUAL.PG.LST.FIG1201
OPTIONS:               M
PRINT WIDTH:           80
PAGE SIZE:             55
PROGRAM SIZE (LINES):  1000



 DXCBL          L.R.V YY.DDD COMPILED:MM/DD/YY HH:MM:SS  OPT=M      PAGE    2
LINE  DEBUG PG/LN   A...B.................................................
   1                IDENTIFICATION DIVISION.
   2                PROGRAM-ID. SEQIO.
   3               *     THIS PROGRAM WAS DESIGNED AS A FUNCTIONAL
   4               *     DEMONSTRATION TEST FOR CHECKING ERROR HANDLING
   5               *     CAPABILITIES THRU DECLARATIVES.
   6                ENVIRONMENT DIVISION.
   7                CONFIGURATION SECTION.
   8                SOURCE-COMPUTER. TI-990.
   9                OBJECT-COMPUTER. TI-990.
  10                INPUT-OUTPUT SECTION.
  11                FILE-CONTROL.
```

Figure 12-1. Checking Error-Handling Capabilities Through DECLARATIVES (Sheet 1 of 6)

```
12                      SELECT MASTER ASSIGN TO RANDOM, "SEQIO."
13                         ORGANIZATION SEQUENTIAL
14                         ACCESS SEQUENTIAL
15                         STATUS SEQ-STATUS.
16                 DATA DIVISION.
17                 FILE SECTION.
18                 FD   MASTER LABEL RECORDS STANDARD.
19                 01   MST.
20                      02 SSAN PIC X(9).
21                      02 BADGE PIC X(7).
22                      02 NAMEX PIC X(20).
23                 WORKING-STORAGE SECTION.
24                 01   ACTION PIC X VALUE " ".
25                 01   SEQ-STATUS PIC XX VALUE " ".
26                 01   FUNCT-KEY PIC 99.
27                      88 F1        VALUE 01.
28                      88 Command   VALUE 40.
29                 01   OPEN-MODE PIC X(6) VALUE " ".
30                      88 OPEN-INPUT  VALUE "INPUT".
31                      88 OPEN-OUTPUT VALUE "OUTPUT".
32                      88 OPEN-IO     VALUE "I-O".
33                      88 OPEN-EXTEND VALUE "EXTEND".
34                 01   OPERATION PIC 99.
35                      88 OP-OPEN              VALUE 01.
36                      88 OP-CLOSE             VALUE 02.
37                      88 OP-READ              VALUE 03.
38                      88 OP-READ-NOLOCK       VALUE 04.
39                      88 OP-WRITE             VALUE 05.
40                      88 OP-REWRITE           VALUE 06.
41                      88 OP-UNLOCK            VALUE 07.
42                      88 OP-STOP              VALUE 08.
43                 01   BLNK PIC X(80) VALUE " ".
44                 01   LAST-OPERATION PIC 99 VALUE 0.
45                 01   ERROR-WORD PIC X(4) VALUE " ".
46                 PROCEDURE DIVISION.
47                 DECLARATIVES.
48  >0002         DECL SECTION.
49                      USE AFTER STANDARD ERROR PROCEDURE ON MASTER.
50  >0002         DEC1. DISPLAY "ERROR STATUS: " LINE 23
51                         SEQ-STATUS LINE 23 POSITION 16.
52
53                ** IF OPEN RETURNS "INVALID RECORD LENGTH ERROR - 97"
54                ** IGNORE ERROR CONDITION. FILE WILL BE OPENED.
55
56  >0010               IF SEQ-STATUS NOT = "97"
```

Figure 12-1. Checking Error-Handling Capabilities Through DECLARATIVES (Sheet 2 of 6)

```
 DXCBL          L.R.V YY.DDD COMPILED:MM/DD/YY HH:MM:SS  OPT=M      PAGE    3
LINE  DEBUG PG/LN   A...B.....................................................
  57                        CALL "C$RERR" USING ERROR-WORD
  58                        DISPLAY "DX10 ERROR: " LINE 23 POSITION 40
  59                           ERROR-WORD LINE 23 POSITION 54.
  60  >0028              DISPLAY "ENTER 'C' TO CONTINUE" LINE 24.
  61  >002E              ACCEPT ACTION PROMPT LINE 24 POSITION 24.
  62  >0038              IF ACTION = "C"
  63                        DISPLAY BLNK LINE 23
  64                        DISPLAY BLNK LINE 24
  65                        GO TO IO-LOOP.
  66  >004C              STOP RUN.
  67                 END DECLARATIVES.
  68  >004E          MAIN SECTION.
  69  >004E          MAIN-PROG.
  70  >004E              DISPLAY "SEQUENTIAL FILE I/O OPERATIONS" LINE 1 ERASE.
  71  >0056              DISPLAY "01 - OPEN"              LINE 2.
  72  >005C              DISPLAY "02 - CLOSE"             LINE 3.
  73  >0062              DISPLAY "03 - READ"              LINE 4.
  74  >0068              DISPLAY "04 - READ NO LOCK"      LINE 5.
  75  >006E              DISPLAY "05 - WRITE"             LINE 2 POSITION 40.
  76  >0076              DISPLAY "06 - REWRITE"           LINE 3 POSITION 40.
  77  >007E              DISPLAY "07 - UNLOCK"            LINE 4 POSITION 40.
  78  >0086              DISPLAY "08 - STOP"              LINE 5 POSITION 40.
  79  >0090          IO-LOOP.
  80  >0090              PERFORM IO-OPERATIONS THRU IO-EXIT
  81                        UNTIL OPERATION = 8 OR FUNCT-KEY = 40.
  82  >00A0              STOP RUN.
  83  >00A2          IO-OPERATIONS.
  84  >00A2              DISPLAY "OPERATION CODE: " LINE 10 POSITION 20.
  85  >00AA              ACCEPT OPERATION CONVERT PROMPT LINE 10 POSITION 40
  86                        ON EXCEPTION FUNCT-KEY PERFORM CHECK-EXCEPTION.
  87  >00BC          PERFORM-OPERATION.
  88  >00BC              IF OP-OPEN
  89                        PERFORM GET-OPEN-MODE
  90                        PERFORM OPEN-CODE.
  91  >00C6              IF OP-CLOSE
  92                        PERFORM CLOSE-CODE.
  93  >00CE              IF OP-READ
  94                        PERFORM READ-CODE.
  95  >00D6              IF OP-READ-NOLOCK
  96                        PERFORM READ-NOLOCK-CODE.
  97  >00DE              IF OP-WRITE
  98                        PERFORM WRITE-CODE.
  99  >00E6              IF OP-REWRITE
 100                        PERFORM REWRITE-CODE.
 101  >00EE              IF OP-UNLOCK
 102                        PERFORM UNLOCK-CODE.
```

Figure 12-1. Checking Error-Handling Capabilities Through DECLARATIVES (Sheet 3 of 6)

```
 103  >00F6              DISPLAY BLNK LINE 24.
 104  >00FC              IF OPERATION > 0 AND OPERATION < 9
 105                        MOVE OPERATION TO LAST-OPERATION
 106                        ELSE DISPLAY "INVALID OPERATION" LINE 24.
 107  >0116         IO-EXIT. EXIT.
 108  >0118         GET-OPEN-MODE.
 109  >0118              DISPLAY BLNK LINE 14.
 110  >011E              DISPLAY "OPEN MODE: INPUT, OUTPUT, I-O, EXTEND" LINE 1.
 111  >0124              ACCEPT OPEN-MODE PROMPT LINE 14 POSITION 50
 112                        ON EXCEPTION FUNCT-KEY PERFORM CHECK-EXCEPTION.
```

```
 DXCBL          L.R.V YY.DDD COMPILED:MM/DD/YY HH:MM:SS  OPT=M      PAGE    4
LINE  DEBUG PG/LN   A...B.................................................
 113  >0134              DISPLAY BLNK LINE 24.
 114  >013A              IF NOT OPEN-INPUT AND NOT OPEN-OUTPUT AND NOT OPEN-IO
 115                        AND NOT OPEN-EXTEND
 116                        DISPLAY "INVALID OPEN MODE" LINE 24
 117                        GO TO GET-OPEN-MODE.
 118  >015C         OPEN-CODE.
 119  >015C              IF OPEN-INPUT   OPEN INPUT  MASTER.
 120  >0168              IF OPEN-OUTPUT  OPEN OUTPUT MASTER.
 121  >0174              IF OPEN-IO      OPEN I-O    MASTER.
 122  >0180              IF OPEN-EXTEND  OPEN EXTEND MASTER.
 123  >018E         CLOSE-CODE.
 124  >018E              CLOSE MASTER.
 125  >0196         READ-CODE.
 126  >0196              READ MASTER.
 127  >01A0              DISPLAY MST LINE 18.
 128  >01A8         READ-NOLOCK-CODE.
 129  >01A8              READ MASTER NO LOCK.
 130  >01B2              DISPLAY MST LINE 18.
 131  >01BA         WRITE-CODE.
 132  >01BA              DISPLAY "SSAN: "LINE 20 "BADGE:" LINE 20 POSITION 2
 133                        "NAMEX: " LINE 20 POSITION 38.
 134  >01D0              ACCEPT SSAN PROMPT LINE 20 POSITION 7
 135                        BADGE PROMPT LINE 20 POSITION 27
 136                        BADGE PROMPT LINE 20 POSITION 45
 137                        ON EXCEPTION FUNCT-KEY PERFORM CHECK-EXCEPTION.
 138  >01F4              DISPLAY BLNK LINE 20.
 139  >01FA              WRITE MST.
 140  >0208         REWRITE-CODE.
 141  >0208              DISPLAY "SSAN:" LINE 20 "BADGE:" LINE 20 POSITION 2
 142                        "NAMEX: " LINE 20 POSITION 38.
 143  >021E              ACCEPT SSAN PROMPT LINE 20 POSITION 7
 144                        BADGE PROMPT LINE 20 POSITION 27
 145                        NAMEX PROMPT LINE 20 POSITION 45
 146                        ON EXCEPTION FUNCT-KEY PERFORM CHECK-EXCEPTION.
```

Figure 12-1. Checking Error-Handling Capabilities Through DECLARATIVES (Sheet 4 of 6)

```
147  >0242              DISPLAY BLNK LINE 20.
148  >0248              REWRITE MST.
149  >0256         UNLOCK-CODE.
150  >0256              UNLOCK MASTER.
151  >025C         CHECK-EXCEPTION.
152  >025C              IF F1 OR Command GO TO IO-EXIT.
153  >026C         END-OF-PROGRAM. EXIT.
154        ZZZZZZ  END PROGRAM.                                *** END OF FIL E


DXCBL          L.R.V YY.DDD COMPILED:MM/DD/YY HH:MM:SS  OPT=M      PAGE    5

  ADDRESS  SIZE  DEBUG  ORDER TYPE                 NAME

             0                FILE                MASTER
   >0026    36   GRP      0   GROUP                MST
   >0026     9   ANS      0   ALPHANUMERIC           SSAN
   >002F     7   ANS      0   ALPHANUMERIC           BADGE
   >0036    20   ANS      0   ALPHANUMERIC           NAMEX

   >004E     1   ANS      0   ALPHANUMERIC         ACTION

   >0050     2   ANS      0   ALPHANUMERIC         SEQ-STATUS

   >0052     2   NSU      0   NUMERIC UNSIGNED     FUNCT-KEY
                           0   CONDITION-NAME        F1
                           0   CONDITION-NAME        Command

   >0054     6   ANS      0   ALPHANUMERIC         OPEN-MODE
                           0   CONDITION-NAME        OPEN-INPUT
                           0   CONDITION-NAME        OPEN-OUTPUT
                           0   CONDITION-NAME        OPEN-IO
                           0   CONDITION-NAME        OPEN-EXTEND

   >005A     2   NSU      0   NUMERIC UNSIGNED     OPERATION
                           0   CONDITION-NAME        OP-OPEN
                           0   CONDITION-NAME        OP-CLOSE
                           0   CONDITION-NAME        OP-READ
                           0   CONDITION-NAME        OP-READ-NOLOCK
                           0   CONDITION-NAME        OP-WRITE
                           0   CONDITION-NAME        OP-REWRITE
                           0   CONDITION-NAME        OP-UNLOCK
                           0   CONDITION-NAME        OP-STOP

   >005C    80   ANS      0   ALPHANUMERIC         BLNK

   >00AC     2   NSU      0   NUMERIC UNSIGNED     LAST-OPERATION

   >00AE     4   ANS      0   ALPHANUMERIC         ERROR-WORD
```

Figure 12-1. Checking Error-Handling Capabilities Through DECLARATIVES (Sheet 5 of 6)

```
READ ONLY BYTE SIZE =           >05D4

READ/WRITE BYTE SIZE =          >0120

OVERLAY SEGMENT BYTE SIZE =     >0000

TOTAL BYTE SIZE =               >06F4

    0 ERRORS

    0 WARNINGS


DXCBL          L.R.V  YY.DDD COMPILED:MM/DD/YY HH:MM:SS  OPT=M      PAGE    6
PROGRAM   USING  COUNT

 CSREER          1
```

Figure 12-1. Checking Error-Handling Capabilities Through DECLARATIVES (Sheet 6 of 6)

13

Optimizing Run-Time Performance

13.1 GENERAL

By using certain coding techniques, you can optimize the performance of COBOL programs. To understand why certain techniques are desirable and others are impossible, you need an overview of the compiler and run-time structure.

The COBOL package contains a one-pass compiler that produces object code in a format called a POP. Because the POP format is especially tailored to the COBOL environment, object code produced by the compiler uses space much more efficiently than does machine language object code. This COBOL structure is well suited to the minicomputer environment where task space is critical.

However, since the compiler in this package only passes over the source once, global optimization techniques cannot be used to reduce the size of the object. Instead, you can optimize programs through coding techniques that decrease the size of the object generated or increase speed of execution. These techniques fall into the following categories:

- Object size considerations
- Arithmetic operations
- Control operations
- Move operations
- I/O operations

Future changes in the software may change some of the following performance considerations and techniques.

13.2 OBJECT SIZE CONSIDERATIONS

One way of optimizing the performance of a TI 990 COBOL program is to decrease the size of the compiler-generated object by using space-efficient code whenever possible.

Each paragraph and section which does not end in an unconditional GOTO increases object size by at least one word. Avoid the PERFORM PARA-NAME THRU PARA-EXIT when the THRU construct is unnecessary. In the following example, PERFORM PARA-NAME achieves the same result more efficiently.

EXAMPLE

Use:

```
    PERFORM PARA-NAME.
        .
        .
        .
    PARA-NAME.
        MOVE. . .
        ADD . . .
   NEXT-PARA.
```

Rather than:

```
    PERFORM PARA-NAME THRU PARA-EXIT.
        .
        .
        .
    PARA-NAME.
          MOVE. . .
          ADD . . .
    PARA-EXIT.
         EXIT.
NEXT-PARA.
```

13.3 ARITHMETIC OPERATIONS

The following statements illustrate ways to optimize code when using arithmetic statements.

- Use binary (COMP-1) data definitions. Binary is 2.5 to 10 times faster than COMP or DISPLAY.
- Do not mix binary and nonbinary operands because mixing slows arithmetic operations. Use constant data items or hexadecimal literals instead of decimal literals with COMP-1.

EXAMPLE

With:

```
    77 XYZ PIC 9(5) COMP-1.
    77 ONEX PIC 9(5) COMP-1 VALUE 1.
```

Use:

```
    ADD ONEX TO XYZ.
```

Rather than:

```
    ADD 1 TO XYZ.
```

The reason for the above is that all decimal literals are stored as type DISPLAY. Arithmetic is performed in the mode of the highest operand in the hierarchy:

```
3 DISPLAY
2 COMP
1 COMP-1
```

The following example provides further information regarding the handling of arithmetic operations.

EXAMPLE

With:

```
77 XYZ PIC 9(5)V9(3) COMP.
77 ZZZ PIC 9(5) COMP-1.
```

Use:

```
ADD 1, ZZZ GIVING XYZ.
```

The above requires conversion of "ZZZ" to display, adds display "1" and "ZZZ", converts the answer to COMP, and places it in "XYZ".

The following additional facts should be noted:

- Rounding increases the execution time of an instruction by 6 to 22 percent.
- Within a given picture size, the magnitude and sign of the operands have minimal effect on execution times.
- Increasing picture size of COMP or COMP-1 slows down execution by 1.5 to 5 percent per character.

13.4 CONTROL OPERATIONS

The logical sequence of program flow can be made more efficient in the following ways:

- Use "PERFORM <some-function> LIMIT TIMES" rather than a counter loop.

EXAMPLE

Use:

```
    PERFORM LOOP LIMIT TIMES.
LOOP.
    <some function>
```

Rather than:

```
    MOVE 1 TO K.
LOOP.
    <some-function>
    ADD 1 TO K.
    IF K NOT = LIMIT GO TO LOOP.
```

- Use "PERFORM ⟨some-function⟩ LIMIT TIMES" rather than "PERFORM-VARYING" unless the variable is used for subscripting in the procedure.

EXAMPLE

Use:

```
    PERFORM XYZ VARYING K FROM 1 BY
    1 UNTIL K > 30.
    .
    .
    .
XYZ. MOVE XM (K) TO L.
```

Rather than:

```
MOVE 1 TO K.
PERFORM XYZ 30 TIMES.
.
.
.
XYZ. MOVE XM (K) TO L.
ADD 1 TO K.
```

- Use "GO TO DEPENDING ON" rather than any N-WAY-BRANCH structure with an IF statement, whenever N ≥ 3. For example:

EXAMPLE

Use:

```
GO TO L1, L2, L3, L4 DEPENDING ON J.
```

Rather than:

```
IF J=1 GO TO L1.
IF J=2 GO TO L2.
IF J=3 GO TO L3.
IF J=4 GO TO L4.
```

- Use modular programming techniques as permitted by the CALL statement. CALLs are efficient.

- When IF statements are mixed with COMP, COMP-1, and/or DISPLAY, more time is required. Avoid statements such as IF COMP1-ITEM ≡ 4 GO TO MN.

- Avoid using CONTROL when possible.

EXAMPLE

With:

```
77 K PIC 9(5) COMP=1.
01 XYZ.
03 MMM PIC 9(5)V99 OCCURS 10.
```

Use:

```
MOVE ZEROS TO XYZ.
```

Rather than:

```
    PERFORM ZONK VARYING K FROM 1
       BY 1 UNTIL > 10.
ZONK. MOVE 0 TO MMM (K).
```

- Combine DISPLAY statements into one literal to reduce the likelihood of swapping.

EXAMPLE

Use:

```
DISPLAY "DATE OF BIRTH - MO:   DA:   YR:  ".
```

Rather than:

```
DISPLAY "DATE OF BIRTH"   LINE 17.
DISPLAY "MO"              LINE 17 POSITION 17.
DISPLAY "DA"              LINE 17 POSITION 23.
DISPLAY "YR"              LINE 17 POSITION 29.
```

- Use the DISPLAY/ACCEPT statement once instead of repeating the statement several times.

EXAMPLE

Use:

```
DISPLAY NAME              LINE 1  POSITION 1
SEX              LINE 1   POSITION 23
DOB-MO           LINE 1   POSITION 25
DOB-DA           LINE 1   POSITION 28
DOB-YR           LINE 1   POSITION 31.
```

Rather than:

```
DISPLAY NAME              LINE 1  POSITION 1.
DISPLAY SEX               LINE 1  POSITION 23.
DISPLAY DOB-MO            LINE 1  POSITION 25.
DISPLAY DOB-DA            LINE 1  POSITION 28.
DISPLAY DOB-YR            LINE 1  POSITION 31.
```

13.5 MOVE OPERATIONS

Efficiency in your use of the MOVE statement can be enhanced by applying the following suggestions:

- Move larger, rather than smaller, groups of characters whenever possible. The larger the group moved, the shorter the time used per character. Moving a 1000-character group is 30 times faster than moving a single character 1000 times.

- Use the MOVE statement for moving one source to multiple destinations. The instruction need not be interpreted multiple times, nor does the source operand need to be set up multiple times.

EXAMPLE

Use:

```
MOVE 0 TO A, B, C.
```

Rather than:

```
MOVE 0 TO A.
MOVE 0 TO B.
MOVE 0 TO C.
```

- Use an alphabetic MOVE statement in preference to numeric or ASCII string moves except with very small ASCII strings.

- When performing multiple moves, combine literals and move to a group area. (Literals can be combined; data items cannot be combined.)

EXAMPLE

With:

```
01  DATES
    02  MO    PIC XX.
    02  DA    PIC XX.
    02  YR    PIC XX.
```

Use:

```
MOVE "042545" TO DATES.
```

Rather than:

```
MOVE "04" TO MO.
MOVE "25" TO DA.
MOVE "45" TO YR.
```

13.6 I/O OPERATIONS

The following considerations relate to the response time in performing data I/O operations to a file or device.

- When possible execute:
  - a READ and not a READ . . . INTO. READ is 30 percent faster. If INTO is required, it is 20 percent faster than a READ followed by a MOVE.
  - a WRITE and not a WRITE . . . FROM. WRITE is 30 percent faster unless a MOVE is required to complete the FROM.
- Record-level locking adds only about 5 percent to the I/O operation.
- Increasing a file's blocking size (physical record size) beyond a factor of about 10 has a minimal effect. Always use a multiple of the ADU size on the disk as the block size.

The following statements describe ways to enhance the response time in performing I/O operations to key indexed files (KIFs):

- If you make numerous insertions, occasionally rebuild KIFs to decrease access time.
- When building a KIF, specify the key that will be accessed most often as the primary key. This will produce a file structure with the fastest sequential access time for that key.
- Keep the number of keys in a file small to keep down the time needed to build the file. Note: the number of keys in a file has little effect on the access time for reading.
- Keep keys short to reduce access time.
- When possible, use sequential input to build files. Files built from sequential input are smaller than files built by random record insertions, and require less access time. Also, the time needed to build a KIF from a sequential file sorted on the primary key is less than for a randomly ordered file.
- When possible, keep the sequential input file in sort order (as when the input file is the result of CKS). This reduces the time needed to build the KIF.

Appendix A

Keycap Cross-Reference

Generic keycap names that apply to all terminals are used for keys on keyboards throughout this manual. This appendix contains specific keyboard information to help you identify individual keys on any supported terminal. For instance, every terminal has an Attention key, but not all Attention keys look alike or have the same position on the keyboard. You can use the terminal information in this appendix to find the Attention key on any terminal.

The terminals supported are the 931 VDT, 911 VDT, 915 VDT, 940 EVT, the Business System terminal, and hard-copy terminals (including teleprinter devices). The 820 KSR has been used as a typical hard-copy terminal. The 915 VDT keyboard information is the same as that for the 911 VDT except where noted in the tables.

Appendix A contains three tables and keyboard drawings of the supported terminals.

Table A-1 lists the generic keycap names alphabetically and provides illustrations of the corresponding keycaps on each of the currently supported keyboards. When you need to press two keys to obtain a function, both keys are shown in the table. For example, on the 940 EVT the Attention key function is activated by pressing and holding down the Shift key while pressing the key labeled PREV FORM NEXT. Table A-1 shows the generic keycap name as Attention, and a corresponding illustration shows a key labeled SHIFT above a key named PREV FORM NEXT.

Function keys, such as F1, F2, and so on, are considered to be already generic and do not need further definition. However, a function key becomes generic when it does not appear on a certain keyboard but has an alternate key sequence. For that reason, the function keys are included in the table.

Multiple key sequences and simultaneous keystrokes can also be described in generic keycap names that are applicable to all terminals. For example, you use a multiple key sequence and simultaneous keystrokes with the log-on function. You log on by *pressing the Attention key, then holding down the Shift key while you press the exclamation (!) key.* The same information in a table appears as *Attention/(Shift)!*.

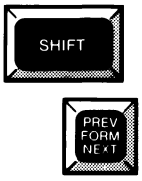
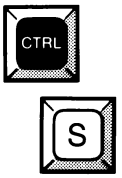
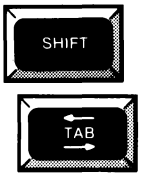
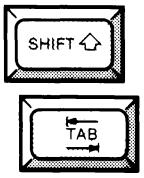
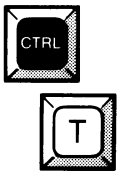
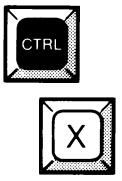
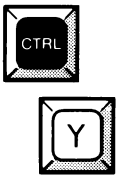
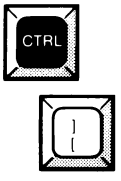
Table A-2 shows some frequently used multiple key sequences.

Table A-3 lists the generic names for 911 keycap designations used in previous manuals. You can use this table to translate existing documentation into generic keycap documentation.

Figures A-1 through A-5 show diagrams of the 911 VDT, 915 VDT, 940 EVT, 931 VDT, and Business System terminal, respectively. Figure A-6 shows a diagram of the 820 KSR.

2274834 (1/14)

Table A-1. Generic Keycap Names

Generic Name	911 VDT	940 EVT	931 VDT	Business System Terminal	820[1] KSR
Alternate Mode	None				None
Attention[2]					
Back Tab	None			None	
Command[2]					
Control					
Delete Character					None
Enter					
Erase Field					

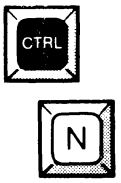
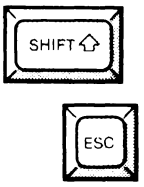
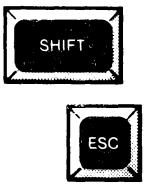
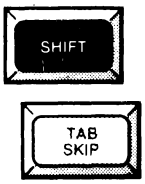
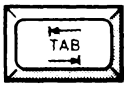
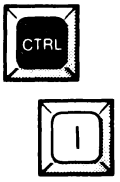
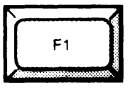
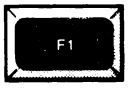
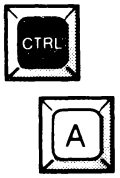
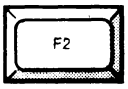
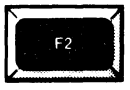
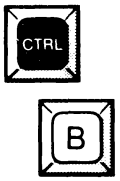
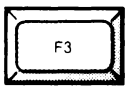
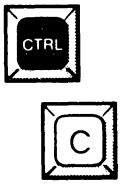
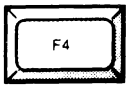
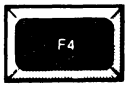
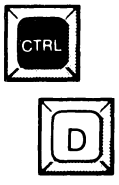
Notes:

[1]The 820 KSR terminal has been used as a typical hard-copy terminal with the TPD Device Service Routine (DSR). Keys on other TPD devices may be missing or have different functions.

[2]On a 915 VDT the Command Key has the label F9 and the Attention Key has the label F10.

2284734 (2/14)

Table A-1. Generic Keycap Names (Continued)

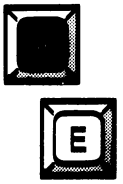
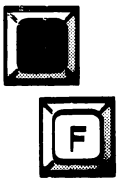
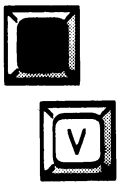
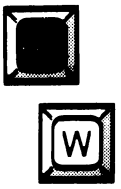
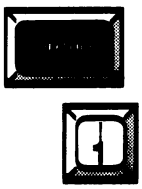
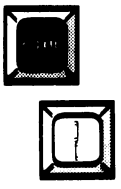
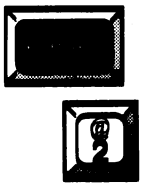
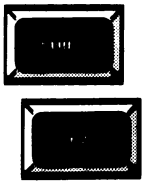
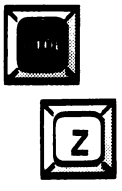
Generic Name	911 VDT	940 EVT	931 VDT	Business System Terminal	820[1] KSR
Erase Input					
Exit					
Forward Tab					
F1					
F2					
F3					
F4					

Notes:

[1]The 820 KSR terminal has been used as a typical hard-copy terminal with the TPD Device Service Routine (DSR). Keys on other TPD devices may be missing or have different functions.

2284734 (3/14)

Table A-1. Generic Keycap Names (Continued)

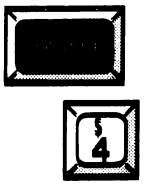
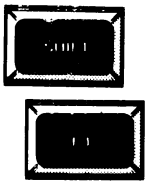
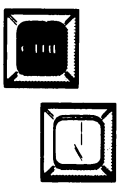
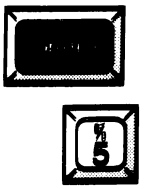
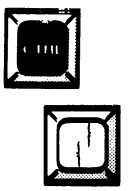
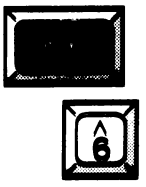
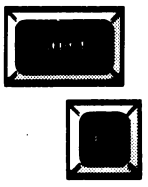
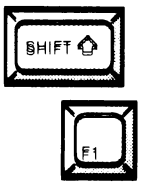
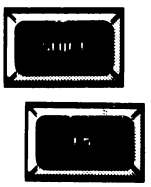
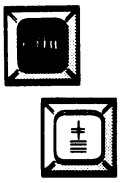
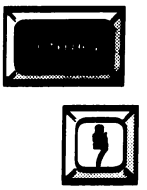
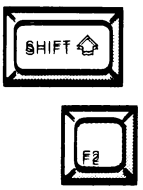
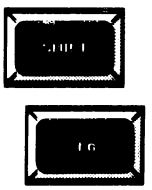
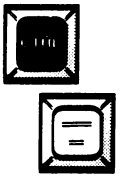
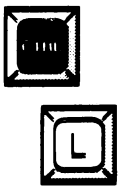
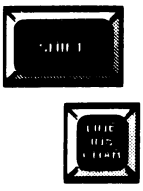
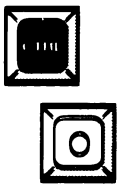
Generic Name	911 VDT	940 EVT	931 VDT	Business System Terminal	820[1] KSR
F5					
F6					
F7					
F8					
F9					
F10					

Notes:

[1]The 820 KSR terminal has been used as a typical hard-copy terminal with the TPD Device Service Routine (DSR). Keys on other TPD devices may be missing or have different functions.

2284734 (4/14)

Table A-1. Generic Keycap Names (Continued)

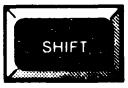
Generic Name	911 VDT	940 EVT	931 VDT	Business System Terminal	820[1] KSR
F11					
F12					
F13					
F14					
Home					
Initialize Input					

Notes:

[1]The 820 KSR terminal has been used as a typical hard-copy terminal with the TPD Device Service Routine (DSR). Keys on other TPD devices may be missing or have different functions.

2284734 (5/14)

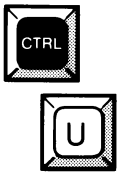
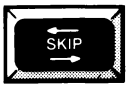
Table A-1. Generic Keycap Names (Continued)

Generic Name	911 VDT	940 EVT	931 VDT	Business System Terminal	820[1] KSR
Insert Character					None
Next Character	or 				None
Next Field					None
Next Line					or
Previous Character	or				None
Previous Field					None

Notes:
[1]The 820 KSR terminal has been used as a typical hard-copy terminal with the TPD Device Service Routine (DSR). Keys on other TPD devices may be missing or have different functions.

2284734 (6/14)

Table A-1. Generic Keycap Names (Continued)

Generic Name	911 VDT	940 EVT	931 VDT	Business System Terminal	820[1] KSR
Previous Line					
Print					None
Repeat		See Note 3	See Note 3	See Note 3	None
Return					
Shift					
Skip					None
Uppercase Lock					

Notes:

[1]The 820 KSR terminal has been used as a typical hard-copy terminal with the TPD Device Service Routine (DSR). Keys on other TPD devices may be missing or have different functions.

[3]The keyboard is typamatic, and no repeat key is needed.

2284734 (7/14)

Table A-2. Frequently Used Key Sequences

Function	Key Sequence
Log-on	Attention/(Shift)!
Hard-break	Attention/(Control)x
Hold	Attention
Resume	Any key

Table A-3. 911 Keycap Name Equivalents

911 Phrase	Generic Name
Blank gray	Initialize Input
Blank orange	Attention
Down arrow	Next Line
Escape	Exit
Left arrow	Previous Character
Right arrow	Next Character
Up arrow	Previous Line

2284734 (8/14)

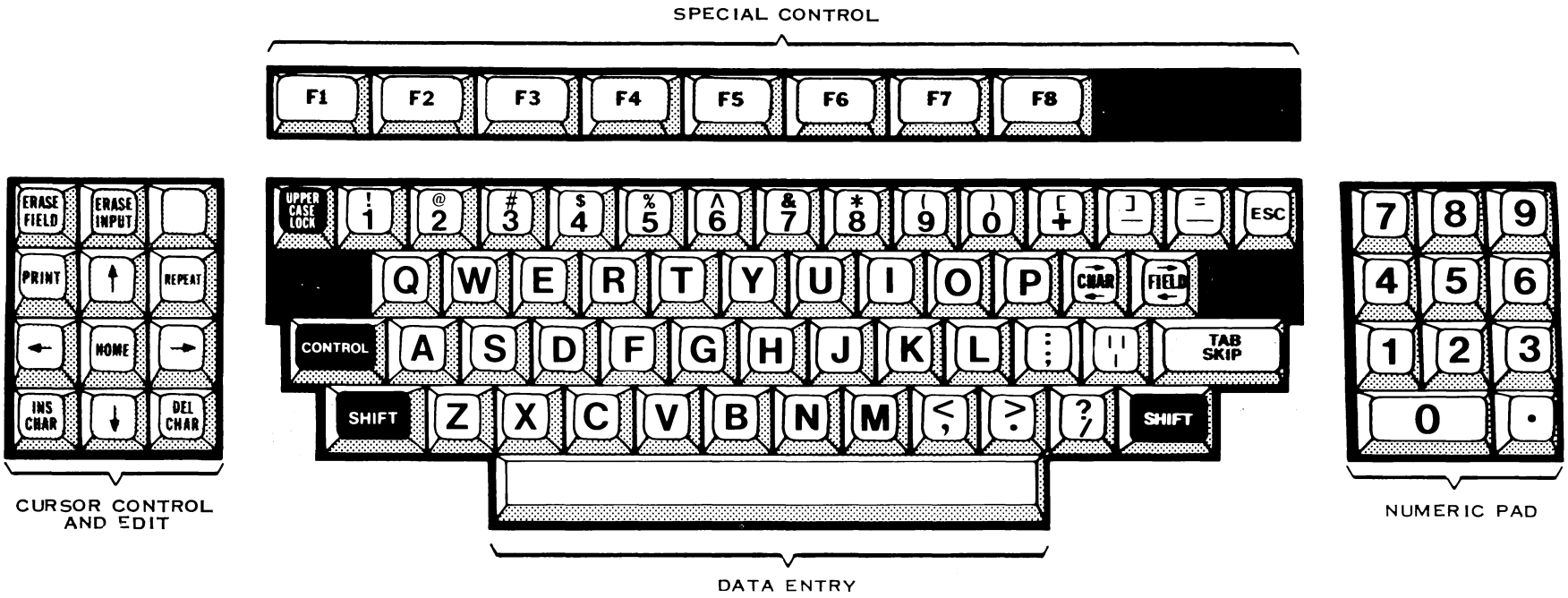


Figure A-1. 911 VDT Standard Keyboard Layout

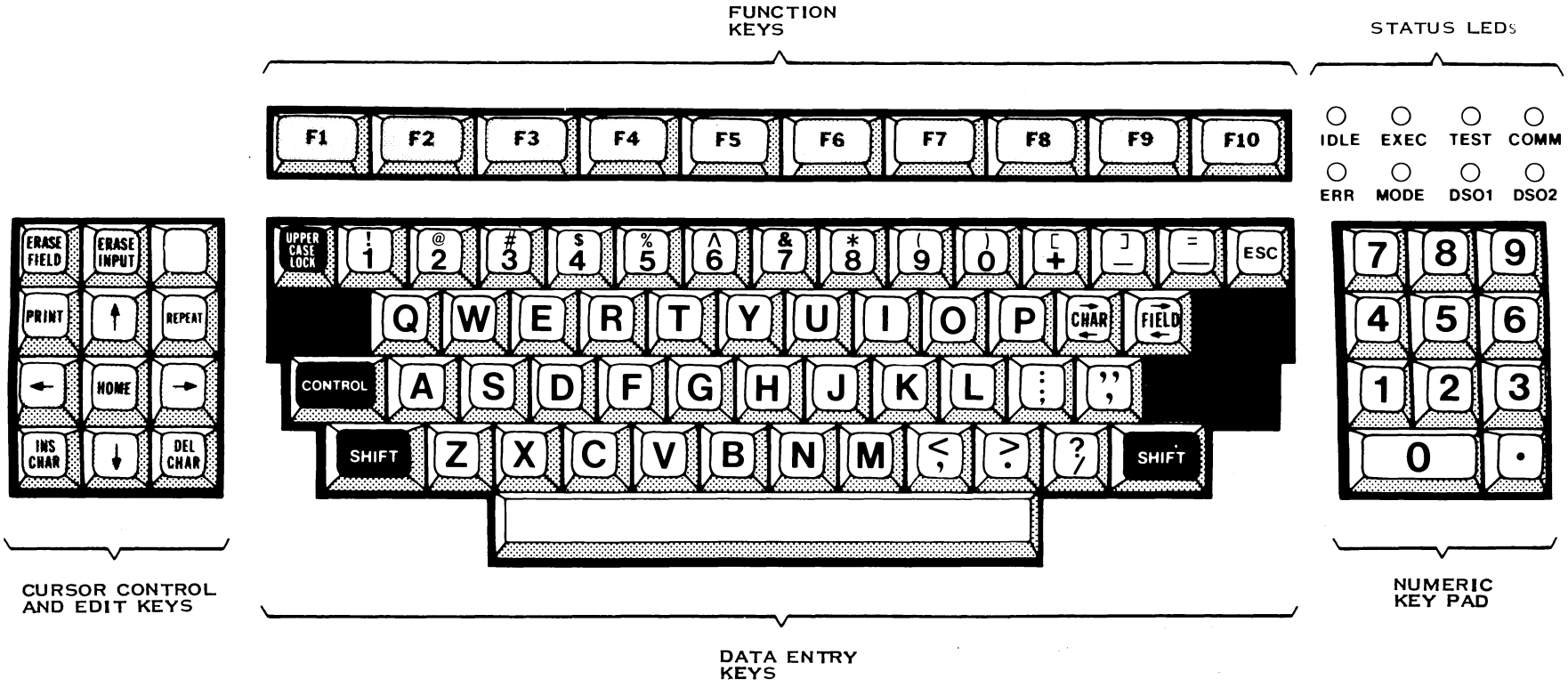
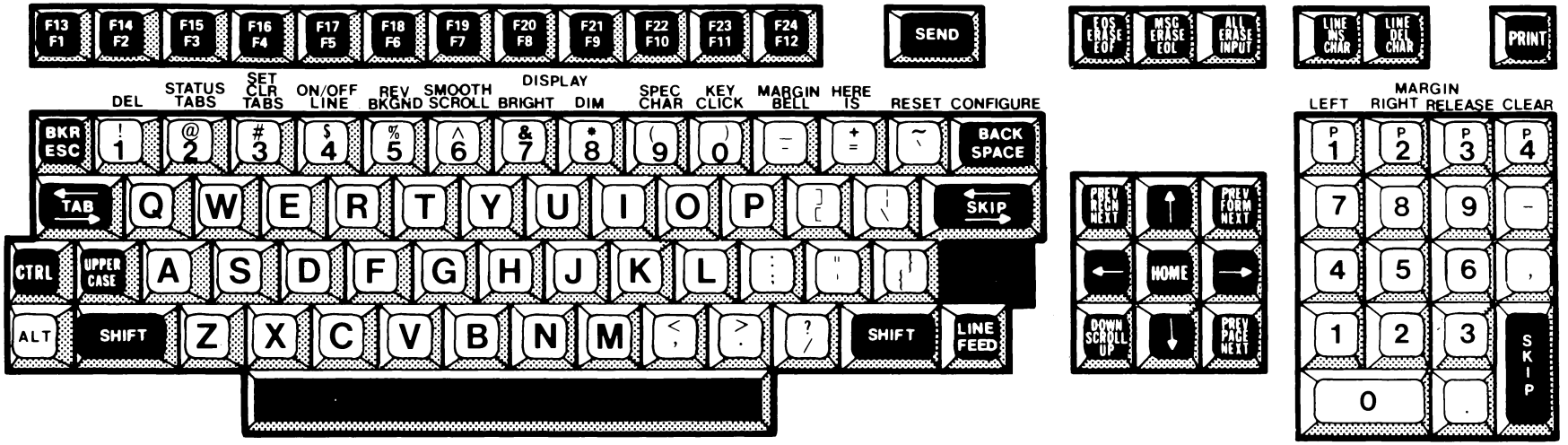
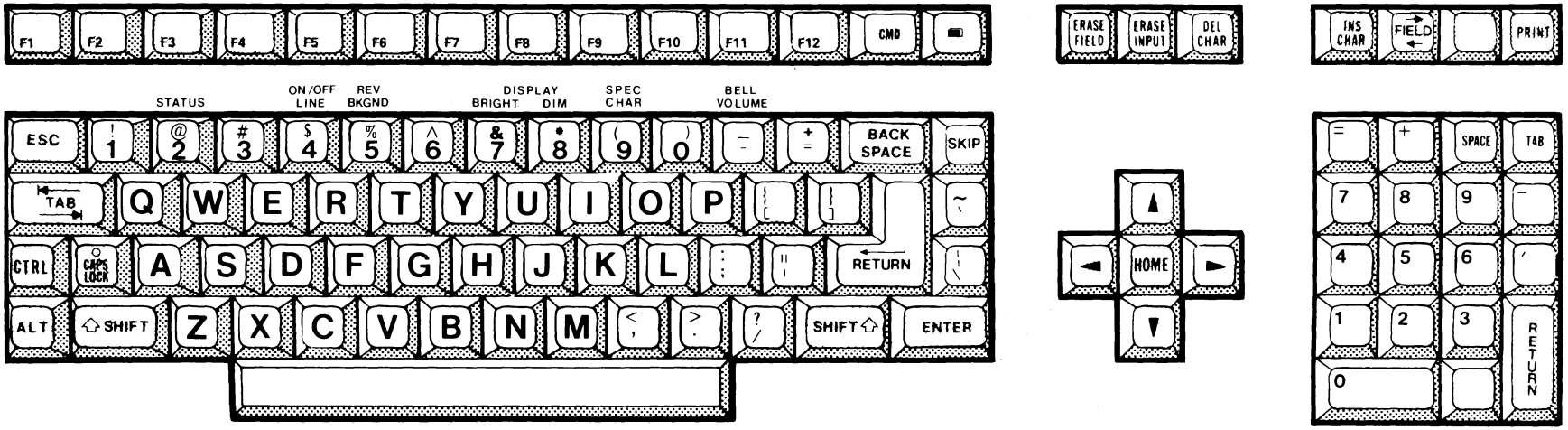


Figure A-2. 915 VDT Standard Keyboard Layout



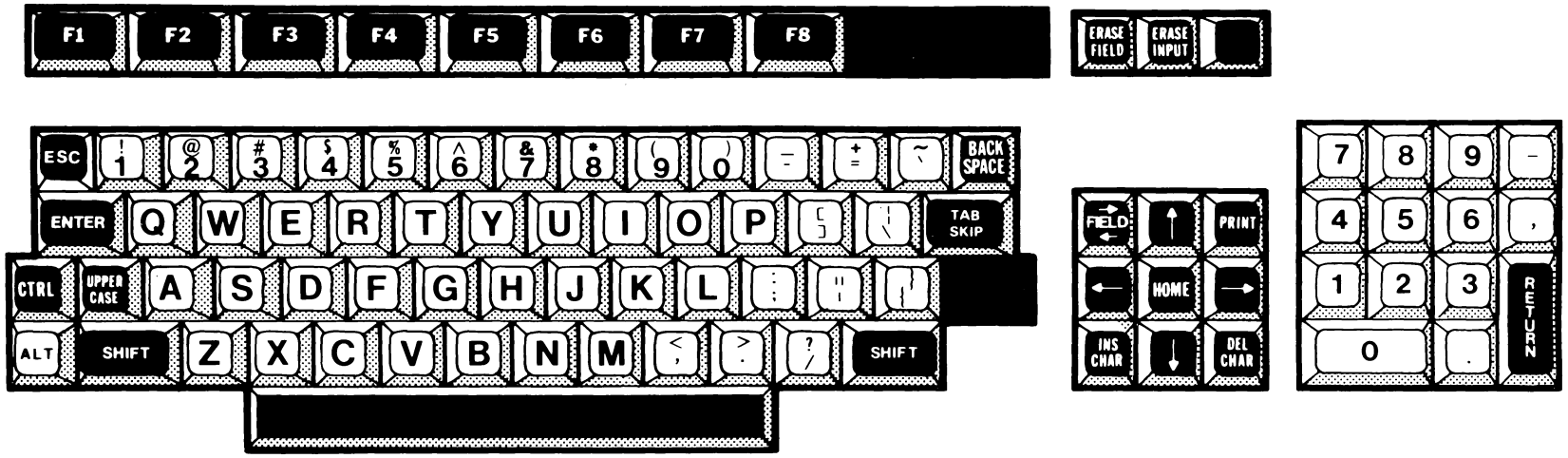
2284734 (11/14)

Figure A-3. 940 EVT Standard Keyboard Layout



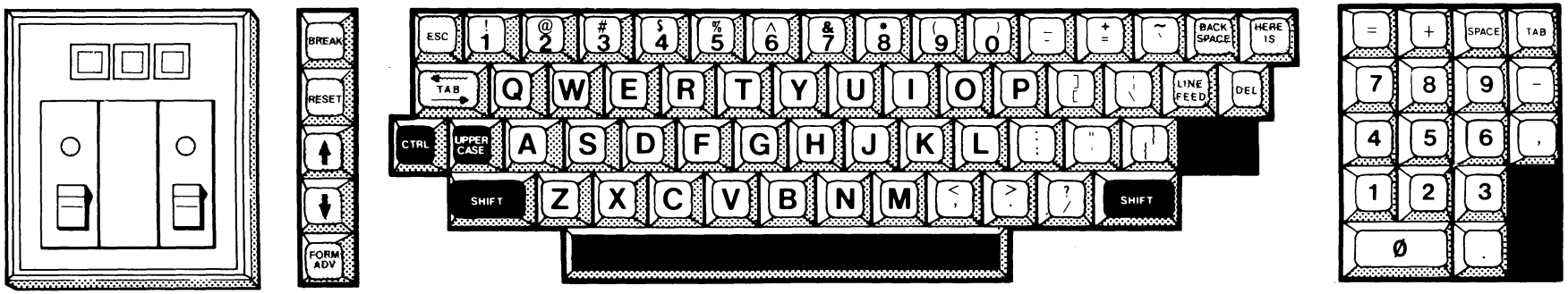
2284734 (12/14)

Figure A-4. 931 VDT Standard Keyboard Layout



2284734 (13/14)

Figure A-5. Business System Terminal Standard Keyboard Layout



2284734 (14/14)

Figure A-6. 820 KSR Standard Keyboard Layout

Appendix B

COBOL Compiler Error Messages

B.1 USER ERROR MESSAGES

As source constructs are scanned, they are checked for syntax and semantic errors. If an error interrupts the scanning, text is ignored until a recovery point is found and a resume message is printed. Recovery points are chosen to minimize the amount of unanalyzed text without producing irrelevant error messages. A warning message referencing the appropriate undermark ($ in the examples in this appendix) indicates the point at which scanning resumes. In any case, the constructs at fault are undermarked and error messages are listed when the source line is printed. The error message includes either Es or Ws, indicating error or warning. For example,

```
004030     02 STOCK  PIC  9(16)PPP COMPUTATIONAL.
                                   $
1) PICTURE *E*E*E*E*E*E*E*E*E*E*E*E*E*E*E*E*E*E*E*E*E*E*E*E*E*E*E*E*E*E*E*E*E*E*E*E*E*E*E*E*E*E*E*E*E*E*E
```

The 1 at the beginning of the error message in the preceding example indicates that the first undermark from the left (the only undermark in this example) indicates the position at which the error was identified. The message PICTURE indicates invalid PICTURE syntax (semantic number size). The string of alternating asterisks and Es identifies it as an error message. This error did not interrupt the scan of the source statement.

The following is an example of an error that interrupted the scan:

```
005040     02    PART PIC  X(4BX(10)) SYNC.
                               $      $

1) SYNTAX *E*E*E*E*E*E*E*E*E*E*E*E*E*E*E*E*E*E*E*E*E*E*E*E*E*E*E*E*E*E*E*E*E*E*E*E*E*E*E*E*E*E*E*E*E*E*E
2) SCAN RESUME *W*W*W*W*W*W*W*W*W*W*W*W*W*W*W*W*W*W*W*W*W*W*W*W*W*W*W*W*W*W*W*W*W*W*W*W*W*W*W*W*W*W*W
```

The 1 at the beginning of the first message specifies that the first undermark from the left indicates the position at which the error was identified. The message SYNTAX indicates a syntax error. The string of alternating asterisks and Es identifies it as an error message. The 2 at the beginning of the second message indicates that the second undermark is associated with this message. The message SCAN RESUME indicates that the scan of the statement resumed at the second undermark (the S in the keyword SYNC). The alternating asterisks and Ws identify the message as a warning.

Global errors such as undefined paragraph names and illegal control transfers are listed with the program summary at the end of the source listing. The following shows a program summary error message.

```
VALUE ERROR *E*E*E*E*E*E*E*E*E*E*E*E*E*E*E*E*E*E*E*E*E DATA-NAME
```

The message VALUE ERROR indicates that the data item listed to the right has been assigned a value which is too large. The alternate asterisks and Es identify the message as an error message.

The error messages described in this paragraph are printed in the compiler listing. Figure B-1 shows a listing that includes errors and error messages. Table B-1 lists the error messages.

```
 DXCBL         L.R.V YY.DDD COMPILED:MM/DD/YY HH:MM:SS  OPT=M      PAGE    2
LINE  DEBUG PG/LN  A...B.....................................................

   1               IDENTIFICATION DIVISION.
   2               PROGRAM-ID.
   3                   TI-DATA-TEST.
   4               ENVIRONMENT DIVISION.
   5               CONFIGURATION SECTION.
   6               SOURCE-COMPUTER.
   7                   TI-990-10.
   8               OBJECT-COMPUTER.
                   $
   1) SYNTAX *E*E*E*
```

```
  1) SCAN RESUME *W*W*W*
```

```
VALUE ERROR *E*E*E*E*E*E*E*E*E*E*E*E*E*E*E*E*E*E*E  ERROR-VALUE

REDEFINES ERROR *E*E*E*E*E*E*E*E*E*E*E*E*E*E*E*E*E  VALUE-2

STATUS ERROR  *E*E*E*E*E*E*E*E*E*E*E*E*E*E*E*E*E*E  CARD-FILE

RELATIVE KEY ERROR  *E*E*E*E*E*E*E*E*E*E*E*E*E*E*E  DISC-FILE-RAN-REL

READ ONLY BYTE SIZE =  >0058

READ/WRITE BYTE SIZE = >00CC

TOTAL BYTE SIZE =      >0124

  16 ERRORS

   7 WARNINGS
```

Figure B-1. COBOL Compiler Listing With Error Message Examples (Sheet 3 of 3)

Table B-1. COBOL Compiler User Error Messages

Message	Cause
ACCESS CLASH	Nonsequential access given for sequential file.
BLANK WHEN ZERO	BLANK WHEN ZERO clause given for nonnumeric or group item.
CLASS	Referenced data type invalid for this form of class test.
COPY	COPY statement failed because of permanent error associated with the undermarked text-name.
CORRESPONDING	The referenced item has no subordinate items for CORRESPONDING operation.
DATA OVERFLOW	The data area (working storage and literals) is larger than 65,535 bytes long.
DATA TYPE	The data type is invalid for this reference.
DEVICE CLASH	Random characteristics given to nonrandom device.
DEVICE TYPE	OPEN or CLOSE mode inconsistent with device type.
DOUBLE DECLARATION	Multiple declaration of a file or identifier attribute.
DOUBLE DEFINITION	Multiple definition of an identifier.

Table B-1. COBOL Compiler User Error Messages (Continued)

Message	Cause
DUPLICATE	Warning only. Multiple USE procedure declared for same function.
FILE DECL ERROR	Error in file declaration.
FILE NAME ERROR	Error in defining variable for external file pathname (must be ANS or GRP).
FILE NAME REQUIRED	File name not given as reference in I/O verb.
FILE RECORD KEY ERROR	The referenced file name has a RECORD KEY that is incorrectly qualified or that is not defined as a data item of the alphanumeric category within a record-description-entry associated with that file name.
FILE RECORD SIZE ERROR	The referenced file name has a declared record size that either conflicts with the actual record description, is a relative organization file with variable length records, or is odd length.
FILE STATUS ERROR	The referenced file name has a status item that is either incorrectly qualified, not defined in the WORKING-STORAGE SECTION, or not a two-character alphanumeric item.
FILE TYPE	Access or organization of file conflicts with undermarked statement.
FILLER LEVEL	A nonelementary FILLER item is declared.
GROUP CLASH	USAGE or VALUE clause of group member conflicts with same clause for group.
GROUP VALUE CLASH	An item subordinate to a group with the VALUE IS clause is described with the SYNCHRONIZED, JUSTIFIED, or USAGE (other than USAGE IS DISPLAY) clause.
IDENTIFIER	Identifier reference is incorrectly constructed, or the identifier has an invalid or double definition.
ILLEGAL ALTER	An ALTER statement references an unalterable paragraph, or violates the rules of COBOL segmentation.
ILLEGAL PERFORM	A PERFORM statement references an undefined or incorrectly qualified paragraph, or the reference violates the rules of COBOL segmentation.
INVALID ID	An invalid identifier is referenced (check declaration).
INVALID PARAGRAPH	An invalid paragraph name is referenced.

Table B-1. COBOL Compiler User Error Messages (Continued)

Message	Cause
JUSTIFY	JUSTIFY clause given in conflict with other attributes.
KEY REQUIRED	Relative key not declared for random access relative file, or record key not declared for KIF.
LABEL	Presence or absence of label record conflicts with device standards.
LEVEL	Level number given is invalid either intrinsically or because of position within a group.
LINKAGE	An identifier in the USING clause of the PROCEDURE header is not a linkage item, or a statement references a linkage item not subordinate to an identifier in the USING clause of the PROCEDURE header.
LITERAL VALUE	Literal value given is incorrect in context.
MOVE	Operands of MOVE verb specify an invalid move.
MUST BE INTEGER	Context requires an integer.
MUST BE PROCEDURE	Context requires procedure name either as reference or definition, or the reference must be a nondeclarative procedure name.
MUST BE SECTION	Context requires procedure name to be section.
NAME ERROR	The referenced file name has an invalid external file name declaration.
NESTING	Illegal nesting of condition that is not an IF condition.
NOT IN REDEFINE	VALUE IS clause given in REDEFINES item.
OCCURS	OCCURS clause given at invalid level (only three nested levels of occurs acceptable).
OCCURS-VALUE CLASH	VALUE IS and OCCURS in effect for the same item.
PICTURE	Invalid picture syntax.
PICTURE-BWZ CLASH	BLANK WHEN ZERO clause given when picture contains asterisk character.
PICTURE-USAGE CLASH	USAGE clause or implied usage conflicts with usage implied by picture.
PROCEDURE INDEPENDENCE	PERFORM given for procedures in independent segments not in the current segments.

Table B-1. COBOL Compiler User Error Messages (Continued)

Message	Cause
PROGRAM OVERFLOW	The instruction area is larger than 32,767 bytes long.
RECORD KEY	Record key declared for other than an indexed organization file, or a START statement KEY phrase references a data item not aligned on the declared leftmost byte of the key.
RECORD REQUIRED	File record must be referenced for WRITE or REWRITE verb.
REDEFINES	REDEFINES given within an OCCURS or not redefining the last allocated item.
REDEFINES ERROR	The referenced data-name redefines an item that does not have the same number of character positions and that is not level 01.
REFERENCE INVALID	Reference given is not valid in context.
RELATION	Operands of relation test are incompatible.
RELATIVE KEY	Relative key declared for other than a relative organization file, or a START statement key phrase references a data item other than the declared key.
RELATIVE KEY ERROR	The referenced file name has a RELATIVE KEY that is incorrectly qualified, is defined in a record description associated with that file name, or is not defined as an unsigned integer.
RESERVED WORD CONFLICT	A COBOL reserved word or symbol is given where a user word is required. In the summary, this is only a warning about an ANSI COBOL reserved word that is not a TI COBOL reserved word.
SCAN RESUME	Warning only. Scanning was terminated at previous error message and resumes at undermarked character.
SECTION CLASH	A VALUE IS clause appears in the FILE or LINKAGE section.
SEGMENT	Warning only. Segment number given in an independent segment is not the same as the current segment or the number of new independent segments.
SEPARATOR	Warning only. Redundant punctuation.
SIGN	SIGN clause given in conflict with usage and picture.

Table B-1. COBOL Compiler User Error Messages (Continued)

Message	Cause
SIZE	Warning only. Size of data referenced not correct for context.
SIZE ERROR	Declared size of record conflicts with the present reference.
STATUS ERROR	The referenced file name has a status item which is incorrectly qualified, is not defined in the WORKING-STORAGE SECTION, or is not a two-character alphanumeric item.
SUBSCRIPT	Incorrect number of subscripts or indexes given for a reference.
SYNC	Synchronized clause given for a group item.
SYNTAX	Character or reserved word given is incorrect for context.
UNDEFINED	File referenced in FD paragraph was not defined.
UNDEFINED PROCEDURE	A GO TO statement references an undefined or incorrectly qualified paragraph.
USE REQUIRED	Compiler expects a USE statement. This error may indicate that no END DECLARATIVES statement is present in the program.
USING COUNT	Warning only. The item count in the USING list of a CALL statement is different from that of the first reference to the same program name.
VALUE	VALUE IS clause given in conflict with other declared attributes.
VALUE ERROR	Value given in VALUE IS required truncation of nonzero digits.
VARIABLE RECORD	Warning only. A READ . . . INTO references a file with variable record sizes.

B.2 SYSTEM ERROR MESSAGES

Other types of errors prevent proper execution of the COBOL compiler. When any of these errors occur, the system displays an error message and terminates the attempt to execute the compiler. Table B-2 lists these messages.

Table B-2. COBOL Compiler System Error Messages

Message	Meaning	Recovery
COBOL COMPILER ABNORMAL TERMINATION	The compiler has unsuccessfully completed creation of the object module from the supplied source program.	Refer to listing to determine reason.
COBOL COMPILER BUG	The compiler was unable to complete the compilation due to a problem internal to the compiler.	Refer problem to system programmer.
COBOL COMPILER FATAL ERROR-END VECTOR TAKEN	The operating system has detected a fatal error and has terminated the compile	Refer to the *DX10 Messages and Codes Reference Manual* for possible errors.
COBOL COMPILER OVERLAY LOAD ERROR	An attempt to load a compiler overlay resulted in an error.	Refer problem to system programmer.
COBOL COMPILER TCA ERROR	The system has detected an error condition in the Task Communication Area (TCA).	Refer problem to system programmer.
COBOL COMPILER TLF ERROR	The system has detected an error condition in the Terminal Local File (TLF) processing.	Refer problem to system programmer.
COPY FILE CLOSE ERROR	An attempt to close the copy file resulted in an error.	Refer problem to system programmer.
COPY FILE OPEN ERROR	An attempt to open the copy file resulted in an error.	Refer problem to system programmer.
COBOL POINTER OVERFLOW	Program too large; has too many variable names.	Make program smaller.
COBOL PROGRAM OVERFLOW	Insufficient memory space is available for storing the program.	Program exceeds task addressability. Program area must be smaller.
FATAL SYNTAX ERROR OR NULL PROGRAM	A syntax error in the source code resulted in a fatal error for the compiler.	Examine source statement and correct syntax. Check for blank line at beginning or end of source code.
I/O ERROR	An error has occurred with one of the I/O devices.	Use system error code to determine reason for error.
INVALID COMPILE OPTION	One or more of the selected compiler option characters entered for this compilation is not a defined option character.	Correct invalid option character. (Compiler will function properly without correction, ignoring the invalid character.)

Table B-2. COBOL Compiler System Error Messages (Continued)

Message	Meaning	Recovery
LISTING FILE ERROR	An error has occurred while transferring data to the listing device.	Use system error code to determine reason for error.
MEMORY REQUESTED EXCEEDS SYSTEM CAPACITY	The memory resources of the system are not sufficient to provide the quantity of memory required.	Allow other tasks to terminate prior to recompiling or delete the X option.
OBJECT FILE ERROR	An error has occurred while transferring data to the object file.	Use system error code to determine reason for error.
ROLL SPILL FILE ERROR	An error has occurred during I/O to the ROLL SPILL file.	System buffer size is possibly too small; refer problem to a system analyst.
SOURCE FILE ERROR	An error has occurred while transferring data from the source file.	Use system error code to determine reason for error.
TOO MANY NESTED	A COPY statement has been read that would result in more than five nesting levels.	Modify source program to limit number of nesting levels of COPY statements to five or less.

B.3 INFORMATIVE MESSAGES

Table B-3. COBOL Compiler Informative Messages

Message	Cause
INFO-ROLL xxxx SPILLING TO DISK	The rolls used by the compiler have exceeded the available memory for the task. The roll indicated has been moved to disk. An increase in compile time will occur after spilling is performed. This informative message has been added to assist in producing programs with optimal compile times.

Appendix C

COBOL Run-Time Error Messages

COBOL provides two types of run-time error messages. Table C-1 lists both types. The first type is related to object code resulting from incorrect source statements. Messages of this type are displayed in the following format:

```
COBOL (ERROR MESSAGE) AT: xxyyyy IN nnnnnn
```

The ERROR MESSAGE portion defines the error and is one of the messages listed in Table C-1. The xx portion represents the segment number of the segmented module, and the yyyy portion represents the address of the statement at which the error was detected. The address corresponds to the address shown in the DEBUG column of the compiler listing. The nnnnnn is the first six characters of the program name, as specified in the PROGRAM-ID statement.

The second type of run-time error messages relates to system errors. The system displays the message and terminates the execution attempt.

The message is as follows:

Explanation:
The object file contains an end-of-file prior to data required for executing the program.

User action:
Verify that the correct object file was read and that it was correctly positioned. If necessary, recompile.

The run-time error message file is installed as .SMSG.COBOL. If at the time of execution the message file does not exist, the message format appears as follows:

```
COBOL--INTERNAL CODE >eeee xxyyyy;nnnnnn
```

The eeee portion represents the number of the message requested and the xxyyyy;nnnnnn portions are as previously defined.

Table C-1. COBOL Run-Time Error Messages

UNEXPECTED END OF FILE ON OBJECT FILE

Explanation:
The object file contains an end-of-file prior to data required for executing the program.

User action:
Verify that the correct object file was read and that it was correctly positioned. If necessary, recompile.

ERROR READING OBJECT FILE, CODE = ?1

Explanation:
An error was detected while reading the object file. The code indicates the relative record number in the object file where the error occurred.

User action:
Recompile and relink.

INVALID TAG, CODE = ?1

Explanation:
A tag character in the object file was not a valid tag. The code indicates the relative record number in the object file where the invalid tag was found.

User action:
Verify that the correct object file was read and that it was correctly positioned. If necessary, recompile. Look for unresolved external references when using XCP or XCPF commands.

CHECKSUM ERROR, CODE = ?1

Explanation:
A checksum in the object file was incorrect. The code indicates the relative record number in the object file where the checksum error was found.

User action:
Verify that the correct object file was read and that it was correctly positioned. If necessary, recompile.

BAD HEX VALUE ON OBJECT FILE, CODE = ?1

Explanation:
Object file contains a character that is not a hexadecimal value. The code indicates the relative record number in the object file where the invalid value was found.

User action:
Verify that the correct object file was read and that it was correctly positioned. If necessary, recompile.

Table C-1. COBOL Run-Time Error Messages (Continued)

LOAD ADDRESS OUT OF RANGE, CODE = ?1

Explanation:
The object file contains a load address that is not within the valid range.

User action:
Verify that the correct opject file was read, and that it was correctly positioned. If necessary, recompile.

UNABLE TO GET MEMORY

Explanation:
The system has insufficient memory space available for program requirements.

User action:
Allow other tasks to terminate. Rerun program withoutcompetition for system resources. Check object code size in the compiler object listing or linked object listing. If the program will not fit in the task address space, then the program must be restructured using segmentation or overlays.

INVALID OBJECT FILE TYPE, CODE = ?1

Explanation:
The object file type must be a relative record file when the program is compiled or linked. The code indicates the relative record number in the object file where the error was found.

User action:
Delete the object file and recompile and/or link edit.

END COBOL RUN

Explanation:
The program has completed.

User action:
None required.

UNABLE TO OPEN OBJECT FILE, CODE = ?1

Explanation:
The system encountered an error when it tried to open the object file. The error code is the DX10 internal SVC error code.

User action:
Refer to the table in the *DX10 Messages and Codes Reference Manual* where correspondence between the internal error and the SVC Message ID is given. Look at the internal error code >00xx, where xx is the DX10 SVC internal error given in the COBOL message. Verify that the correct object file was read and that it was correctly positioned. If necessary, recompile.

Table C-1. COBOL Run-Time Error Messages (Continued)

LOAD OVERLAY SVC ERROR

Explanation:
The system encountered an error when loading an independent segment.

User action:
Check that the link control file is correct.

COBOL DEBUGGER NOT AVAILABLE

Explanation:
The COBOL debugger is not linked with this task.

User action:
Relink the task with RCBTSKD instead of RCBTSK.

COBOL RUN-TIME TCA ERROR

Explanation:
The system encountered an error in the Task Communication Area (TCA) processing.

User action:
Refer to system programmer.

COBOL RUN-TIME MESSAGE ACCESS ERROR

Explanation:
The system encountered an error with the MESSAGE ACCESS NAME specified in the execution command prompt.

User action:
Ensure that the MESSAGE ACCESS NAME specified in the command prompt is not being used concurrently by multiple programs. Ensure that foreground-only programs are not being run in background. If a file name is specified, ensure that it is sequential.

CFD KEY COUNT DOES NOT MATCH FCB KEY COUNT

Explanation:
The number of keys declared in a program for a key indexed file does not match the number of keys in the file control block declared when the file was created.

User action:
Verify program description of key indexed file and modify program to match file or recreate file to match program.

Table C-1. COBOL Run-Time Error Messages (Continued)

CFD KEY FLAGS/LENGTH DOES NOT MATCH FCB KEY FLAGS/LENGTH

Explanation:
The key indexed file key attributes declared in a program do not match key attributes in the file control block declared when the file was created.

User action:
Check all keys for compatibility with file; specifically key length attribute, modifiable attribute, and duplicate attribute.

CFD KEY OFFSET DOES NOT MATCH FCB KEY OFFSET

Explanation:
The displacement of a key-indexed file declared in a program does not match the displacement in the file control block declared when the file was created.

User action:
Check all keys for compatibility with the file; specifically key displacements.

UNEXPECTED ERROR CONDITION

Explanation:
An unexpected error has occurred.

User action:
Refer to system programmer.

COBOL INVALID ADDRESS AT: ?1 IN ?2

Explanation:
The statement references a linkage item for which no corresponding parameter exists. The error occurred at the given statement in the indicated program.

User action:
Examine the USING parameter lists for correspondence in both sending and receiving programs. An invalid UNIT number in an ACCEPT or DISPLAY can also cause this error.

COBOL INVALID EDIT AT: ?1 IN ?2

Explanation:
Statement implies editing of data for which the PICTURE clause is in error. Normally occurs only when attempting to execute object code that contains compiler errors. The error occurred at the given statement in the indicated program.

User action:
Correct the statement and recompile.

Table C-1. COBOL Run-Time Error Messages (Continued)

COBOL INVALID DATA DESCRIPTOR AT: ?1 IN ?2

Explanation:
Statement references data for which the PICTURE clause is in error. Normally occurs only when attempting to execute object code that contains compiler errors. The error occurred at the given statement in the indicated program.

User action:
Correct PICTURE clause for data and recompile.

COBOL UNDEFINED LABEL REFERENCE AT: ?1 IN ?2

Explanation:
Statement transfers control to a paragraph that is undefined. The error occurred at the given statement in the indicated program.

User action:
Correct statement or source program and recompile.

COBOL INVALID INSTRUCTION AT: ?1 IN ?2

Explanation:
The statement resulted in an undefined object instruction. Normally occurs only when attempting to execute object code that contains compiler errors. The error occurred at the given statement in the indicated program.

User action:
Correct the statement and recompile.

COBOL OVERLAY LOAD AT: ?1 IN ?2

Explanation:
This message provides traceback when an error has occurred. The previous overlay was loaded by the given statement in the indicated program.

User action:
None required.

COBOL SUBSCRIPT RANGE ERROR AT: ?1 IN ?2

Explanation:
Statement contains a subscript that is not within proper range. The error occurred at the given statement in the indicated program.

User action:
Correct statement and recompile.

Table C-1. COBOL Run-Time Error Messages (Continued)

COBOL COMPILATION ERROR AT: ?1 IN ?2

Explanation:
Statement compiled in error. The error occurred at the given statement in the indicated program.

User action:
Correct statement and recompile.

COBOL STOP RUN AT: ?1 IN ?2

Explanation:
The program has completed with the execution of a STOP RUN statement at the given statement in the indicated program.

User action:
None required.

CALLED AT: ?1 IN ?2

Explanation:
This message provides traceback when an error has occurred. The previous program was called by the given statement in the indicated program.

User action:
None required.

CRT ?1 I/O ERROR = ?2,?3 AT: ?4 IN ?5

Explanation:
An error has occurred reading the characteristics of the given device.

User action:
Refer to system programmer.

COBOL I/O ERROR = ?1,?2 AT: ?3 IN ?4

Explanation:
I/O statement did not execute successfully, and no AT END or INVALID KEY clause was applicable. The first number represents the status key. The second number is the SVC internal error code. The error occurred at the given statement in the indicated program.

User action:
For explanation of the status key, refer to the *COBOL Programmer's Guide*, Section 12: Error Processing. For explanation of the SVC error code, refer to the table in the Messages and Codes Manual where correspondence between the internal error and the SVC Message ID is given. Look at the internal error code >00xx, where xx is the SVC internal error given in the COBOL message. Either write a USE procedure for the I/O or add the AT END or INVALID KEY clause to the statement and recompile.

Exceptions to the SVC internal error code: On an open operation when the status key is 97 (invalid record length), the second number is the record length of the file being opened. When the status key is 93 or 94 (invalid open function), the second number indicates that some condition other than a detected error resulted in this condition.

Table C-1. COBOL Run-Time Error Messages (Continued)

COBOL STOP "?1" AT: ?2 IN ?3

Explanation:
The program has completed with the execution of a STOP "literal" statement at the given statement in the indicated program.

User action:
None required.

PATHNAME: ?1

Explanation:
This message accompanies a COBOL I/O error message (C01F). The error occurred using the given pathname.

User action:
None required.

END ACTION TAKEN: TASK ERROR CODE = ?1; PC = ?2

Explanation:
The task error indicated by the code has occurred. The program counter where the error occurred is noted by the second code. A task error message is also logged to the system log.

User action:
Refer to the Error Messages and Codes section of your operating system reference manual for explanation.

Appendix D

COBOL Subroutine Library Package

D.1 INTRODUCTION

This appendix contains a general explanation for each of the subroutines in the COBOL subroutine library. These are listed in Table D-1.

Where a subroutine has multiple alternate entry points, a main module name is listed, followed by the alternate entry points. The subroutine is not accessible by the main module name, but only through the entry points. The names given for the latter in Table D-1 are assigned aliases and can be used to access the code. Where a subroutine has only a single entry point, the code can be accessed via the module name shown.

All data fields used as parameters to the COBOL subroutines MUST be aligned on word boundaries. This can be accomplished by making the parameter an 01-Level data item in the WORKING-STORAGE section of the program. There are no provisions in either the compiler or the run-time packages to test for this condition, since to include such provisions would increase program size, causing a space problem in some user programs. However, user failure to ensure word alignment can result in improper execution or erroneous results.

Table D-1. COBOL Library Subroutines

Name	Description
C$CBID	Bid a COBOL task from COBOL.
C$BSRT	Sort an array on a given character string.
C$CARG	Return USING argument information.
C$CMPR	Compare character strings logically.
C$CVDT	Close all VDTs currently open.
C$EXCP	Turn off function key accessibility.
C$FCFD	
C$BKSP	Backspace I/O on sequential file.
C$DLTE	Delete a file from COBOL.
C$MFAP	Modify file access privilege.
C$MKEY	Modify a KIF alternate key attribute to be nonmodifiable in program declaration.
C$RPRV	Read Previous I/O on KIF.
C$TMPF	Set a temporary file flag; next OPEN ... OUTPUT creates a temporary file.
C$GRPC	
C$GROF	Turn off graphic display option.
C$GRPH	Turn on graphics bit.
C$LOC	Return the address of the data argument.
C$RERR	Return the I/O completion status of the last file.
C$SCI	
C$MAPS	Map and return synonym value.
C$PARM	Get parameter from terminal communications area.
C$SETS	Define or redefine a synonym in the terminal communications area.
C$SCRN	
C$CLOS	Close VDT device and output file.
C$OPEN	Open the VDT device and output file.
C$WRIT	Write the VDT screen contents to the output file or device.
C$SIGN	
C$ADDP	Embed the sign character with last data character.
C$SEPP	Separate embedded data character and sign character into data character and separate trailing sign.
C$SRCH	Perform a binary search on the array for the specified key value.
C$SVC	Issue an SVC to operating system.

All of these subroutines reside in the library .S$SYSLIB.C$SUBS. They must be linked with the user application program using the LIBRARY or SEARCH command. These routines must be included in the task area, and the library .SCI990.S$OBJECT must also be declared. The following shows a typical link control file, which can be used successfully to link any of the subroutines onto a program file:

```
FORMAT IMAGE,REPLACE
LIBR .S$SYSLIB.C$SUBS
LIBR .SCI990.S$OBJECT
PROC RCOBOL
DUMMY
INCL .S$SYSLIB.RCBPRC
TASK CBLTSK
INCL .S$SYSLIB.RCBTSK
INCL .S$SYSLIB.RCBMPD
INCL <COBOL object module>
END
```

NOTE

Refer to the *COBOL Reference Manual* for details on using the CALL statement with the USING option for passing parameters to subroutines. Pathnames specified as parameters in the USING list may be specified as valued data items or nonnumeric literals enclosed in quotes.

The available routines are discussed below. Each discussion addresses the function of the routine, the COBOL calling sequence, each required argument, and where applicable, the error codes returned.

D.2 C$CBID

Function: This bids a COBOL task from COBOL with the message access name set to DUMY. Any file that will be deleted from within that bid task via a call to C$DLTE must not be open at the time C$CBID is called.

Any task bid through C$CBID should not use any SCI interface routines that update the Task Communications Area (TCA). These include S$PTCA, S$SETS (and C$SETS), S$OPEN, S$WRIT, S$WEOL, and S$CLOS. The completion code and message are not available to the terminal associated with the original task.

When a COBOL task bids another COBOL task using C$CBID, a copy of the original task's synonym table is copied for use in the bid task. Any further updates of the synonym table in the original task are not available to the bid task.

Calling Sequence:

CALL "C$CBID" USING ERR ID LUNO FLAGS.

ERR is a two-byte alphanumeric data item for returning an error code. A value of 00 indicates successful completion. Refer to Table D-2 for a complete list of error codes.

ID is an eight-byte alphanumeric data item specifying the installed task name.

LUNO is a two-byte alphanumeric data item specifying the LUNO assigned to the program file containing the task to be bid. The LUNO must be specified as the hexadecimal value of the LUNO; for example, LUNO >40 would be specified as '40'.

FLAGS is a COMP-1 data item containing bid information, as specified by the following values:

Value	Meaning
0	Task is bid without suspending calling task
1	Task is bid; calling task suspended until task ends
2	Task is bid suspended; calling task continues
3	Task is bid suspended; calling task suspended
4	Task is bid controlled; calling task continues
5	Task is bid controlled; calling task suspended
6	Task is bid controlled and suspended; calling task continues
7	Task is bid controlled and suspended; calling task suspended until task ends
8	Task is bid; calling task terminated
9	Same as 8
10	Task is bid suspended; calling task terminated
11	Same as 10
12	Task is bid controlled; calling task terminated
13-15	Task is bid; calling task terminated

Values 8-15 use task chaining which does the following:

- When C$CBID is called in foreground to bid another task, it does so and then terminates itself. SCI does not reactivate until the bid task has terminated. The bid task can perform ACCEPT and DISPLAY commands to and from the terminal. Any synonyms updated by the calling task and the bid task are available to SCI after the bid task terminates.

- When C$CBID is called in batch mode, the batch stream is suspended until the bid task has completed.

When using the task chaining facility, the bidding program must take into account the additional memory requirements for the C$CBID subroutine, the S$BIDT routine, and the TCA record which is initiated by S$BIDT.

A controlled bid, as selected by values 4-7 and 12 for parameter FLAGS in a C$CBID call, corresponds to a bid performed using the SCI primitive .DBID. That is, the task is bid in ASSEMBLY DEBUG mode.

Table D-2. Error Codes Returned for C$SUBS Subroutines

Code	Meaning
00	Successful completion
21	Cannot map synonym
90	Illegal open mode or operation
91	File CFD not found
92	Illegal file type
99	Incorrect argument list
BF	Invalid key number
Others	Operating system errors

D.3 C$BSRT

Function: Sort an array in ascending sequence on a given character-string key, using a bubble-sort technique.

Calling Sequence:

CALL "C$BSRT" USING RECORD-CNT ARRAY-NAME RECORD-LENGTH
KEY-LENGTH KEY-DISPLACEMENT.

RECORD-CNT is a COMP-1 data item specifying the maximum number of entries in the array to be sorted.

ARRAY-NAME is the array containing entries to be sorted. It must either be declared as a one-dimensional array, or made to appear as one by the specification of a RECORD-LENGTH to encompass any subarrays.

RECORD-LENGTH is a COMP-1 data item specifying the size of an individual array element.

KEY-LENGTH is a COMP-1 data item specifying the length of the character string to be used as the sort key.

KEY-DISPLACEMENT is a COMP-1 data item specifying the displacement of the sort key from the beginning of the array element. The first character position in the array element has a displacement of 0.

D.4 C$CARG

Function: This fetches and returns information about an argument in the USING list of a CALL statement. Refer to the section of this manual, "Calling Subroutines", for a description of the information returned.

Calling Sequence:

CALL "C$CARG" USING ERR BUFFER ARG.

ERR is a two-byte alphanumeric data item that gets set by C$CARG to an error code. A value of 00 indicates successful completion. Refer to Table D-2 for a complete list of error codes.

BUFFER is the address of a buffer area to which the argument information is returned. The buffer must be at least 10 bytes long.

ARG is the name of the data item for which argument information is obtained.

D.5 C$CMPR

Function: This compares character strings logically using a specified table. The table defines nonstandard ASCII collating sequences.

Calling Sequence:

CALL "C$CMPR" USING STAT STRNG1 STRNG2 TABLE.

STAT is a two-byte alphanumeric data item for returning a comparison result. Possible completion codes are:

HI if STRNG1 > STRNG2
EQ if STRNG1 = STRNG2
LO if STRNG1 < STRNG2

A value of 99 indicates an incorrect argument list.

STRNG1 is the name of string records that are to be compared.

STRNG2 is the name of string records that are to be compared.

TABLE is the name of a record area that defines nonstandard collating sequences.

If a character is not defined in the table, it is assumed to use its standard ASCII code value.

EXAMPLE:

```
01    ALT-TABLE   PICTURE X(64) VALUE
      "A[BCDEFGHIJKLMNO/PQRSTU]VWXYZ__ 'a{bcdefghi"
      "jklmno|pqrs~tuvwxyz}".
```

D.6 C$CVDT

Function: Close all VDTs currently open for ACCEPT or DISPLAY functions. This allows tasks that are bid by C$CBID to gain access to the function keys.

Calling Sequence:

CALL "C$CVDT".

D.7 C$EXCP

Function: This turns off accessibility to the function keys. C$EXCP should be called prior to the first ACCEPT or DISPLAY command because the first ACCEPT or DISPLAY command to a VDT gains exclusive access to the function keys within a task. C$EXCP causes the ACCEPT and DISPLAY commands to ignore function key requests.

Calling Sequence:

CALL "C$EXCP".

D.8 C$FCFD

The alternate entry points C$BKSP, C$DLTE, C$MFAP, C$MKEY, C$RPRV, and C$TMPF all require COBOL file definitions (CFD) contained in the COBOL module. In all cases the pathname values in the working storage field, or the string used in the select statement, is used as is. No synonym resolution will be performed.

D.8.1 C$BKSP

Function: This performs backspace I/O on sequential file. The calling program must contain the file description, and the file must be open when called.

Calling Sequence:

CALL "C$BKSP" USING ERR PATHNAME CNT.

ERR is a two-byte alphanumeric data item for returning an error code. A value of 00 indicates successful completion. Refer to Table D-2 for a complete list of error codes.

PATHNAME must contain the same pathname, synonym, or data name value as specified in the SELECT...ASSIGN statement. PATHNAME must be declared as a valued data item or a nonnumeric literal. Refer to Figure D-1 for examples.

CNT is a COMP-1 data item specifying the number of records to backspace.

D.8.2 C$DLTE

Function: This deletes a file from COBOL. Any LUNO associated with the file will be released. Files with assigned global LUNOs will not be deleted. The calling program must contain the file description. The file must have been opened and closed within the COBOL program before calling C$DLTE.

Calling Sequence:

CALL "C$DLTE" USING ERR PATHNAME.

ERR is a two-byte alphanumeric data item for returning an error code. A value of 00 indicates successful completion. Refer to Table D-2 for a complete list of error codes.

PATHNAME must contain the same pathname, synonym, or data name value as specified in the SELECT...ASSIGN statement. PATHNAME must be declared as a valued data item or a nonnumeric literal. Refer to Figure D-1 for examples.

D.8.3 C$MFAP

Function: This changes the file access privilege to "exclusive all". The calling program must contain the file description. The file must be opened prior to calling C$MFAP.

Calling Sequence:

CALL "C$MFAP" USING ERR PATHNAME.

ERR is a two-byte alphanumeric data item for returning an error code. A value of 00 indicates successful completion. Refer to Table D-2 for a complete list of error codes.

PATHNAME must contain the same pathname, synonym, or data name value as specified in the SELECT...ASSIGN statement. PATHNAME must be declared as a valued data item or a nonnumeric literal. Refer to Figure D-1 for examples.

D.8.4 C$MKEY

Function: This changes a COBOL program's declaration of an alternate key from modifiable (default) to nonmodifiable. The calling program must contain the file description. C$MKEY must be called prior to opening the file.

The KIF file to be opened must contain the characteristics specified in the SELECT statement and the C$MKEY description.

Calling Sequence:

CALL "C$MKEY" USING ERR PATHNAME KEY.

ERR is a two-byte alphanumeric data item for returning an error code. A value of 00 indicates successful completion. Refer to Table D-2 for a complete list of error codes.

PATHNAME must contain the same pathname, synonym, or data name value as specified in the SELECT ... ASSIGN statement. PATHNAME must be declared as a valued data item or as a nonnumeric literal. Refer to Figure D-1 for examples.

KEY is a COMP-1 data item specifying the ordinal number of the key whose status is to be changed. The primary key is always key "1" regardless of displacement.

D.8.5 C$RPRV

Function: This reads previous I/O on KIF. The calling module must contain the file description. The file must be open and in the sequential access mode. C$RPRV functions like a Read Next command except the previous record is read based on the key of reference.

Calling Sequence:

CALL "C$RPRV" USING ERR PATHNAME BUFFER.

ERR is a two-byte alphanumeric data item for returning an error code. A value of 00 indicates successful completion. Refer to Table D-2 for a complete list of error codes.

PATHNAME must contain the same pathname, synonym, or data name value as specified in the SELECT...ASSIGN statement. PATHNAME must be declared as a valued data item or a nonnumeric literal. Refer to Figure D-1 for examples. If the PATHNAME is a data item, it must be unique to that KIF file. Otherwise, the program may produce indeterminant results.

BUFFER is the name of a record area in which to return the record read.

D.8.6 C$TMPF

Function: This sets a flag that causes the following OPEN... OUTPUT command to create a temporary file. If more than one file is specified in the OPEN statement, only the first will be created as a temporary. The calling program must contain the file description. Refer to Figure D-1 for examples.

Calling Sequence:

CALL "C$TMPF".

D.9 C$GRPC

The alternate entry points, C$GROF and C$GRPH, affect the graphics VDT capabilities.

D.9.1 C$GROF

Function: This turns off the graphics display option.

Calling Sequence:

CALL "C$GROF".

D.9.2 C$GRPH

Function: This turns on the graphics display option.

Calling Sequence:

CALL "C$GRPH".

D.10 C$LOC

Function: This returns the run-time address of a COBOL data item or I/O buffer.

Calling Sequence:

CALL "C$LOC" USING VARIABLE-NAME, VARIABLE-ADDRESS.

VARIABLE-NAME is a group level item or a single data name. It may not be a table item.

VARIABLE-ADDRESS must be aligned on a word boundary as an 01-level PIC 9(4) COMP-4 data item.

D.11 C$RERR

Function: This retrieves the last file I/O completion status.

Calling Sequence:

CALL "C$RERR" USING RET-STATUS.

RET-STATUS is a four-byte alphanumeric data item used for returning the DX10 I/O completion code (bytes 1-2) and the KIF information code (bytes 3-4) where applicable. These codes are in ASCII representation.

D.12 C$SCI

The alternate entry points, C$MAPS, C$PARM, and C$SETS, all use SCI routines. Refer to Figure D-2 for examples.

D.12.1 C$MAPS

Function: This maps and returns a synonym value from the terminal communications area.

Calling Sequence:

CALL "C$MAPS" USING ERR SYNONYM SYN-VALUE.

ERR is a two-byte alphanumeric data item for returning an error code. A value of 00 indicates successful completion. Refer to Table D-2 for a complete list of error codes.

SYNONYM specifies an alphanumeric data item or a nonnumeric literal value which may neither exceed 50 characters nor contain embedded blanks.

SYN-VALUE specifies an alphanumeric data item of sufficient length to contain the mapped value of the synonym. If the value contains embedded blanks, only that part which precedes the first blank will be returned.

D.12.2 C$PARM

Function: This gets the parameter placed in the terminal communications area by the command procedure via the PARMS parameter.

Calling Sequence:

CALL "C$PARM" USING ERR PARM-NO PARM-VALUE.

ERR is a two-byte alphanumeric data item for returning an error code. A value of 00 indicates successful completion. Refer to Table D-2 for a complete list of error codes.

PARM-NO is a COMP-1 data item specifying the parameter in the PARMS list for the desired parameter.

PARM-VALUE is an alphanumeric data item of sufficient length to contain the returned parameter value.

D.12.3 C$SETS

Function: This defines or redefines a synonym in the terminal communications area.

Calling Sequence:

CALL "C$SETS" USING ERR SYNONYM SYNONYM-VALUE.

ERR is a two-byte alphanumeric data item for returning an error code. A value of 00 indicates successful completion. Refer to Table D-2 for a complete list of error codes.

SYNONYM specifies an alphanumeric data item or a nonnumeric literal value which may neither exceed 50 characters nor contain blanks.

SYNONYM-VALUE specifies that an alphanumeric data item contain the value of the synonym. The value may neither exceed 50 characters nor contain embedded blank characters. If blanks are present, only that portion of the value which precedes the first blank will be used, and the remainder will be ignored.

D.13 C$SCRN

The alternate entry points, C$OPEN, C$WRIT, and C$CLOS are used together to open the terminal screen and output file, write the screen contents, and close all files.

D.13.1 C$CLOS

Function: This closes the VDT device and output file to which the VDT screen contents were written. C$CLOS should be called only if C$OPEN was called to open the files.

Calling Sequence:

CALL "C$CLOS" USING ERR.

ERR is a two-byte alphanumeric data item for returning an error code. A value of 00 indicates successful completion. Refer to Table D-2 for a complete list of error codes.

D.13.2 C$OPEN

Function: This opens the VDT device and an output file or device for the writing of screen contents. This must be called before calling C$WRIT or C$CLOS.

Calling Sequence:

CALL "C$OPEN" USING ERR PATHNAME.

ERR is a two-byte alphanumeric data item for returning an error code. A value of 00 indicates successful completion. Refer to Table D-2 for a complete list of error codes.

PATHNAME is the pathname of the output file, or the device name if writing to a device.

D.13.3 C$WRIT

Function: Write the contents of the VDT screen to a file or other device. C$OPEN must be called prior to calling C$WRIT.

Calling Sequence:

CALL "C$WRIT" USING ERR.

ERR is a two-byte alphanumeric data item for returning an error code. A value of 00 indicates successful completion. Refer to Table D-2 for a complete list of error codes.

D.14 C$SIGN

The alternate entry points C$ADDP and C$SEPP respectively generate and remove "overpunch" signs.

D.14.1 C$ADDP

Function: This embeds a separate trailing sign with the last data byte forming an overpunch character or unpacked format. Only a negative sign is processed in forming the overpunch character. The characters], J, K, L, M, N, O, P, Q, and R represent the negative values −0 through −9, respectively.

Calling Sequence:

CALL "C$ADDP" USING IN-DATA OUT-DATA.

IN-DATA is a numeric sign-trailing-separate data item of length n.

OUT-DATA is an unsigned numeric data item of length n − 1.

D.14.2 C$SEPP

Function: This separates the last data byte referenced, which is assumed to have an embedded sign, into a data byte and a separate trailing sign. The characters], J, K, L, M, N, O, P, Q, and R convert to a negative sign; the characters 0 through 9, and A through I, and hexadecimal 7B convert to a positive sign.

Calling Sequence:

CALL "C$SEPP" USING IN-DATA CUT-DATA.

IN-DATA is an unsigned numeric data item of length n.

OUT-DATA is a signed numeric data item of length n + 1.

D.15 C$SRCH

Function: This performs a binary search on an array for a specified key value. The array must be in ascending sort order on the specified key.

Calling Sequence:

CALL "C$SRCH" USING RECORD-CNT ARRAY-NAME RECORD-LENGTH KEY-LENGTH KEY-DISPLACEMENT KEY-VALUE.

RECORD-CNT is a COMP-1 data item specifying the maximum number of entries in the array to be searched.

ARRAY-NAME is the array containing entries to be searched. It must be declared as a one-dimensional array.

RECORD-LENGTH is a COMP-1 data item specifying the size of an individual array element.

KEY-LENGTH is a COMP-1 data item specifying the length of the character string to be used as the search key.

KEY-DISPLACEMENT is a COMP-1 data item specifying the displacement of the search key from the beginning of the array element. The first character position in the array element has a displacement of 0.

KEY-VALUE specifies the value of the key to locate in the array. It must be used in the same way as the key field of the array described by KEY-LENGTH and KEY-DISPLACEMENT.

On return, RECORD-CNT contains the array element occurrence number that matches KEY-VALUE or the value 0 if a match is not found.

D.16 C$SVC

Function: This issues an SVC call block to the operating system as defined in the *DX10 System Programmer's Guide*.

Calling Sequence:

CALL "C$SVC" USING SVC-CALL-BLOCK.

SVC-CALL-BLOCK is a COBOL description of the call block for the particular SVC code to execute. Most of the items in the call block need to be unsigned binary items of differing lengths.

The actual SVC status is returned by the SVC mechanism into the second byte of the SVC call block. The second byte is in the range 0 through 255 (decimal), or 0 through >FF (hexadecimal).

D.17 COBOL PROGRAMS USING SUBROUTINES

Figure D-1 and Figure D-2 show two examples of the use of subroutines.

```
 DXCBL          L.R.V YY.DDD COMPILED:MM/DD/YY HH:MM:SS  OPT=M      PAGE    1

SOURCE ACCESS NAME:    MANUAL.PG.SRC.FIG0D01
OBJECT ACCESS NAME:    DUMY
LISTING ACCESS NAME:   MANUAL.PG.LST.FIG0D01
OPTIONS:               M
PRINT WIDTH:           80
PAGE SIZE:             55
PROGRAM SIZE (LINES):  1000



 DXCBL          L.R.V YY.DDD COMPILED:MM/DD/YY HH:MM:SS  OPT=M      PAGE    2
LINE  DEBUG PG/LN   A...B.................................................
   1                IDENTIFICATION DIVISION.
   2                PROGRAM-ID. CALLER.
   3              *     THIS PROGRAM WAS DEVELOPED AS A FUNCTIONAL
   4              *     DEMONSTRATION TEST TO ILLUSTRATE EXECUTION
   5              *     OF SELECTED MEMBERS OF THE COBOL SUBROUTINE
   6              *     LIBRARY.
   7                ENVIRONMENT DIVISION.
   8                CONFIGURATION SECTION.
   9                SOURCE-COMPUTER. TI-990.
  10                OBJECT-COMPUTER. TI-990.
  11                INPUT-OUTPUT SECTION.
  12                FILE-CONTROL.
  13                    SELECT KIFALTF ASSIGN TO RANDOM, "KIFALT"
  14                      ORGANIZATION INDEXED
  15                      ACCESS SEQUENTIAL
  16                      RECORD KEY KEY-NO1
  17                      ALTERNATE RECORD KEY KEY-NO2
  18                      FILE STATUS IS KIFALT-STATUS.
  19                    SELECT SEQFILE ASSIGN TO RANDOM "SEQF".
  20                    SELECT KIFFILE ASSIGN TO RANDOM "KIFF"
  21                      ORGANIZATION INDEXED
  22                      ACCESS SEQUENTIAL
  23                      RECORD KEY KEY-NO.
  24                    SELECT OUTFILE ASSIGN TO RANDOM, DATA-NAME.
  25                    SELECT DLTEFLE ASSIGN TO RANDOM, DATA-NAME2.
  26
  27                DATA DIVISION.
  28                FILE SECTION.
  29                FD   KIFALTF LABEL RECORDS STANDARD.
  30                01   KIF-ALT-REC.
  31                     02 KEY-NO1  PIC XX.
  32                     02 KEY-NO2  PIC X(4).
  33                     02 FILLER   PIC X(74).
```

Figure D-1. COBOL Subroutine Example 1 (Sheet 1 of 7)

```
 34                FD   SEQFILE LABEL RECORDS STANDARD.
 35                01   SEQ-REC PIC X(80).
 36                FD   KIFFILE LABEL RECORDS STANDARD.
 37                01   KIF-REC.
 38                     02 KEY-NO PIC X(2).
 39                     02 FILLER PIC X(78).
 40                FD   OUTFILE LABEL RECORDS STANDARD.
 41                01   OUT-REC PIC X(80).
 42
 43                FD   DLTEFLE LABEL RECORDS STANDARD.
 44                01   DLT-REC PIC X(80).
 45
 46                WORKING-STORAGE SECTION.
 47                01   T-CODE PIC 9.
 48                01   ACTION PIC X.
 49                01   REC-NO PIC 99.
 50                01   KIFALT-STATUS PIC XX VALUE " ".
 51                01   DATA-NAME PIC X(20) VALUE "TEMPF".
 52                01   DATA-NAME2 PIC X(20) VALUE "DLTEF".
 53                01   PATHNAME  PIC X(20).
 54                01   ERR-FLG         PIC X(4) VALUE " ".
 55                01   KEY-NUMBER      PIC 99 COMP-1.
 56                01   BACKSPACE-COUNT PIC 99 COMP-1.


 DXCBL          L.R.V YY.DDD COMPILED:MM/DD/YY HH:MM:SS  OPT=M      PAGE    3
LINE  DEBUG PG/LN   A...B.....................................................
 57                01   ERR-RETRIEVED   PIC XXXX VALUE " ".


 DXCBL          L.R.V YY.DDD COMPILED:MM/DD/YY HH:MM:SS  OPT=M      PAGE    4
LINE  DEBUG PG/LN   A...B.....................................................
 58               /
 59                PROCEDURE DIVISION.
 60                DECLARATIVES.
 61  >0002         DECL1 SECTION.
 62                      USE AFTER STANDARD ERROR PROCEDURE ON OUTFILE.
 63  >0002         DEC1. GO TO RETRIEVE-ERROR.
 64  >0004         END-DEC1. EXIT.
 65  >0008         DECL2 SECTION.
 66                      USE AFTER STANDARD ERROR PROCEDURE ON KIFFILE.
 67  >0008         DEC2. GO TO RETRIEVE-ERROR.
 68  >000A         END-DEC2. EXIT.
 69                END DECLARATIVES.
 70
 71  >000E         MAIN SECTION.
 72  >000E         BEGIN.
```

Figure D-1. COBOL Subroutine Example 1 (Sheet 2 of 7)

```
 73  >000E              DISPLAY "RETRIEVE ERROR WORD - 1" LINE 1 ERASE.
 74  >0016              DISPLAY "SET TEMP FILE FLAG  - 2".
 75  >001A              DISPLAY "BACKSPACE SEQ FILE  - 3".
 76  >001E              DISPLAY "ALT KIF KEY NON-MOD - 4".
 77  >0022              DISPLAY "READ PREVIOUS KIF   - 5".
 78  >0026              DISPLAY "DELETE FILE         - 6".
 79  >002A              DISPLAY "EXCLUSIVE ACCESS    - 7".
 80  >002E              DISPLAY "END TEST            - 8".
 81  >0032              DISPLAY "ENTER TEST CODE: " LINE 10.
 82  >0038              ACCEPT T-CODE LINE 10 POSITION 20 PROMPT.
 83  >0042              GO TO RETRIEVE ERROR
 84                           SET-TEMP-FLAG
 85                           BACKSPACE-SEQ
 86                           ALT-KEY-NON-MOD
 87                           READ-PREVIOUS
 88                           DELETE-FILE
 89                           EXCLUSIVE-ACCESS
 90                           FINISH-TEST
 91                               DEPENDING ON T-CODE.
 92  >0056              GO TO MAIN.
 93  >0058          END-MAIN. EXIT.


 DXCBL          L.R.V YY.DDD COMPILED:MM/DD/YY HH:MM:SS  OPT=M      PAGE    5
LINE  DEBUG PG/LN    A...B.....................................................
 94                 /
 95  >005C           BACKSPACE-SEQ SECTION.
 96  >005C           BEGIN.
 97  >005C                DISPLAY "C$BKSP - BACKSPACE TEST" LINE 1 ERASE.
 98  >0064                OPEN INPUT SEQFILE.
 99  >006A                MOVE "SEQF" TO PATHNAME.
100  >006E                READ SEQFILE  AT END GO TO END-RD.
101  >0076                READ SEQFILE  NO LOCK AT END GO TO END-RD.
102  >007E                READ SEQFILE  AT END GO TO END-RD.
103  >0086                MOVE 3 TO BACKSPACE-COUNT.
104  >008A                CALL "C$BKSP" USING ERR-FLG "SEQF" BACKSTAGE-COUNT.
105  >008C                DISPLAY "ERROR = " ERR-FLG.
106  >0094                MOVE SPACE TO SEQ-REC.
107  >0098                READ SEQFILE AT END GO TO END-RD.
108  >00A0                DISPLAY SEQ-REC.
109  >00A4                DISPLAY "  TEST1 PASSES IF RECORD 1 DISPLAYED".
110  >00A8                CLOSE SEQFILE.
111  >00AE                OPEN I-O SEQFILE.
112  >00B4                MOVE "SEQF" TO PATHNAME.
113  >00B8                READ SEQFILE  AT END GO TO END-RD.
114  >00C0                READ SEQFILE  NO LOCK AT END GO TO END-RD.
115  >00C8                READ SEQFILE  AT END GO TO END-RD.
116  >00D0                MOVE 5 TO BACKSPACE-COUNT.
117  >00D4                CALL "C$BKSP" USING ERR-FLG "SEQF" BACKSPACE-COUNT.
```

Figure D-1. COBOL Subroutine Example 1 (Sheet 3 of 7)

```
118  >00D6              DISPLAY "ERROR = " ERR-FLG.
119  >00DE              MOVE SPACE TO SEQ-REC.
120  >00E2              READ SEQFILE  AT END GO TO END-RD.
121  >00EA              DISPLAY SEQ-REC.
122  >00EE              DISPLAY "  TEST2 PASSES IF RECORD 1 DISPLAYED".
123  >00F2              DISPLAY "HIT Return TO CONTINUE".
124  >00F6              ACCEPT ACTION PROMPT.
125  >00FE          END-RD. CLOSE SEQFILE.
126  >0106          END-SEQ. GO TO MAIN.
127
128  >0108          RETRIEVE-ERROR SECTION.
129  >0108          BEGIN.
130  >0108              DISPLAY "C$RERR - RETRIEVE ERROR TEST" LINE 1 ERASE.
131  >0110              CALL "C$RERR" USING ERR-RETRIEVED.
132  >0112              DISPLAY "ERROR RETRIEVED = " ERR-RETRIEVED.
133  >011A              DISPLAY "HIT Return TO CONTINUE".
134  >011E              ACCEPT ACTION PROMPT.
135  >0124              GO TO MAIN.
136  >0126          END-RET. EXIT.
137
138  >012A          SET-TEMP-FLAG SECTION.
139  >012A          BEGIN.
140  >012A              DISPLAY "C$TMPF - TEMPORARY FILE TEST" LINE 1 ERASE.
141  >0132              CALL "C$TMPF".
142  >0134              OPEN OUTPUT OUTFILE.
143  >013A              MOVE ALL "*" TO OUT-REC.
144  >013E              WRITE OUT-REC.
145  >014A              CLOSE OUTFILE.
146  >0150              OPEN INPUT OUTFILE.
147  >0156              READ OUTFILE AT END GO TO END-TMP-RD.
148  >015E              DISPLAY OUT-REC.
149  >0162              DISPLAY "  TEST PASSES IF RECORD OF ALL * DISPLAYED".


 DXCBL          L.R.V YY.DDD COMPILED:MM/DD/YY HH:MM:SS  OPT=M      PAGE    6
LINE  DEBUG PG/LN   A...B.....................................................
 150  >0166             DISPLAY "HIT Return TO CONTINUE".
 151  >016A             ACCEPT ACTION PROMPT.
 152  >0172          END-TMP-RD. CLOSE OUTFILE.
 153  >017A          END-TEMP. GO TO MAIN.
```

Figure D-1. COBOL Subroutine Example 1 (Sheet 4 of 7)

```
 DXCBL          L.R.V YY.DDD COMPILED:MM/DD/YY HH:MM:SS  OPT=M      PAGE    7
LINE  DEGUB PG/LN   A...B.....................................................
 154                /
 155  >017C          READ-PREVIOUS SECTION.
 156  >017C          BEGIN.
 157  >017C               DISPLAY "C$RPRV - READ PREVIOUS TEST" LINE 1 ERASE.
 158  >0184               OPEN INPUT KIFFILE.
 159  >018A               READ KIFFILE AT END GO TO END-KIFR.
 160  >0192               READ KIFFILE AT END GO TO END-KIFR.
 161  >019A               MOVE "KIFF" TO PATHNAME.
 162  >019E               CALL "C$RPRV" USING ERR-FLG PATHNAME KIF-REC.
 163  >01A0               DISPLAY "ERROR = " ERR-FLG.
 164  >01A8               DISPLAY KIF-REC.
 165  >01AC               DISPLAY "  TEST 1 PASSES IF RECORD 1 DISPLAYED".
 166  >01B0               MOVE "05" TO KEY-NO.
 167  >01B4               START KIFFILE KEY NOT < KEY-NO
 168                        INVALID KEY DISPLAY "INVALID START"
 169                        GO TO END-KIFR.
 170  >01C4               MOVE "KIFF" TO PATHNAME.
 171  >01C8               CALL "C$RPRV" USING ERR-FLG PATHNAME KIF-REC.
 172  >01CA               DISPLAY "ERROR = " ERR-FLG.
 173  >01D2               DISPLAY KIF-REC.
 174  >01D6               DISPLAY "  TEST 2 PASSES IF RECORD 5 DISPLAYED".
 175  >01DA               DISPLAY "HIT Return TO CONTINUE".
 176  >01DE               ACCEPT ACTION PROMPT.
 177  >01E6          END-KIFR. CLOSE KIFFILE.
 178  >01EE          END-PRV. GO TO MAIN.
 179
 180  >01F0          DELETE-FILE SECTION.
 181  >01F0          BEGIN.
 182  >01F0               DISPLAY "C$DLTE - DELETE TEST" LINE 1 ERASE.
 183  >01F8               MOVE DATA-NAME2 TO PATHNAME.
 184                      OPEN OUTPUT DLTEFLE.
 185  >0202               CLOSE DLTEFLE.
 186  >0208               CALL "C$DLTE" USING ERR-FLG PATHNAME.
 187  >020A               DISPLAY "ERROR = " ERR-FLG.
 188  >0212               DISPLAY "  TEST PASSES IF NO ERROR RETURNED".
 189  >0216               DISPLAY "HIT Return TO CONTINUE".
 190  >021A               ACCEPT ACTION PROMPT.
 191  >0222          END-DEL. GO TO MAIN.
 192
 193  >0224          ALT-KEY-NON-MOD SECTION.
 194  >0224          BEGIN.
 195  >0224               DISPLAY "C$MKEY - SET ALT KIF KEY NON-MOD TEST" LINE 1
 196                      MOVE "KIFALT" TO PATHNAME.
 197  >0230               MOVE 2 TO KEY-NUMBER.
 198  >0234               CALL "C$MKEY" USING ERR-FLG PATHNAME KEY-NUMBER.
 199  >0236               DISPLAY "ERROR = " ERR-FLG.
 200  >023E               OPEN INPUT KIFALTF.
 201  >0244               DISPLAY "FILE STATUS = " KIFALT STATUS.
 202  >024C               DISPLAY "  TEST PASSES IF FILE STATUS = 00".
 203  >0250               IF KIFALT-STATUS = "00" CLOSE KIFALTF.
 204  >025C               DISPLAY "HIT Return TO CONTINUE."
 205  >0260               ACCEPT ACTION PROMPT.
 206  >0268          END-ALT-MOD. GO TO MAIN.
```

Figure D-1. COBOL Subroutine Example 1 (Sheet 5 of 7)

```
 DXCBL          L.R.V YY.DDD COMPILED:MM/DD/YY HH:MM:SS  OPT=M      PAGE    8
LINE  DEBUG PG/LN   A...B.................................................
 207               /
 208  >026A         EXCLUSIVE-ACCESS SECTION.
 209  >026A         BEGIN.
 210  >026A              DISPLAY "C$MFAP - EXCLUSIVE ACCESS TEST" LINE 1 ERASE.
 211  >0272              OPEN I-O KIFFILE.
 212  >0278              MOVE "KIFF" TO PATHNAME.
 213  >027C              CALL "C$MFAP" USING ERR-FLG "KIFF".
 214  >027E              DISPLAY "ERROR = " ERR-FLG.
 215  >0286              DISPLAY "  TEST PASSES IF NO ERROR RETURNED".
 216  >028A              CLOSE KIFFILE.
 217  >0290              DISPLAY "HIT Return TO CONTINUE".
 218  >0294              ACCEPT ACTION PROMPT.
 219  >029C         END-EXCL. GO TO MAIN.
 220
 221  >029E         FINISH-TEST SECTION.
 222  >029E         BEGIN.
 223  >029E              DISPLAY "END OF JOB".
 224  >02A2              STOP RUN.
 225         ZZZZZZ END PROGRAM.                              *** END OF FILE



 DXCBL          L.R.V YY.DDD COMPILED:MM/DD/YY HH:MM:SS  OPT=M      PAGE    9
 ADDRESS  SIZE  DEBUG ORDER TYPE                  NAME

             0              FILE                 KIFALTF
 >0026      80   GRP    0   GROUP                 KIF-ALT-REC
 >0026       2   ANS    0   ALPHANUMERIC            KEY-NO1
 >0028       4   ANS    0   ALPHANUMERIC            KEY-NO2

             0              FILE                 SEQFILE
 >0076      80   ANS    0   ALPHANUMERIC          SEQ-REC

             0              FILE                 KIFFILE
 >00C6      80   GRP    0   GROUP                 KIF-REC
 >00C6       2   ANS    0   ALPHANUMERIC            KEY-NO

             0              FILE                 OUTFILE
 >0116      80   ANS    0   ALPHANUMERIC          OUT-REC

             0              FILE                 DLTEFLE
 >0166      80   ANS    0   ALPHANUMERIC          DLT-REC

 >01BA       1   NSU    0   NUMERIC UNSIGNED      T-CODE

 >01BC       1   ANS    0   ALPHANUMERIC          ACTION

 >01BE       2   NSU    0   NUMERIC UNSIGNED      REC-NO
```

Figure D-1. COBOL Subroutine Example 1 (Sheet 6 of 7)

```
  >01C0      2   ANS    0   ALPHANUMERIC          KIFALT-STATUS

  >01C2     20   ANS    0   ALPHANUMERIC          DATA-NAME

  >01D6     20   ANS    0   ALPHANUMERIC          DATA-NAME2

  >01EA     20   ANS    0   ALPHANUMERIC          PATHNAME

  >01FE      4   ANS    0   ALPHANUMERIC          ERR-FLG

  >0202      2   NBS    0   BINARY SIGNED         KEY-NUMBER

  >0204      2   NBS    0   BINARY SIGNED         BACKSPACE-COUNT

  >0206      4   ANS    0   ALPHANUMERIC          ERR-RETRIEVED

READ ONLY BYTE SIZE =        >0762

READ/WRITE BYTE SIZE =       >043E

OVERLAY SEGMENT BYTE SIZE =  >0000

TOTAL BYTE SIZE =            >0BA0

    0 ERRORS

    0 WARNINGS


 DXCBL          L.R.V YY.DDD COMPILED:MM/DD/YY HH:MM:SS  OPT=M      PAGE   10
 PROGRAM   USING  COUNT

  C$BKSP          3
  C$DLTE          2
  C$MFAP          2
  C$MKEY          3
  C$RERR          1
  C$RPRV          3
  C$TMPF          0
```

Figure D-1. COBOL Subroutine Example 1 (Sheet 7 of 7)

```
 DXCBL          L.R.V YY.DDD COMPILED:MM/DD/YY HH:MM:SS  OPT=M      PAGE    1

SOURCE ACCESS NAME:    MANUAL.PG.SRC.FIG0C02
OBJECT ACCESS NAME:    DUMY
LISTING ACCESS NAME:   MANUAL.PG.LST.FIG0C02
OPTIONS:               M
PRINT WIDTH:           80
PAGE SIZE:             55
PROGRAM SIZE (LINES):  1000



 DXCBL          L.R.V YY.DDD COMPILED:MM/DD/YY HH:MM:SS  OPT=M      PAGE    2
LINE  DEBUG PG/LN   A...B.....................................................
   1                IDENTIFICATION DIVISION.
   2                PROGRAM-ID. CBLMAPS.
   3               * THIS PROGRAM WAS DEVELOPED AS A FUNCTIONAL
   4               * DEMONSTRATION TEST TO VERIFY THE INTERFACE
   5               * OF COBOL TO SCI VIA SUBROUTINES.
   6                ENVIRONMENT DIVISION.
   7                CONFIGURATION SECTION.
   8                SOURCE-COMPUTER. TI-990.
   9                OBJECT-COMPUTER. TI-990.
  10                DATA DIVISION.
  11                FILE SECTION.
  12                WORKING-STORAGE SECTION.
  13                01  ACTION PIC X.
  14                01  ERR-FLG PIC 99 VALUE 0.
  15                01  PATHNAME PIC X(50) VALUE " ".
  16                01  SYNONYM PIC X(20) VALUE " ".
  17                01  BUFFR  PIC X(50) VALUE " ".
  18                01  NUM PIC 99 COMP-1.
  19                PROCEDURE DIVISION.
  20  >0000         MAIN-PROG.
  21  >0000            DISPLAY"C$PARM, C$SETS, C$MAPS - SCI TESTS" LINE 1 ERASE
  22  >0008            DISPLAY "ENTER PARM NO." LINE 3.
  23  >000E            ACCEPT NUM LINE 3 POSITION 20 CONVERT.
  24  >0018            MOVE " " TO BUFFR.
  25  >001C            CALL "C$PARM" USING ERR-FLG NUM BUFFR.
  26  >001E            IF ERR-FLG = 00
  27                      DISPLAY "PARAMETER = " LINE 4
  28                         BUFFR LINE 4 POSITION 20
  29                      ELSE DISPLAY "PARM ERROR" LINE 4
  30                         DISPLAY ERR-FLG LINE 4 POSITION 20.
  31               *
  32  >0042             DISPLAY "ENTER SET SYNONYM" LINE 6.
  33  >0048             ACCEPT SYNONYM LINE 6 POSITION 20 PROMPT.
  34  >0052             DISPLAY "ENTER PATHNAME" LINE 7.
  35  >0058             ACCEPT PATHNAME LINE 7 POSITION 20 PROMPT.
  36  >0062             CALL "C$SETS" USING ERR-FLG SYNONYM PATHNAME.
  37  >0064             IF ERR-FLG = 00
```

Figure D-2. COBOL Subroutine Example 2 (Sheet 1 of 3)

```
 38                        DISPLAY "SYNONYM SET" LINE 8
 39                        ELSE DISPLAY "SET ERROR" LINE 8
 40                           DISPLAY ERR-FLG LINE 8 POSITION 20.
 41                *
 42  >0080                 MOVE " " TO PATHNAME.
 43  >0084                 DISPLAY "ENTER MAP SYNONYM" LINE 10.
 44  >008A                 ACCEPT SYNONYM LINE 10 POSITION 20 PROMPT.
 45  >0094                 CALL "C$MAPS" USING ERR-FLG SYNONYM PATHNAME.
 46  >0096                 IF ERR-FLG = 00
 47                           DISPLAY "SYNONYM VALUE = " LINE 11
 48                              PATHNAME LINE 11 POSITION 20.
 49                           ELSE DISPLAY "MAP ERROR" LINE 11
 50                              DISPLAY ERR-FLG LINE 11 POSITION 20.
 51                *
 52  >00BA                 DISPLAY "HIT Return TO CONTINUE" LINE 22.
 53  >00C0                 ACCEPT ACTION LINE 22 POSITION 30 PROMPT.
 54  >00CA                 STOP RUN.
 55          ZZZZZZ END PROGRAM.                              *** END OF FILE


 DXCBL          L.R.V YY.DDD COMPILED:MM/DD/YY HH:MM:SS  OPT=M      PAGE    3
 ADDRESS  SIZE  DEBUG ORDER TYPE                    NAME

  >002A      1   ANS     0   ALPHANUMERIC          ACTION

  >002C      2   NSU     0   NUMERIC SIGNED        ERR-FLG

  >002E     50   ANS     0   ALPHANUMERIC          PATHNAME

  >0060     20   ANS     0   ALPHANUMERIC          SYNONYM

  >0074     50   ANS     0   ALPHANUMERIC          BUFFR

  >00A6      2   NBS     0   BINARY SIGNED         NUM

READ ONLY BYTE SIZE =        >0284

READ/WRITE BYTE SIZE =       >00B0

OVERLAY SEGMENT BYTE SIZE =  >0000

TOTAL BYTE SIZE =            >0334

    0 ERRORS

    0 WARNINGS
```

Figure D-2. COBOL Subroutine Example 2 (Sheet 2 of 3)

```
DXCBL          L.R.V YY.DDD COMPILED:MM/DD/YY HH:MM:SS  OPT=M      PAGE    4
PROGRAM   USING  COUNT

 C$MAPS          3
 C$PARM          3
 C$SETS          3
```

Figure D-2. COBOL Subroutine Example 2 (Sheet 3 of 3)

Appendix E

COBOL Compiler Listing Format

This appendix shows the output that results from the M, O, and X options on the COBOL compiler. Refer to the *COBOL System Design Document* for details of generated COBOL compiler output.

```
 DXCBL          L.R.V YY.DDD COMPILED:MM/DD/YY HH:MM:SS  OPT=MOX    PAGE    1

SOURCE ACCESS NAME:    MANUAL.PG.SRC.FIGOE01
OBJECT ACCESS NAME:    DUMY
LISTING ACCESS NAME:   MANUAL.PG.LST.FIGOE01
OPTIONS:               MOX
PRINT WIDTH:           80
PAGE SIZE:             55
PROGRAM SIZE (LINES):  1000



 DXCBL          L.R.V YY.DDD COMPILED:MM/DD/YY HH:MM:SS  OPT=MOX    PAGE 2
LINE  DEBUG PG/LN   A...B.....................................................
   1                IDENTIFICATION DIVISION.
   2                PROGRAM-ID. OBJLST.
   3               *     THIS PROGRAM IS USED TO ILLUSTRATE THE FORMAT
   4               *     OF THE COMPILER LISTING WITH M, O, & X OPTIONS.
   5                ENVIRONMENT DIVISION.
   6                CONFIGURATION SECTION.
   7                SOURCE-COMPUTER.  TI-990.
   8                OBJECT-COMPUTER.  TI-990.
   9                INPUT-OUTPUT SECTION.
  10                FILE-CONTROL.
  11                    SELECT OUTFILE ASSIGN TO PRINT "OUTX".
  12                DATA DIVISION.
  13                FILE SECTION.
  14                FD   OUTFILE LABEL RECORDS OMITTED.
  15                01   OUT-REC.
  16                     02 REC1 PIC X(80).
  17                WORKING-STORAGE SECTION.
  18                01   ACTION PIC X.
  19                01   HEADER PIC X(80) VALUE ALL "-".
```

Figure E-1. COBOL Compiler Listing Format (Sheet 1 of 8)

```
E >0084     S >0084     B >2D2D
E >0086     S >0086     B >2D2D
E >0088     S >0088     B >2D2D
E >008A     S >008A     B >2D2D
E >008C     S >008C     B >2D2D
E >008E     S >008E     B >2D2D
E >0090     S >0090     B >2D2D
E >0092     S >0092     B >2D2D
E >0094     S >0094     B >2D2D
E >0096     S >0096     B >2D2D
E >0098     S >0098     B >2D2D
E >009A     S >009A     B >2D2D
E >009C     S >009C     B >2D2D
E >009E     S >009E     B >2D2D
E >00A0     S >00A0     B >2D2D
E >00A2     S >00A2     B >2D2D
E >00A4     S >00A4     B >2D2D
E >00A6     S >00A6     B >2D2D
E >00A8     S >00A8     B >2D2D
E >00AA     S >00AA     B >2D2D
E >00AC     S >00AC     B >2D2D
E >00AE     S >00AE     B >2D2D
E >00B0     S >00B0     B >2D2D
E >00B2     S >00B2     B >2D2D
E >00B4     S >00B4     B >2D2D
E >00B6     S >00B6     B >2D2D
E >00B8     S >00B8     B >2D2D
E >00BA     S >00BA     B >2D2D
E >00BC     S >00BC     B >2D2D
E >00BE     S >00BE     B >2D2D
E >00C0     S >00C0     B >2D2D
E >00C2     S >00C2     B >2D2D
E >00C4     S >00C4     B >2D2D
E >00C6     S >00C6     B >2D2D
E >00C8     S >00C8     B >2D2D
E >00CA     S >00CA     B >2D2D
E >00CC     S >00CC     B >2D2D
```

Figure E-1. COBOL Compiler Listing Format (Sheet 2 of 8)

```
 DXCBL          L.R.V YY.DDD COMPILED:MM/DD/YY HH:MM:SS  OPT=MOX    PAGE    3
LINE  DEBUG PG/LN   A...B.....................................................
    E >00CE    S >00CE   B >2D2D
    E >00D0    S >00D0   B >2D2D
    E >00D2    S >00D2   B >2D2D
    |   |      |   |     |   |
    |   |      |   |     |   |
    |   |      |   |     |   |
    |   |      |   |     |   |
    |   |      |   |     |   ---- DATA VALUES
    |   |      |   |     |
    |   |      |   |     -------- B-TAG:  DENOTES ABSOLUTE VALUE
    |   |      |   |
    |   |      |   |               RELOCATABLE ADDRESS RELATIVE TO START
    |   |      |   --------------- OF SDATA
    |   |      |
    |   |      |                   S-TAG:  DATA (SDATA) RELOCATABLE LOAD
    |   |      ------------------- ADDRESS TAG
    |   |
    |   |                          LOGICAL OFFSET IN ESECT (EXPLICIT DATA
    |   -------------------------- SECTION OF INITIALIZED VALUES ONLY
    |
    ----------------------------- LOGICAL ESECT DESIGNATOR TAG


 20                PROCEDURE DIVISION.
 21  >0000         MAIN-PROG.
       |
       |
       ------------------------- LOGICAL PSECT ALWAYS STARTS AT ZERO

   P >0000   A >0030     B >4707
               |
               |
               --------------- PSECT STARTS AT >30; OBJECT HEADER
                               PRECEDES PSECT.

   P >0002   A >0032     B >4405
   P >0004   A >0034     B >3400
 22  >0000               OPEN OUTPUT OUTFILE WITH NO REWIND.
   P >0006   A >0036     B >5308
   P >0008   A >0038     B >0C09
 23  >0006               MOVE ALL "*" TO OUT-REC.
   P >000A   A >003A     B >470B
   P >000C   A >003C     B >5E0E
   P >000E   A >003E     B >2C0C
   P >0010   A >0040     B >7400
 24  >000A               WRITE OUT-REC.
   P >0012   A >0042     B >530D
   P >0014   A >0044     B >0C09
   P >0016   A >0046     B >460F
```

Figure E-1. COBOL Compiler Listing Format (Sheet 3 of 8)

```
  P >0018   A >0048     B >5E0E
  P >001A   A >004A     B >2C0C
  P >001C   A >004C     B >7400
25  >0012               WRITE OUT-REC FROM HEADER.
  P >001E   A >004E     B >4710
  P >0020   A >0050     B >4401
  P >0022   A >0052     B >0A00
26  >001E               CLOSE OUTFILE WITH NO REWIND.
  P >0024   A >0054     B >0F00
  P >0026   A >0056     B >4C11
  P >0028   A >0058     B >4404
  P >002A   A >005A     B >1613
27  >0024               DISPLAY "ENTER 'C' TO CONTINUE" LINE 1 ERASE.
  P >002C   A >005C     B >0F00
  P >002E   A >005E     B >4408
  P >0030   A >0060     B >0215
28  >002C               ACCEPT ACTION PROMPT.
  P >0032   A >0062     B >5315
  P >0034   A >0064     B >6A17
  P >0036   A >0066     B >1C19
  P >0038   A >0068     B >261A
  |   |     |   |       |   |
  |   |     |   |       |   |
  |   |     |   |       |   |
  |   |     |   |       |   |
  |   |     |   |       |   |    ABSOLUTE INSTRUCTIONS. INSTRUCTION IS
  |   |     |   |       |   |    >25, BUT FIRST INSTRUCTION OF STATEMENT
  |   |     |   |       |   ---- HAS BIT 7 OF FIRST BYTE ON - THUS IT
  |   |     |   |       |        BECOMES >26
  |   |     |   |       |
  |   |     |   |       -------- B-TAG:  DENOTES ABSOLUTE VALUE
  |   |     |   |
  |   |     |   |                PROGRAM (PROG-PSEG) RELOCATABLE
  |   |     |   ---------------- ADDRESS
  |   |     |
  |   |     |                    A-TAG:  DENOTES PROGRAM RELOCATABLE
  |   |     -------------------- ADDRESS
  |   |
  |   |                          LOGICAL PSECT ADDRESS (OBJECT
  |   ------------------------- INSTRUCTIONS)
  |
  ----------------------------- LOGICAL PSECT DESIGNATOR TAG

29  >0032               IF ACTION = "C" GO TO MAIN-PROG.
  P >003A   A >006A     B >5900
30  >003A               STOP RUN.
31        ZZZZZZ END PROGRAM.                          *** END OF FILE
```

Figure E-1. COBOL Compiler Listing Format (Sheet 4 of 8)

```
    L >00D4    A >006C      B >4F55
    L >00D6    A >006E      B >5458
    L >00D8    A >0070      B >3100
    L >00DA    A >0072      B >454E
    L >00DC    A >0074      B >5445
    L >00DE    A >0076      B >5220
    L >00E0    A >0078      B >2743
    L >00E2    A >007A      B >2720
    L >00E4    A >007C      B >544F
    L >00E6    A >007E      B >2043
    L >00E8    A >0080      B >4F4E
```

```
 DXCBL          L.R.V YY.DDD COMPILED:MM/DD/YY HH:MM:SS  OPT=MOX    PAGE    4
LINE  DEBUG PG/LN   A...B.....................................................
    L >00EA    A >0082      B >5449
    L >00EC    A >0084      B >4E55
    L >00EE    A >0086      B >4500
    L >00F0    A >0088      B >4300
    |   |      |   |        |   |
    |   |      |   |        |   |
    |   |      |   |        |   |
    |   |      |   |        |   |
    |   |      |   |        |   ---- ABSOLUTE DATA - LITERALS.
    |   |      |   |        |
    |   |      |   |        -------- B-TAG:  DENOTES ABSOLUTE VALUE
    |   |      |   |
    |   |      |   |                 PROGRAM RELOCATABLE ADDRESS. LOADED
    |   |      |   ----------------- AFTER PSECT.
    |   |      |
    |   |      |                     A-TAG:  DENOTES PSEG OR SPROG LOAD
    |   |      --------------------- ADDRESS
    |   |
    |   ---------------------------- COBOL RELATIVE ADDRESS
    |
    ------------------------------- LSECT (LITERAL SECTION) DESIGNATOR
```

Figure E-1. COBOL Compiler Listing Format (Sheet 5 of 8)

```
F >0000    S >00D4      C >006C
F >0001    S >00D6      B >0004
F >0002    S >00D8      B >0000
F >0003    S >00DA      T >002A
F >0004    S >00DC      B >0056
F >0005    S >00DE      B >0050
F >0006    S >00E0      B >0000
F >0007    S >00E2      B >0000
F >0008    S >00E4      B >0000
F >0009    S >00E6      B >0000
|    |     |    |       |    |
|    |     |    |       |    |
|    |     |    |       |    |
|    |     |    |       |    |
|    |     |    |       |    ---- DATA
|    |     |    |       |
|    |     |    |       |         TAGS:
|    |     |    |       |           C-TAG:  PROGRAM (PSEG) RELOCATABLE DATA
|    |     |    |       |           B-TAG:  ABSOLUTE DATA
|    |     |    |       --------    T-TAG:  DATA (DSEG-SDATA) RELOCATABLE DATA
|    |     |    |
|    |     |    ---------------- RELOCATABLE ADDRESS IN SDATA
|    |     |
|    |     --------------------- S-TAG:  DENOTES SDATA RELOCATABLE
|    |
|    -------------------------- RELATIVE ADDRESS WITHIN FSECT
|
------------------------------ FSECT DESIGNATOR TAG
```

```
F >000A    S >00E8      B >0000
F >000B    S >00EA      B >0000
F >000C    S >00EC      B >0300
F >000D    S >00EE      B >0100
F >000E    S >00F0      B >00FC
F >000F    S >00F2      B >0000
F >0010    S >00F4      B >0000
F >0011    S >00F6      B >0000
F >0012    S >00F8      B >0000
F >0013    S >00FA      B >0000

D >0000    A >008A      B >2C24
D >0001    A >008C      B >2E01
D >0002    A >008E      B >0000
D >0003    A >0090      B >0000
D >0004    A >0092      B >0000
D >0005    A >0094      B >0000
D >0006    A >0096      T >0120
D >0007    A >0098      B >0006
D >0008    A >009A      B >102A
```

Figure E-1. COBOL Compiler Listing Format (Sheet 6 of 8)

```
    D >0009    A >009C      B >6050
    D >000A    A >009E      B >002A
    D >000B    A >00A0      B >0012
    D >000C    A >00A2      B >0050
    D >000D    A >00A4      B >5050
    D >000E    A >00A6      B >0084
    D >000F    A >00A8      B >001E
    D >0010    A >00AA      B >0024
    D >0011    A >00AC      B >8001
    D >0012    A >00AE      B >00D8
    D >0013    A >00B0      B >5015
    D >0014    A >00B2      B >00DA
    D >0015    A >00B4      B >5001
    D >0016    A >00B6      B >0082
    D >0017    A >00B8      B >5001
    D >0018    A >00BA      B >00F0
    D >0019    A >00BC      B >003A
    D >001A    A >00BE      B >0000
    |   |      |   |        |   |
    |   |      |   |        |   |
    |   |      |   |        |   |
    |   |      |   |        |   |
    |   |      |   |        |   ---- ABSOLUTE DATA
    |   |      |   |        |
    |   |      |   |        -------- B-TAG:  ABSOLUTE DATA DESIGNATOR
    |   |      |   |
    |   |      |   |                 RELOCATABLE ADDRESS WITHIN PSEG OR
    |   |      |   ----------------$PROG
    |   |      |
    |   |      |                     A-TAG:  PROGRAM RELOCATABLE PSEG
    |   |      --------------------- DESIGNATOR
    |   |
    |   -------------------------- DSECT RELATIVE ADDRESS
    |
    ------------------------------ DSECT DESIGNATOR TAG
```

```
 DXCBL          L.R.V YY.DDD COMPILED:MM/DD/YY HH:MM:SS  OPT=MOX    PAGE    5
LINE  DEBUG  PG/LN   A...B.................................................

    H >0000    A >0000      T >0000
    H >0000    A >0002      C >0004
    H >0000    A >0004      B >05CE
    H >0000    A >0006      B >069E
    P >0000    A >0008      C >0030
    T >0000    A >000A      T >00D4
    F >0000    A >000C      T >00D4
    D >0000    A >000E      C >008A
    C >0000    A >0010      C >00C0
    E >0000    A >0012      T >0000
    U >0000    A >0014      B >0000
    L >0000    A >0016      C >006C
```

Figure E-1. COBOL Compiler Listing Format (Sheet 7 of 8)

```
    I >0000    A >0018      B >0000
    H >0000    A >001A      B >4F42
    H >0000    A >001C      B >4A4C
    H >0000    A >001E      B >5354
    P >0000    A >0020      B >003C
    T >0000    A >0022      B >0000
    F >0000    A >0024      B >0028
    D >0000    A >0026      B >0036
    X >0000    A >0028      B >0000
    E >0000    A >002A      B >00D4
    U >0000    A >002C      B >0000
    L >0000    A >002E      B >001E


    H >0000    A >0000      B >0000
    H >0000    A >0002      B >0008



 DXCBL          L.R.V YY.DDD COMPILED:MM/DD/YY HH:MM:SS  OPT=MOX     PAGE     6
 ADDRESS  SIZE  DEBUG ORDER TYPE                    NAME

            0              FILE                    OUTFILE
  >002A    80    GRP    0   GROUP                    OUT-REC
  >002A    80    ANS    0   ALPHANUMERIC               REC1

  >0082     1    ANS    0   ALPHANUMERIC            ACTION

  >0084    80    ANS    0   ALPHANUMERIC            HEADER


READ ONLY BYTE SIZE =        >00C0

READ/WRITE BYTE SIZE =       >0122

OVERLAY SEGMENT BYTE SIZE =  >0000

TOTAL BYTE SIZE =            >01E2

    0 ERRORS

    0 WARNINGS



 DXCBL          L.R.V YY.DDD COMPILED:MM/DD/YY HH:MM:SS  OPT=MOX     PAGE     7
CROSS REFERENCE                  /DECL/ *DEST*
ACTION                           /0018/ *0028*  0029
HEADER                           /0019/  0025
MAIN-PROG                        /0021/  0029
OUTFILE                          /0011/ /0014/  0022   0026
OUT-REC                          /0015/ *0023* *0024* *0025
REC1                             /0016/
```

Figure E-1. COBOL Compiler Listing Format (Sheet 8 of 8)

Index

This index lists key topics of this manual and specifies where each topic appears, as follows:

- Sections — Section references appear as *Section n*, where *n* represents the section number.
- Appendixes — Appendix references appear as *Appendix Y*, where *Y* represents the appendix letter.
- Paragraphs — Paragraph references appear as alphanumeric characters separated by decimal points. The first character refers to the section or appendix containing the paragraph, and any other numbers indicate the sequence of the paragraph within the section or appendix. For example:

  — *3.5.2* refers to Section 3, paragraph 5.2.

  — *A.2* refers to Appendix A, paragraph 2.

- Figures — Figure references appear as *Fn-x* or *FY-x*, where *n* represents the section and *Y* represents the appendix containing the figure; *x* represents the number of the figure within the section or appendix. For example:

  — *F2-7* refers to the seventh figure in Section 2.

  — *FG-1* refers to the first figure in Appendix G.

- Tables — Table references appear as *Tn-x* or *TY-x*, where *n* represents the section and *Y* represents the appendix containing the table; *x* represents the number of the table within the section or appendix. For example:

  — *T3-10* refers to the tenth table in Section 3.

  — *TB-4* refers to the fourth table in Appendix B.

- See and See also references — *See* and *See also* direct you to other entries in the index. For example:

  Logical Unit Number . . . . . *See* LUNO

  Device . . . . . *See also* individual device names or numbers

Page numbers that correspond to these index references appear in the Table of Contents.

A Debug Command 7.2.2.1
ACCEPT:
  Statement 11.1
    Use F11-1
Access Name 2.5
Activation, Debugger 7.2.1
Address Stop 7.2
Adjustment, Blank 2.7.1.1
ADU 2.7
AGL SCI Command 6.4.1
AL SCI Command 6.4.1
Alias 5.7
Allocatable Disk Unit (ADU) 2.7
ALLOCATE:
  Linking Using F5-13
  SCI Command 5.6.5
ALN SCI Command 6.4.1
Alternate Directory Structure 3.3
Argument 8.3
  List 8.3
Arithmetic Operations 13.3
AS SCI Command 1.2, 2.6
Assembler Subroutine, COBOL
  Routine Calling F8-5
Assembly Language:
  Module F7-3, F7-4
    COBOL Program Calling F7-2
  Subroutine 8.3
    Debugging 7.3
Assign Address Stop (A) Debug
  Command 7.2.2.1
Assign Global LUNO (AGL) SCI
  Command 6.4.1
Assign Logical Name (ALN) SCI
  Command 6.4.1
Assign LUNO (AL) SCI Command 6.4.1
Assign Synonym (AS) SCI
  Command 1.2, 2.6
AT END Phrase 12.4
Attributes:
  Relative Record 2.7.2.1
  Sequential File 2.7.1.1

Background Task:
  Synonym Availability With a 2.6
BATCH SCI Command 2.3.4.1
Batch Job 2.2.2
  Stream Format 2.3.4.1
  Use 2.3.4
Blank:
  Adjustment 2.7.1.1
  Suppression 2.7.1.1
Blocking Factor 2.7
Building, Program 3.5
By File Type, Organization 3.3.2
By Programs, Organization 3.3.1

CALL Statement 8.3
Calling:
  Assembly Language Module,
    COBOL Program F7-2
  Subroutine 8.1
Capabilities, System 2.1
CC SCI Command 2.3.4.4
CF SCI Command 2.4.3
CFDIR SCI Command 2.4.2, 3.4
CFIMG SCI Command 2.4.3
CFKEY SCI Command 2.4.3
CFPRO SCI Command 2.4.3
CFREL SCI Command 2.4.3
CFSEQ SCI Command 2.4.3
Change Program Location (L)
  Debug Command 7.2.2.4
Characters, Graphic F11-4
CL SCI Command 3.5
CLOSE Statement 3.5, 12.2
COBOL:
  Compilation 4.1
  Compiler:
    Informative Messages TB-3
    Listing With Error Messages FB-1
    Options 4.2.1
    System Error Messages TB-2
    User Error Messages TB-1
  Debug:
    Commands 7.2.2
    Mode 1.2
  Execution 6.1
  Interfacing With DBMS-990 .....F9-3
  I/O Operation Validity Table T12-3
  Library Subroutines TD-1
  Module:
    Debugging 7.2
    Linking DBMS-990 .....9.4.3
    Linking Query-990 and F9-5
    Segmentation 5.4.3
  Overview 1.1
  Procedure F10-3
  Program:
    Calling Assembly Language
      Module F7-2
    Development Overview 1.2
    Module Retrieving Additional
      SCI Parameters F10-4
    Source Module F3-1
    Using Subroutines D.17
  Routine Calling:
    Assembler Subroutine F8-5
    Sort/Merge F9-2
  Run Time 5.6.1
    Error Messages TC-1
    Interpreter 5.6.1
  Segmentation Within Overlay
    Phase Modules F5-8
  Source Module 3.1
  Subroutine FD-1, FD-2

Library Package 8.2, D.1
Codes, C$SUBS Subroutine Error TD-2
Command:
Name 1.3.1
Procedure 10.2
Example 10.3
Procedures, SCI 10.1
Processor 10.2
Prompt 1.3.2
Format, SCI 1.3
Notation T1-1
SCI 1.3
Commands:
COBOL Debug 7.2.2
Debug T7-1
Link Editor T5-1
Communication 9.6
Comparison of Memory Requirements .. F5-7
Compilation, COBOL 4.1
Compiler:
Completion Codes 4.4
Error Messages 4.5
Execution 4.2
Informative Messages, COBOL TB-3
Limitations 4.6
Listing F1-1
With Error Messages, COBOL FB-1
Options 4.2.1
COBOL 4.2.1
Output 4.3
Listing F7-1
System Error Messages, COBOL TB-2
User Error Messages, COBOL TB-1
Completion Codes 6.3, 6.5
Compiler 4.4
Compressed Format 5.5
Contents, Program File F5-3
Control:
File, Link 1.2, 5.5
Operations 13.4
Copy/Concatenate (CC)
SCI Command 2.3.4.4
Copy Lines (CL) SCI Command 3.5
Correspondence Table, Device T12-4
Create Directory File (CFDIR)
SCI Command 2.4.2, 3.4
Create File (CF) SCI Command 2.4.3
Create Image File (CFIMG)
SCI Command 2.4.3
Create Key Indexed File (CFKEY)
SCI Command 2.4.3
Create Program File (CFPRO)
SCI Command 2.4.3
Create Relative Record File (CFREL)
SCI Command 2.4.3
Create Sequential File (CFSEQ)
SCI Command 2.4.3
Creating:
Directory 2.4.2, 3.4
File 2.4.3, 3.4
Linked Object Modules 5.5
Program Images 5.6
Relative Record 2.7.2.2
Relative Record File F2-8
Sequential File 2.7.1.2, F2-2
C$ADDP D.14.1
C$BKSP D.8.1
C$BSRT D.3
C$CARG D.4
C$CBID D.2
C$CLOS D.13.1
C$CMPR D.5
C$CVDT D.6
C$DLTE D.8.2
C$EXCP D.7
C$FCFD D.8
C$GROF D.9.1
C$GRPC D.9
C$GRPH D.9.2
C$LOC D.10
C$MAPS D.12.1
C$MFAP D.8.3
C$MKEY D.8.4
C$OPEN D.13.2
C$PARM D.12.2
C$RERR D.11
C$RPRV D.8.5
C$SCI D.12
C$SCRN D.13
C$SEPP D.14.2
C$SETS D.12.3
C$SIGN D.14
C$SRCH D.15
C$SUBS Subroutine Error Codes TD-2
C$SVC D.16
C$TMPF D.8.6
C$WRIT D.13.3

D Debug Command 7.2.2.2
Data Base Management System
(DBMS-990) 9.4
Data Definition Language (DDL) File F9-4
DBMS-990 ..... F9.4
COBOL:
Interfacing With F9-3
Module, Linking 9.4.3
Features 9.4.1
User Interface 9.4.2
Debug:
Commands T7-1
COBOL 7.2.2
Mode 7.1
COBOL 1.2
Debug Command:
A 7.2.2.1
Assign Address Stop (A) 7.2.2.1
Change Program Location (L) 7.2.2.4
D 7.2.2.2
Dump Data Item (D) 7.2.2.2
E 7.2.2.3
Execute Next Single Statement (S) .. 7.2.2.8
Exit Debug Mode (E) 7.2.2.3

L 7.2.2.4
M 7.2.2.5
Modify Data Item (M) 7.2.2.5
Q 7.2.2.6
Quit Execution (Q) 7.2.2.6
R 7.2.2.7
Resume Program Execution (R) 7.2.2.7
S 7.2.2.8
U 7.2.2.9
Undo Address Stop (U) 7.2.2.9
W 7.2.2.10
Write Screen to Message File (W) . . 7.2.2.10
Debugger Activation 7.2.1
Debugging:
Assembly Language Subroutine 7.3
COBOL Module 7.2
Functions 7.1
Interactive F7-1
Declarative Use 12.4
Default Value 1.3.3.2
DELETE Statement 12.2
Delete Lines (DL) SCI Command 3.5
Description, Relative Record File F2-8
Description:
SCI 2.3.1
Sequential File F2-2
Development Overview, COBOL
Program 1.2
Device Correspondence Table T12-4
Directory 2.4
Creating 2.4.2, 3.4
File 2.4.2, 2.7.2.3
Preparation 3.2
Structure 2.4, F2-1
Alternate 3.3
DISPLAY Statement 1.1
Use F11-1
DL SCI Command 3.5
DSEG 5.6.4, 5.6.5, 5.6.6
Dump Data Item (D) Debug
Command 7.2.2.2

E Debug Command 7.2.2.3
EBATCH SCI Command 2.3.4.1
Editor, Text 3.5
Error:
Codes, C$SUBS Subroutine TD-2
Processing 12.1
Error Messages:
COBOL:
Compiler Listing With FB-1
Compiler System TB-2
Compiler User TB-1
Run-Time TC-1
Compiler 4.5
Run-Time 6.3, 6.5
System B.2
User B.1
Errors, Intercepting I/O F12-1
Example:
Command Procedure 10.3
SCI Command 2.3.3
Execute Batch (XB) SCI Command 2.3.4
Execute COBOL Compiler in Background
(XCC) SCI Command 4.2.2
Execute COBOL Compiler in Foreground
(XCCF) SCI Command . . . . . 1.2, 4.2.
Execute COBOL Program in Background
(XCP) SCI Command 6.2.2
Execute COBOL Program in Foreground
(XCPF) SCI Command 1.2, 6.2.1, 7.2.1
Execute COBOL Task in Background
(XCT) SCI Command 6.4.2
Execute COBOL Task in Foreground
(XCTF) SCI Command 1.2, 6.4.1, 7.2.1
Execute Link Editor
(XLE) SCI Command 1.2
Execute Next Single Statement (S)
Debug Command 7.2.2.8
Execution:
COBOL 6.1
Compiler 4.2
Object Module 6.2
Program Image 6.4
Exit Debug Mode (E) Debug
Command 7.2.2.3
Expected Response 1.3.3

Features, DBMS-990 . . . . . 9.4.1
File 2.4, 2.7
Attributes, Sequential 2.7.1.1
Contents, Program F5-3
Creating 2.4.3, 3.4
Relative Record F2-8
Sequential 2.7.1.2, F2-2
Description, Relative Record F2-8
Description, Sequential F2-2
Directory 2.4.2, 2.7.2.3
Image 2.7.2.3
I/O:
Status 12.2
Status Value 12.3
Key Indexed 2.7.3
Key Indexed (KIF) F2-14
Link Control 1.2, 5.5
Linked Object 1.2
Listing 4.3
Name, Synonym as a 2.6
Object 1.2, 4.3
Preparation 3.2
Program 1.2, 2.7.2.3, 5.4
Relative Record 2.7.2, F2-9, F2-10,
F2-11, F2-12, F2-13
Sequential 2.7.1, F2-3, F2-4,
F2-5, F2-6, F2-7
Source 1.2, F1-1
Special Types Relative Record 2.7.2.3
Status Table T12-1

Structure 2.4, F2-1
Type 2.7
.S$SHARED Program 5.4.1.2
Find String (FS) SCI Command 3.5
Foreground Task 2.2
Synonym Availability With a 2.6
Format:
Batch Stream 2.3.4.1
Compressed 5.5
SCI Command Prompt 1.3
Tagged 5.5
FS SCI Command 3.5
Function Key 11.1
Mapping T11-1
Functions, Debugging 7.1

Generic Key Name 2.3.2, 2.3.3.1, 2.3.3.2
Graphic:
Characters F11-4
Input/Output 11.3
Graphics F11-3

IF SCI Command 3.5
Image:
Execution, Program 6.4
File 2.7.2.3
Images, Creating Program 5.6
Images From Relative File,
Installing Program 5.6.10
Informative Messages B.3
COBOL Compiler TB-3
Initial Value 1.3.3.1
Initialize New Volume
(INV) SCI Command 2.4.1
Initiate Text Editor (XE) SCI Command ... 3.5
Input/Output:
Graphic 11.3
Low Volume 11.2
Insert File (IF) SCI Command 3.5
Installing Program Images From
Relative File 5.6.10
Interactive:
Debugging F7-1
Job 2.2.1
Intercepting I/O Errors F12-1
Interface, DBMS-990 User 9.4.2
Interfacing With DBMS-990, COBOL F9-3
Interpreter, COBOL Run-Time 5.6.1
INV SCI Command 2.4.1
INVALID KEY Phrase 12.4
I/O:
Errors, Intercepting F12-1
Operation Validity Table, COBOL T12-3
Operations 13.6
Status:
File 12.2
Value, File 12.3

Job:
Batch 2.2.2
Interactive 2.2.1
Structure 2.2

Key Indexed File 2.7.3
Key Indexed (KIF) File F2-14
Key Name, Generic 2.3.2, 2.3.3.1, 2.3.3.2
Keyboard Functions 3.5
KIF 2.7.3, F2-14

L Debug Command 7.2.2.4
Language, SCI 10.2
LD SCI Command 2.3.3.2
Library:
Linking 5.7
Package, COBOL Subroutine 8.2, D.1
Random 5.7, F5-18
Sequential 5.7
Subroutines, COBOL TD-1
LIBRARY SCI Command 8.2
Limitations, Compiler 4.6
Link Control File 1.2, 5.5
Link Editor 5.1
Commands T5-1
Requirements F5-1
Linked:
Object File 1.2
Object Modules, Creating 5.5
Linking:
DBMS-990 COBOL Module 9.4.3
Library 5.7
Overlay 5.6.6
P1 With Different P2 ..... 5.6.8, F5-16
Query-990 and COBOL Module F9-5
Single Procedure With:
Multiple Tasks 5.6.3, F5-10
Single Task 5.6.2, F5-9
Single Task User Program
File 5.6.9, F5-17
Two Procedures With:
Multiple Tasks 5.6.5, F5-12
Single Task 5.6.4, F5-11
Using ALLOCATE F5-13
List, Argument 8.3
List Directory (LD) SCI Command 2.3.3.2
Listing Compiler F1-1
File 4.3
Output F7-1
With Error Messages, COBOL
Compiler FB-1
LOAD SCI Command 5.6.6
Logical Record 2.7
Logical Record Length (LRECL) 2.7
Logical Unit Number (LUNO) 1.2
Low Volume Input/Output 11.2
LRECL 2.7
LUNO 1.2

M Debug Command 7.2.2.5
Main Program Module:
Sharing 5.6.7

With P2, Sharing F5-15
Mapping:
Function Key T11-1
Program 5.3
Memory Requirements,
Comparison of F5-7
ML SCI Command 3.5
Mode:
COBOL Debug 1.2
Debug 7.1
VDT 2.3.2
Modify Data Item (M) Debug
Command 7.2.2.5
Modify Roll (MR) SCI Command 3.5
Modify Synonym (MS) SCI Command 2.6
Module 1.2
Assembly Language F7-3, F7-4
COBOL:
Program Calling Assembly
Language F7-2
Program Source F3-1
Source 3.1
Debugging COBOL 7.2
Linking:
DBMS-990 COBOL 9.4.3
Query-990 and COBOL F9-5
Retrieving Additional SCI Parameters,
COBOL Program F10-4
Segmentation, COBOL 5.4.3
Move Operations 13.5
Move Lines (ML) SCI Command 3.5
MR SCI Command 3.5
MS SCI Command 2.6
Multiple Tasks:
Linking:
Single Procedure With 5.6.3, F5-10
Two Procedures With 5.6.5, F5-12
Separate Program Files F5-6
Sharing:
Same P1 and P2 . . . . . F5-4
Same P1 but Different P2s F5-5

Name:
Access 2.5
Command 1.3.1
Synonym as a File 2.6
Volume 2.4.1
Notation:
Command Prompt T1-1
SCI Command Prompt 1.3

Object File 1.2, 4.3
Linked 1.2
Object Module Execution 6.2
Object Modules, Creating Linked 5.5
OPEN Statement 12.2
Operating System 2.1
Operation Validity Table, COBOL I/O . . . T12-3
Operations:
Arithmetic 13.3
Control 13.4
I/O 13.6
Move 13.5
Optimizing Run-Time Performance 13.1
Options:
COBOL Compiler 4.2.1
Compiler 4.2.1
Organization:
By File Type 3.3.2
By Programs 3.3.1
Output:
Compiler 4.3
Listing, Compiler F7-1
Overlay 5.4.2
Linking 5.6.6
Phase 5.4.2
Structure 5.6.6, F5-14
Overview:
COBOL 1.1
Program Development 1.2
System 1.1

Package, COBOL Subroutine
Library 8.2, D.1
Paragraph, SPECIAL-NAMES 6.2.1, F6-18
Parameters, COBOL Program Module
Retrieving Additional SCI F10-4
PARMS 6.6
Pathname 2.5
Performance, Optimizing Run-Time 13.1
Phase, Overlay 5.4.2
PHASE SCI Command 5.6.5, 5.6.6
Phrase:
AT END 12.4
INVALID KEY 12.4
Physical Record 2.7
Physical Record Length (PRECL) 2.7.2.2
PRECL 2.7.2.2
Preparation:
Directory 3.2
File 3.2
Procedure:
COBOL F10-3
SCI F10-1
Segment 5.4.1.2
Reentrant 5.4.1.2
Tailored SCI F10-2
Processing, Error 12.1
Productivity Tools 9.1
Program 1.2
Building 3.5
Calling Assembly Language
Module, COBOL F7-2
Development Overview, COBOL 1.2
File 1.2, 2.7.2.3, 5.4
Contents F5-3
.S$SHARED 5.4.1.2
Image Execution 6.4
Images, Creating 5.6
From Relative File, Installing 5.6.10
Mapping 5.3

Module Retrieving Additional SCI Parameters, COBOL F10-4
Source Module, COBOL F3-1
Using Subroutines, COBOL D.17
Program File, Linking Single Procedure With Single Task User 5.6.9, F5-17
PSEG 5.6.4, 5.6.5, 5.6.6
P1 with Different P2, Linking 5.6.8, F5-16

Q Debug Command 7.2.2.6
QE SCI Command 3.5
QUERY-990 . . . . . 9.5
Query-990 and COBOL Module, Linking F9-5
Quit Edit (QE) SCI Command 3.5
Quit Execution (Q) Debug Command . . 7.2.2.6

R Debug Command 7.2.2.7
Random Library 5.7, F5-18
READ Statement 12.2
Reentrant Procedure Segment 5.4.1.2
Relative File, Installing Program Images from 5.6.10
Relative Record:
Attributes 2.7.2.1
Creating 2.7.2.2
File . . 2.7.2, F2-9, F2-10, F2-11, F2-12, F2-13
Creating F2-8
Description F2-8
Special Types 2.7.2.3
Release Level, Software 4.1
Requirements, Link Editor F5-1
Response, Expected 1.3.3
Resume Program Execution (R) Debug Command 7.2.2.7
Retrieving Additional SCI Parameters, COBOL Program Module F10-4
REWRITE Statement 12.2
Routine Calling:
Assembler Subroutine, COBOL F8-5
Sort/Merge, COBOL F9-2
Run Time, COBOL 5.6.1
Run-Time:
Error Messages 6.3, 6.5
COBOL TC-1
Interpreter, COBOL 5.6.1
Performance, Optimizing 13.1

S Debug Command 7.2.2.8
Same P1 and P2, Multiple Tasks Sharing F5-4
Same P1 but Different P2s, Multiple Tasks Sharing F5-5
SBS SCI Command 2.3.3.1
SCI 1.1, 2.2.1, 2.3, 2.3.3
Command Procedures 10.1
Command Prompt:
Format 1.3
Notation 1.3
Description 2.3.1
Language 10.2
Parameters, COBOL Program Module Retrieving Additional F10-4
Procedure F10-1
Tailored F10-2
SCI Command:
AGL 6.4.1
AL 6.4.1
ALLOCATE 5.6.5
ALN 6.4.1
AS 1.2, 2.6
Assign Global LUNO (AGL) 6.4.1
Assign Logical Name (ALN) 6.4.1
Assign LUNO (AL) 6.4.1
Assign Synonym (AS) 1.2, 2.6
BATCH 2.3.4.1
CC 2.3.4.4
CF 2.4.3
CFDIR 2.4.2, 3.4
CFIMG 2.4.3
CFKEY 2.4.3
CFPRO 2.4.3
CFREL 2.4.3
CFSEQ 2.4.3
CL 3.5
Copy/Concatenate (CC) 2.3.4.4
Copy Lines (CL) 3.5
Create Directory File (CFDIR) 2.4.2, 3.4
Create File (CF) 2.4.3
Create Image File (CFIMG) 2.4.3
Create Key Indexed File (CFKEY) 2.4.3
Create Program File (CFPRO) 2.4.3
Create Relative Record File (CFREL) . . 2.4.3
Create Sequential File (CFSEQ) 2.4.3
Delete Lines (DL) 3.5
DL 3.5
EBATCH 2.3.4.1
Example 2.3.3
Execute Batch (XB) 2.3.4
Execute COBOL Compiler in Background (XCC) 4.2.2
Execute COBOL Compiler in Foreground (XCCF) 1.2, 4.2.1
Execute COBOL Program in Background (XCP) 6.2.2
Execute COBOL Program in Foreground (XCPF) 1.2, 6.2.1, 7.2.1
Execute COBOL Task in Background (XCT) 6.4.2
Execute COBOL Task in Foreground (XCTF) 1.2, 6.4.1, 7.2.1
Execute Link Editor (XLE) 1.2
Find String (FS) 3.5
FS 3.5
IF 3.5
Initialize New Volume (INV) 2.4.1
Initiate Text Editor (XE) 3.5
Insert File (IF) 3.5
INV 2.4.1
LD 2.3.3.2
LIBRARY 8.2

List Directory (LD) 2.3.3.2
LOAD 5.6.6
ML 3.5
Modify Roll (MR) 3.5
Modify Synonym (MS) 2.6
Move Lines (ML) 3.5
MR 3.5
MS 2.6
PHASE 5.6.5, 5.6.6
QE 3.5
Quit Edit (QE) 3.5
SBS 2.3.3.1
SEARCH 8.2
Show Background Status (SBS) 2.3.3.1
Show Line (SL) 3.5
SL 3.5
TASK 5.6.5
XB 2.3.4
XCC 4.2.2
XCCF 1.2, 4.2.1
XCP 6.2.2
XCPF 1.2, 6.2.1, 7.2.1
XCT 6.4.2
XCTF 1.2, 6.4.1, 7.2.1
XE 3.5
XLE 1.2, 2.3.4.1
Screen Description, TIFORM VDT F9-1
SEARCH SCI Command 8.2
Segment:
Procedure 5.4.1.2
Reentrant Procedure 5.4.1.2
Task 5.4.1.1
Segmentation:
COBOL Module 5.4.3
Within Overlay Phase Modules, COBOL F5-8
Separate Program Files, Multiple Tasks F5-6
Sequential:
File 2.7.1, F2-3, F2-4, F2-5, F2-6, F2-7
Attributes 2.7.1.1
Creating 2.7.1.2, F2-2
Description F2-2
Library 5.7
Sharing:
Main Program Module 5.6.7
With P2 . . . . . F5-15
Same P1 and P2, Multiple Tasks F5-4
Same P1 but Different P2s, Multiple Tasks F5-5
Show Background Status (SBS) SCI Command 2.3.3.1
Show Line (SL) SCI Command 3.5
Single Procedure With:
Multiple Tasks, Linking 5.6.3, F5-10
Single Task:
Linking 5.6.2, F5-9
User Program File, Linking . . 5.6.9, F5-17
Single Task:
Linking:
Single Procedure With 5.6.2, F5-9
Two Procedures With 5.6.4, F5-11
User Program File, Linking Single Procedure With 5.6.9, F5-17
SL SCI Command 3.5
Software Release Level 4.1
Sort/Merge 9.3
COBOL Routine Calling F9-2
Source:
File 1.2, F1-1
Module:
COBOL 3.1
COBOL Program F3-1
Special Types Relative Record File . . . 2.7.2.3
SPECIAL-NAMES Paragraph 6.2.1, F6-18
START Statement 12.2
Statement:
ACCEPT 1.1
CALL 8.3
CLOSE 3.5, 12.2
DELETE 12.2
DISPLAY 1.1
OPEN 12.2
READ 12.2
REWRITE 12.2
START 12.2
STOP RUN 7.2
Use:
ACCEPT F11-1
DISPLAY F11-1
WRITE 12.2
Status:
File I/O 12.2
Table, File T12-1
Value, File I/O 12.3
Stop, Address 7.2
STOP RUN Statement 7.2
Stream Format, Batch 2.3.4.1
Structure:
Alternate Directory 3.3
Directory 2.4, F2-1
File 2.4, F2-1
Job 2.2
Overlay 5.6.6, F5-14
Subroutine 1.2
Assembly Language 8.3
Calling 8.1
COBOL FD-1, FD-2
Debugging, Assembly Language 7.3
Library Package, COBOL 8.2, D.1
Subroutines, COBOL Library TD-1
Suppression, Blank 2.7.1.1
Synonym 1.2, 2.6
Synonym as a File Name 2.6
Synonym Availability With a:
Background Task 2.6
Foreground Task 2.6
System:
Capabilities 2.1
Error Messages B.2
COBOL Compiler TB-2
Overview 1.1

System Command Interpreter
(SCI) 1.1, 2.2.1, 2.3, 2.3.3

Table:
Device Correspondence T12-4
File Status T12-1
Tagged Format 5.5
Tailored SCI Procedure F10-2
Task 1.2, 2.2
Background 2.2
Foreground 2.2
Segment 5.4.1.1
Synonym Availability With a:
Background 2.6
Foreground 2.6
TASK SCI Command 5.6.5
Termination Messages 6.6
Text Editor 3.5
TIFORM 9.2
VDT Screen Description F9-1
Tools, Productivity 9.1
Two Procedures With:
Multiple Tasks, Linking 5.6.5, F5-12
Single Task, Linking 5.6.4, F5-11
Type, File 2.7

U Debug Command 7.2.2.9
Undo Address Stop (U) Debug
Command 7.2.2.9
Use:
ACCEPT Statement F11-1
Batch 2.3.4
Declarative 12.4
DISPLAY Statement F11-1
User:
Error Messages B.1
COBOL Compiler TB-1
Interface, DBMS-990 . . . . . 9.4.2
Program File, Linking Single
Procedure With Single
Task 5.6.9, F5-17
Using ALLOCATE, Linking F5-13
Using Subroutines, COBOL Program . . . D.17

Value:
Default 1.3.3.2
File I/O Status 12.3
Initial 1.3.3.1
VCATALOG 2.4.2
VDT:
Mode 2.3.2
Screen Description, TIFORM F9-1
Volume Name 2.4.1

W Debug Command 7.2.2.10
With Error Messages, COBOL
Compiler Listing FB-1
With P2, Sharing Main Program
Module F5-15
Within Overlay Phase Modules,
COBOL Segmentation F5-8
WRITE Statement 12.2
Write Screen to Message File
(W) Debug Command 7.2.2.10

XB SCI Command 2.3.4
XCC SCI Command 4.2.2
XCCF SCI Command 1.2, 4.2.1
XCP SCI Command 6.2.2
XCPF SCI Command 1.2, 6.2.1, 7.2.1
XCT SCI Command 6.4.2
XCTF SCI Command 1.2, 6.4.1, 7.2.1
XE SCI Command 3.5
XLE SCI Command 1.2, 2.3.4.1

.S$SHARED Program File 5.4.1.2
.S$SYSLIB.RCBMPD 5.6.1
.S$SYSLIB.RCBNOIO 5.6.1
.S$SYSLIB.RCBPRC 5.6.1
.S$SYSLIB.RCBTSK 5.6.1
.S$SYSLIB.RCBTSKD 5.6.1

CUT ALONG LINE

USER'S RESPONSE SHEET

Manual Title: DX10 COBOL Programmer's Guide (2270521-9701)

Manual Date: February 1985 **Date of This Letter:** _____

User's Name: _____ **Telephone:** _____

Company: _____ **Office/Department:** _____

Street Address: _____

City/State/Zip Code: _____

Please list any discrepancy found in this manual by page, paragraph, figure, or table number in the following space. If there are any other suggestions that you wish to make, feel free to include them. Thank you.

Location in Manual	Comment/Suggestion
_____	_____
	_____
	_____
	_____
_____	_____
	_____
	_____
	_____
_____	_____
	_____
	_____
	_____

NO POSTAGE NECESSARY IF MAILED IN U.S.A.
FOLD ON TWO LINES (LOCATED ON REVERSE SIDE), TAPE AND MAIL

FOLD

NO POSTAGE
NECESSARY
IF MAILED
IN THE
UNITED STATES

BUSINESS REPLY MAIL

FIRST CLASS PERMIT NO. 7284 DALLAS, TX

POSTAGE WILL BE PAID BY ADDRESSEE

TEXAS INSTRUMENTS INCORPORATED
DATA SYSTEMS GROUP

ATTN: TECHNICAL PUBLICATIONS
P.O. Box 2909 M/S 2146
Austin, Texas 78769

FOLD